AALBORG UNIVERSITY

DENMARK

P10 Project
Mathematical Engineering

The EMP-EMCI Interface

Developing Software for an Interface between the Atlas Detector and the Detector Control System at CERN

Author:
Kevin Graversgaard Jepsen

Supervisors:
Jan Østergaard

CERN Supervisor:
Paris Moschovakos
Vladimir Ryjov

June 7, 2024

This project is written in LaTeX.

Mathematical Sciences - Aalborg University
Skjernvej 4A, 9220 Aalborg Ø
http://math.aau.dk

AALBORG UNIVERSITY

STUDENT REPORT

Theme:
The EMP-EMCI Interface

Title:
Developing Software for an Interface between the Atlas Detector and the Detector Control System at CERN

Project Period:
Spring Semester 2024

Project Group:
MAT-TEK 10, Kevin Jepsen

Participants:
Kevin Graversgaard Jepsen

Supervisors:
Jan Østergaard

Copies: 1

Number of Pages: 31

Date of Completion:
June 7, 2024

Abstract:

The Atlas experiments is the largest experiment among the four major experiments on the Large Hadron Collider ring at CERN. The experiment features the Atlas particle detector, which is able to record up to 100 million data points 40 million times per second, with its many sensors positioned in concentric cylinders around a 46m section of the particle accelerator pipe. The conditions of these sensors needs to be constantly monitored to ensure proper operation of Atlas. To perform this monitoring, large amounts of data needs to be moved from the sensors to the control room, which require radiation hard interface devices. These interfaces concentrate the upstream diagnostic data, and propagate system parameter changes downstream. The EMP-EMCI system is such an interface device in development at CERN. In this project software is developed for the EMP, that allows a user to interface with the many functionalities on the EMCIs application-specific integrated circuit, called the low power Gigabit transceiver.

Preface

This project is written by Kevin Jepsen during the 4th semester of the Master of Science in Mathematical Engineering at the Department of Mathematical Sciences, Aalborg University. The project is written in collaboration with CERN, as an internship was taking place during writing. The time period for the project is the 1st of February 2024 to the 7th of June 2024.

Readers of this project are expected to have knowledge about the basic concepts of digital electronics and communication protocols.

The illustrations used in this project have been created using the TikZ-package in LaTeX. Many illustrations were also downloaded from the CERN document server, `https://cds.cern.ch/`, which are provided free of charge for educational use.

Thanks to the supervisor Jan Østergaard, as well as the CERN supervisors Vladimir Ryjov and Paris Moschovakos for support and guidance throughout the project.

Aalborg Universitet, June 7, 2024.

Contents

1 Introduction **1**
- 1.1 CERN . . . 1
- 1.2 The ATLAS Detector System . . . 2
  - 1.2.1 The Inner Detector . . . 2
  - 1.2.2 Calorimeters and the Muon Spectrometer . . . 3
- 1.3 Atlas Detector Control System . . . 4

2 The EMP-EMCI System **6**
- 2.1 The Embedded Monitoring and Control Interface . . . 6
- 2.2 The Embedded Monitoring Processor . . . 7

3 Low Power Gigabit Transceiver **9**
- 3.1 Basics of Digital Electronics . . . 9
  - 3.1.1 Voltage Levels and the Power Supply . . . 10
  - 3.1.2 Pull Resistors . . . 10

4 General Purpose Input/Output **11**
- 4.1 The lpGbt GPIO . . . 11
  - 4.1.1 Basic Functionalities of the GPIO . . . 11
  - 4.1.2 Advanced functionalities of the GPIO . . . 12
- 4.2 Software Architecture of the lpGbt GPIO . . . 13
  - 4.2.1 GPIO Demonstrator . . . 14
- 4.3 GPIO Testing . . . 15
  - 4.3.1 Software Tests . . . 15
  - 4.3.2 Hardware Tests . . . 15

5 Inter-integrated Circuit **20**
- 5.1 Physical Layer . . . 20
- 5.2 The I^2C Protocol . . . 21
  - 5.2.1 Start and Stop Conditions . . . 21
  - 5.2.2 Address Frames . . . 22
  - 5.2.3 Data Frames . . . 23
  - 5.2.4 Bus Speed . . . 23
- 5.3 LpGbt I^2C Interface . . . 24

6 Conclusion **26**

Bibliography **26**

Image references **28**

Appendices **30**

A Copyright of CERB Document Server **31**

1 | Introduction

This project has been written in collaboration with CERN, as the author was working in an intern position at CERN at the period of writing. The objective of this project is to develop an interface. An interface is a shared boundary which multiple, but separate, components communicate across. An input/output (I/O) device is a hardware interface, while every program shown on a computer screen is a computer-human interface, i.e., a graphical user interface (GUI).

1.1 CERN

The Conseil Européen pour la Recherche Nucléaire (CERN) is an international organisation, whose missions is to uncover what the universe is made of and how it functions [10]. CERN was founded in 1954 and has since been responsible for many scientific achievements, such as the discovery of the Higgs boson, creation of the first antimatter atom (antihydrogen) and the invention of the worldwide web [9, 7, 8]. To make these discoveries, engineers and physicists from around the world collaborate at CERN to create some of the worlds largest and most complex scientific instruments. Among these instruments are the particle accelerators, which aim to collide particles together at incredible speeds. These collisions are then directed to occur inside detector systems, which use varying methods to detect the particles resulting from the collision. The largest accelerator at CERN is the Large Hadron Collider (LHC), which can be seen in Figure 1.1.

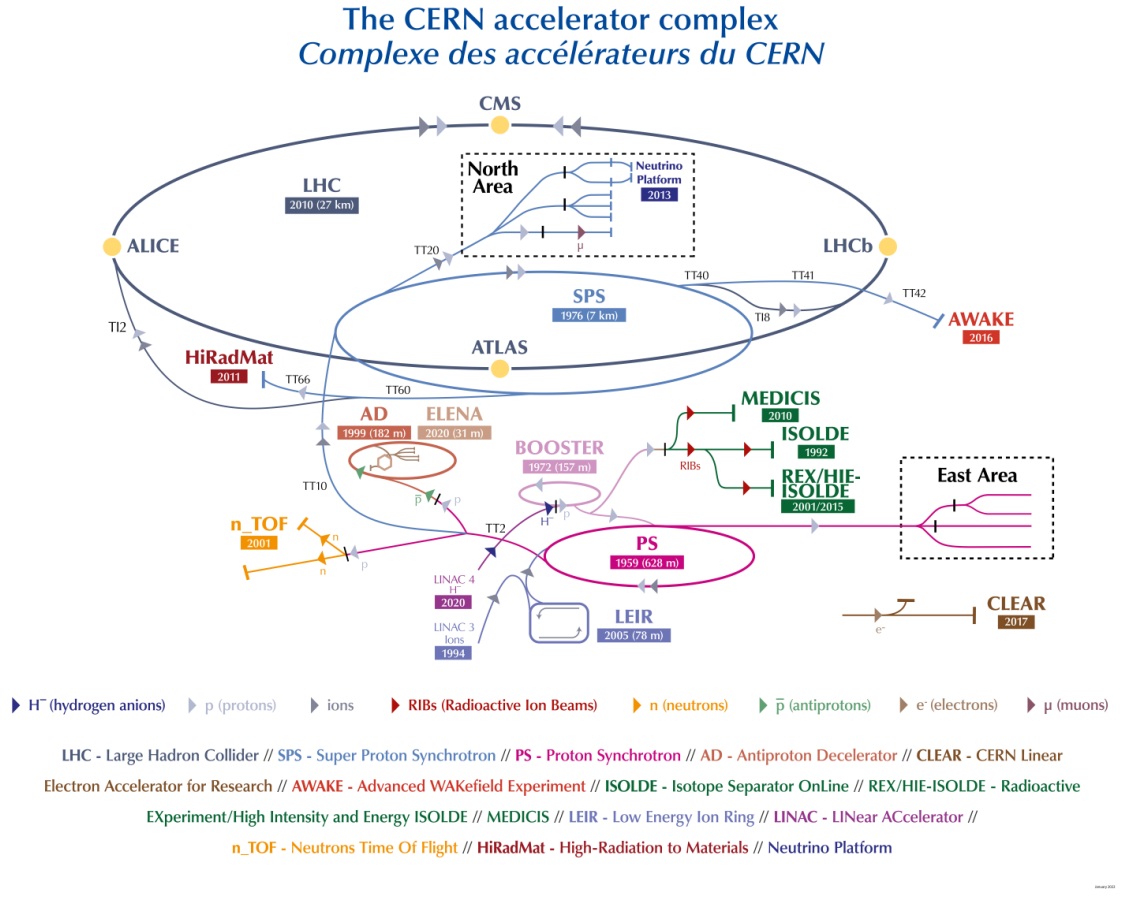

Figure 1.1: The accelerator complex at CERN, [image: 21].

The LHC features four detector systems, all of which it provides with a constant stream of particles, commonly referred to as beams. Of these four detectors, this report is conCERNed with the ATLAS detector system, as the contribution to CERN outlined in this report was done within the ATLAS experiment.

1.2 The ATLAS Detector System

This section is primarily based on [4, ch. 4] and [5]. For a detailed overview of the Atlas detector see [1]

The Atlas experiment is one of the two largest ongoing experiments at CERN and its associated collaboration consists of around 6000 members and 182 institutions situated in 42 different countries [13]. Together with the CMS experiment, Atlas discovered the Higgs Boson back in 2012.

The Atlas particle detector consists of 6 detector subsystems which are arranged as concentric cylinders around the collision point, [14]. These subsystems are commonly categorized as

- The inner detector system, which measures cross direction, speed and charge of electrically-charged particles produced in the collisions.
- The calorimeters, which absorbs and measures the energy of particles passing through them.
- The Muon spectrometer, which is specialised in detecting Muons.

A slice of the Atlas detector is shown in Figure 1.2, which specifies the 6 subsystems and the two types of magnets used in Atlas.

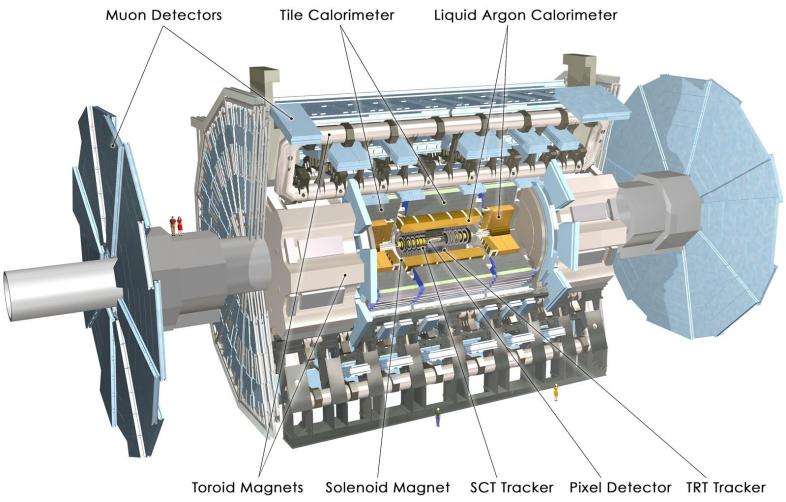

Figure 1.2: Computer illustration of the Atlas detector, [image: 22].

The superconducting magnets play a key role in the detector as they create a magnetic field of up to 4 Tesla, that bends the paths of the particles resulting from the collisions outwards. This strong bending makes it so that the particles fly through the cylindrical detector, rather than continuing in the direction of the beam pipe, at a slight obtuse angle.

1.2.1 The Inner Detector

The inner detector, also often called the tracking detector, consists of three subsystems. These subsystem all have barrel and end-cap detector parts, which are placed along the cylinder body or at the end points of the cylinder, respectively. Figure 1.3 shows the layering of these three subsystems.

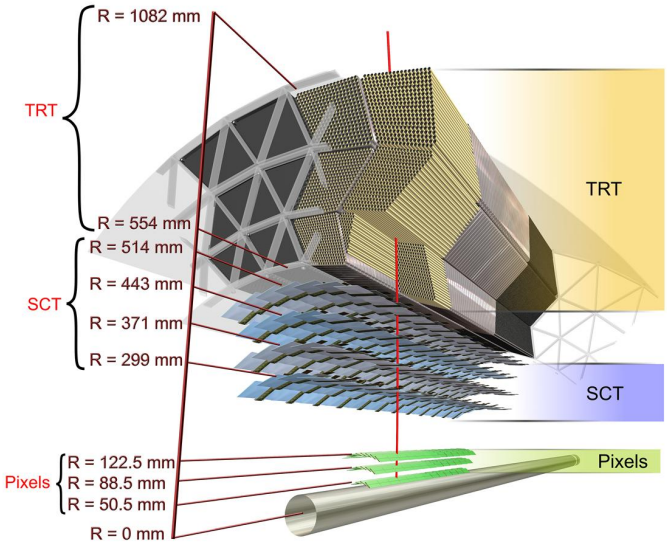

Figure 1.3: Slice of the Atlas inner detector subsystems, [image: 23].

The first subsystem is the pixel detector, which consists of 92 M semiconductor elements called silicon pixels, each of which becomes briefly charged when an electrically charged particle passes through it. In the barrel these pixels are packed in four layers around the collision point in 'sheets' of approximately 1.9m^2, hence the size of each pixel is extremely small with sizes ranging from 50×250 to 50×400 [μm^2]. This fine granularity of the sheets allows the pixel detector to detect a particles position within a precision of 10 μm, and the layering makes it possible to estimate the particles speed and trajectory.

The second subsystem is the semiconductor tracker, which consists of more than 4000 modules of micro-strip detectors made of silicon sensors. Together these modules features over 6M silicon sensors, which are spaced 80 μm apart on each module. Similar to the pixel detector, modules of the semiconductor tracker for the barrel are packed into sheets which are then formed into cylinders and layered, but this time the innermost sheet also surrounds the pixel detector.

The third subsystem is the transition radiation tracker, which consists of 300.000 thin tubes that each contain gas and a thin 30 μm tungsten wire in the center. When a charged particle passes through a tube it ionises the gas, creating an electrical charge that the wire can pick up. This allows for reconstruction of the particles track. The transition radiation tracker is the outermost subsystem of the inner detector, and thus wraps around both the pixel detector and semiconductor tracker.

The inner detector handles the tracking of particle paths, but the particles still need to be identified. This task is handled by the remaining 2 subsystems of the Atlas detector.

1.2.2 Calorimeters and the Muon Spectrometer

After particles have passed through the inner detector they reach the calorimeters, which are designed to absorb most of the particles resulting from the collisions. The calorimeters converts the particle into showers of lower energy particles, which are inevitably absorbed. This can be seen in Figure 1.4, which also shows which of the two calorimeters absorbs what particles.

Particles from the collision first reach the liquid Argon (electromagnetic) calorimeter, which fully absorbs photons and electrons while also absorbing some of the energy of hadronic particles, such as protons or neutrons. The layers of this calorimeter are made of metals.

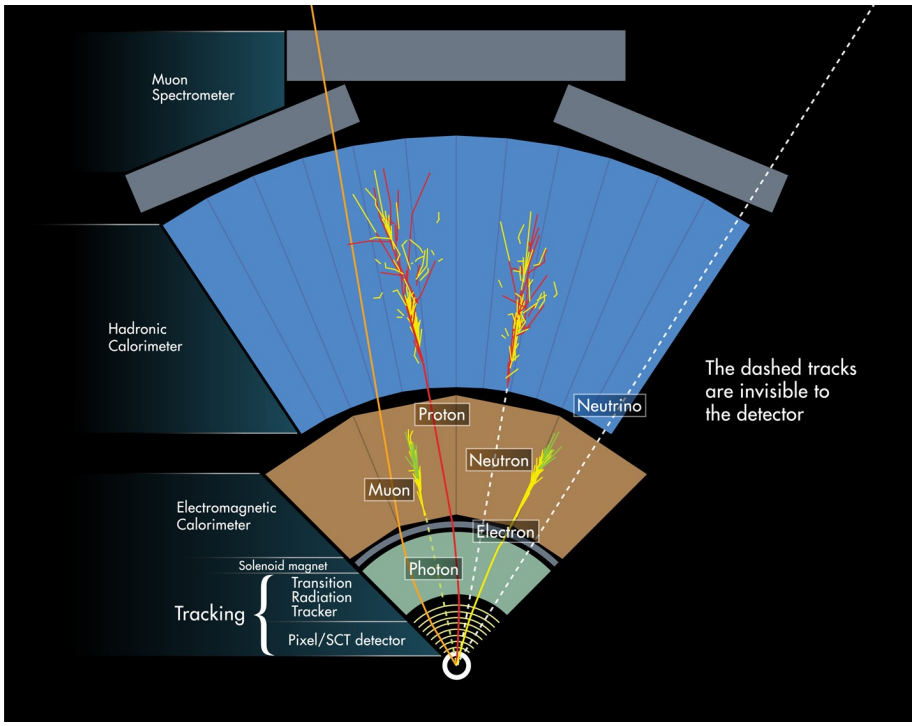

Figure 1.4: Illustration of where each fundamental particles stops inside Atlas, [image: 24].

Particles that can pass through the electromagnetic calorimeter, then reach the tile hadronic calorimeter, which absorbs hadronic particles. This calorimeter is made of steel and stops all but muons and neutrinos.

After the two calorimeters, any remaining particles pass through the muon spectrometer, which consists of many resistive plates located at the outermost part of the detector barrel, combined with the large 'wheels' at either end of the Atlas detector. The spectrometer does not absorb the muons, but it can detect their position to an accuracy of 10^{-4} m.

The last remaining particle from Figure 1.4 is the Neutrino, which cannot be detected directly by any of the subsystems in Atlas. Instead its presence has to be inferred from predictive models and the data of the collision gathered by Atlas [4, ch. 4, p. 5]. The reason that none of the other existing particles are mentioned, is that they decay too fast to be detected. The muon decays in about 2.2 μs, which allows it to travel around 600m before disappearing. In contrast the tau lepton, only travels around 1/10th a millimeter before decaying [4, ch. 4, p. 6], which is less than 0.3 % of the distances needed to reach the innermost layer of the pixel detector, located a 3.3 cm from the collision point.

With its many subsystems the Atlas detector can record up to 100M data points 40M times pr. second [4, ch. 4, p. 1]. This amount of data is required to reconstruct the collisions, which the LHC can produce 1.7 billion (10^9) of each second [12]. Such a complex machine requires a lot of control and monitoring, a task handled by the Atlas detector control system (DCS)

1.3 Atlas Detector Control System

The Atlas DCS enables supervision of all equipment in all Atlas sub-detectors. Its task is to "permit coherent and safe operation of ATLAS and to serve as a homogeneous interface to all sub-detectors and the technical infrastructure of the experiment" [5, p. 1]. The DCS must be able to bring the detector into any desired operation state, manage all of Atlas 10^6 operational parameters, and monitor the operation of the detector including identifying any abnormal behavior. The DCS departments at CERN also develop the tools that the people in the Atlas control room use to monitor and configure the systems of the detector. This includes the electronics that continuously

carries diagnostic data from all the front-end (FE) sensors, to the back-end (BE) systems used by the control room.

In the pipeline that moves the data upstream, from the FEs to the BE, the data is usually also funneled into into electronic devices that are specialised in moving the data. One such electronic device is the Embedded Local Monitoring Board (ELMB), which is a low-cost, I/O concentrator device that is tolerant to both the strong magnetic fields and ionizing radiation present in many of the experiments at CERN. More than 5.000 ELMBs are used in Atlas and over 10.000 are in use between all the LHC experiments [5, p. 2].

The experiments at CERN are always improving. One big upcoming improvement of the LHC is the high-luminosity (HL) upgrade, which is expected to be implemented between 2026 and 2028 during the Long Shutdown 3 [11]. Luminosity is a measure of a particle accelerators performance, as it indicates the expected number of collisions pr. cross sectional area pr. second, giving it the unit $cm^{-2}s^{-1}$. The HL upgrade is expected to increase the luminosity of the LHC by a factor of 5x-7.5x. The luminosity is also projected to increase further in after the HL upgrade [5, p. 4].

The electronic devices in the data pipeline from the FEs to the BE has to continually keep up with the increased demand for radiation tolerance, imposed by the luminosity upgrades. Thus a new I/O concentrator system is being developed for Atlas, which is radiation tolerant enough to be used after the HL upgrade. It is called the EMP-EMCI system, as it comprises two devices, and is being developed to complement the current generation of ELMBs.

As this project is focused on a small part of a much bigger system, the problem statement is deferred to the end of the next chapter, where the EMP-EMCI system is discussed.

2 | The EMP-EMCI System

This section is based on [15] and [19], as well as the current specifications of the EMCI [2] and the EMP [3], with the latter being an internel document at the time of writing.

The EMP-EMCI system compose an interface between the front-end sensors (FE) of a detector and the related backend control system. For use in the Atlas experiment, the front-ends will be the subsystems of the Atlas detector described in Section 1.2, and the backend will be the detector control system (DCS). The EMP-EMCI system consists of the Embedded Monitoring Processor (EMP) and the Embedded Monitoring and Control Interface (EMCI). It is being developed to meet the increasing requirements in radiation tolerance, imposed by the HL upgrade for the LHC.
The EMCI is capable of communicating with several FEs, and will be placed in a radiation environment as it is developed to have a high radiation tolerance. The EMP will interface with up to 12 EMCIs and be placed in a non-radiation environment, performing the data packaging and processing that prepares the data to be transmitted to the backend. A diagram of the EMP-EMCI system and related components can be seen in Figure 2.1.

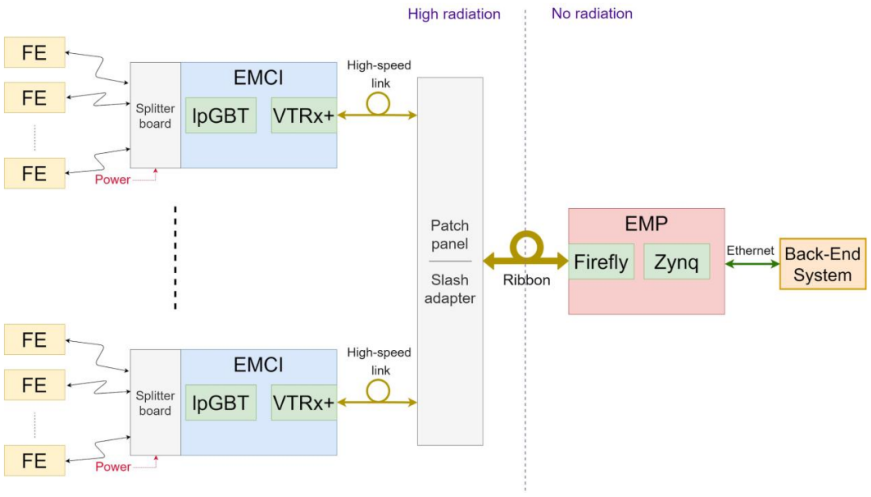

Figure 2.1: Box diagram of the EMP-EMCI system, [image: 2, p. 5].

2.1 The Embedded Monitoring and Control Interface

The EMCI is a slow control module, in the sense that its function is not time-critical, developed at CERN for the HL-LHC experiments [2, p. 1]. It is supposed to be a bidirectional interface that compliment the current generation of the ELMB, in radiation areas that exceed the ELMBs tolerance.

The heart of the EMCI is the low power Gigabit transceiver (lpGbt), which combines all the data, incoming from the connected FEs, and forwards it to the EMP through an optical link cable. The EMCI-side of the link is handled by the versatile transceiver+ (VTRx+), which is an optical transceiver developed at CERN. A detailed block diagram of the EMCI can be seen in Figure 2.2. The figure highlights the functionalities of the lpGbt, some of which will later be discussed. It also shows the components that surrounds the lpGbt on the EMCI, some of which deserve a short description.

The EMCI communicates with the FEs through eLinks, which carry low-power differential signals [2, p. 15]. The EMCI can either receive data from up to 28 devices or transmit data to up to 16 devices, at any one time. The maximum bandwidth at which the EMCI can communicate with the

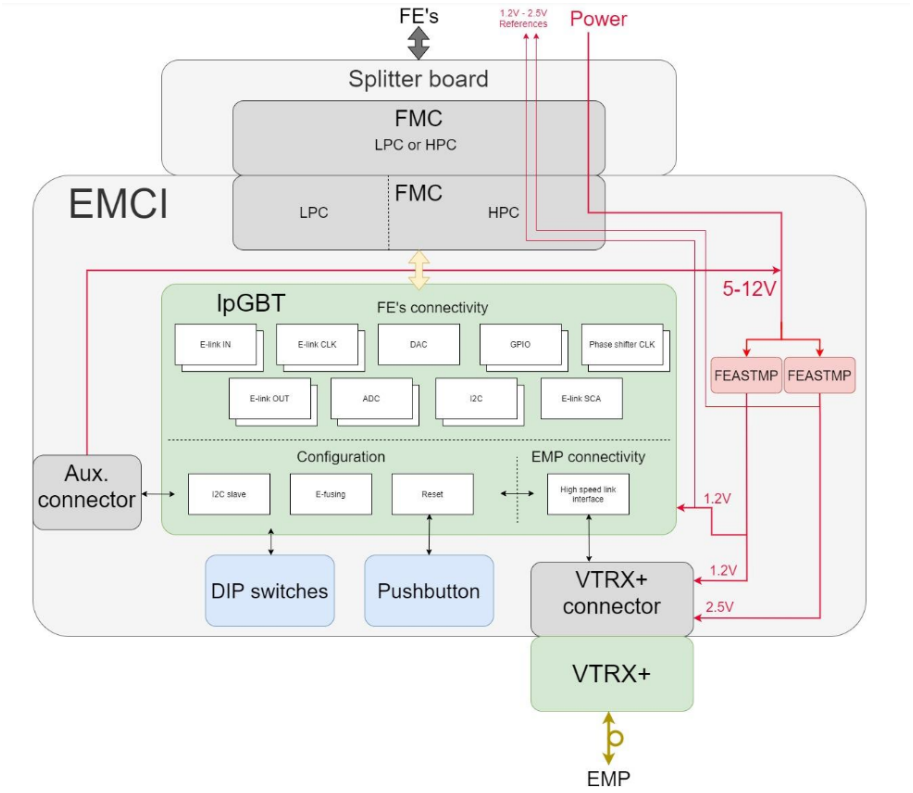

Figure 2.2: Diagram of the EMCI that shows the lpGbt and its surrounding components [image: 2, p. 15].

FE devices depends on the amount of devices connected, as well as the type of error correction used. The highest downlink (EMCI to FE) bandwidth is 1.280 Gb/s, and the highest uplink (FE to EMCI) is 8.960 Gb/s. The bandwidth is a resource which has to be split equally between all Elink connections. This fact makes the EMCI quite versatile, as the user can use fewer eLinks connections if a high data rate is needed. Data is moved from the EMCI towards the backend by the VTRx+, which connects the lpGbt to the EMP. The VTRx+ can transmit data at a rate of 10.24 Gb/s, and receive data at 2.56 Gb/s. Both the VTRx+ and the Elinks have higher data rates in the upstream direction, because the main function of the system is to forward the data from the FEs to the backend. The downstream direction is only used for moving configuration data.

The EMCI also features two DCDC devices, called FEASTMPs, which convert the input voltage level to 1.2V and 2.5V, respectively. These modules are radiation tolerant and have been developed at CERN. It should be noted that the FEASTMPs takes up a small amount of the lpGbt resources, as 2 of the GPIO pins have been dedicated controlling these devices.

2.2 The Embedded Monitoring Processor

The EMP is a board that houses a Xilinx Zynq Ultrascale+ multiprocessor system on chip (MPSoC), together with other components like transceivers, a FPGA mezzanine card (FMC) connector and an ethernet network interface [3, p. 5]. The Zynq is able to use both programmable logic and software, as it features a ARM multi-core processing system. Control of the EMP can be done through the ethernet interface, which will be for communication with the backend. A picture of the EMP can be seen in Figure 2.3. The large black box is the Zynq and to the right of the Zynq are a pair of FireFly transceivers partly covered by a fan.

Besides the Zynq, the Fireflys are one of the most important components on the EMP, because they handle the communication with the EMCIs. The Fireflys are a custom design, developed to be used at CERN, but are inspired by the Samtec UEC5 and UCC8 models. The pair functions as

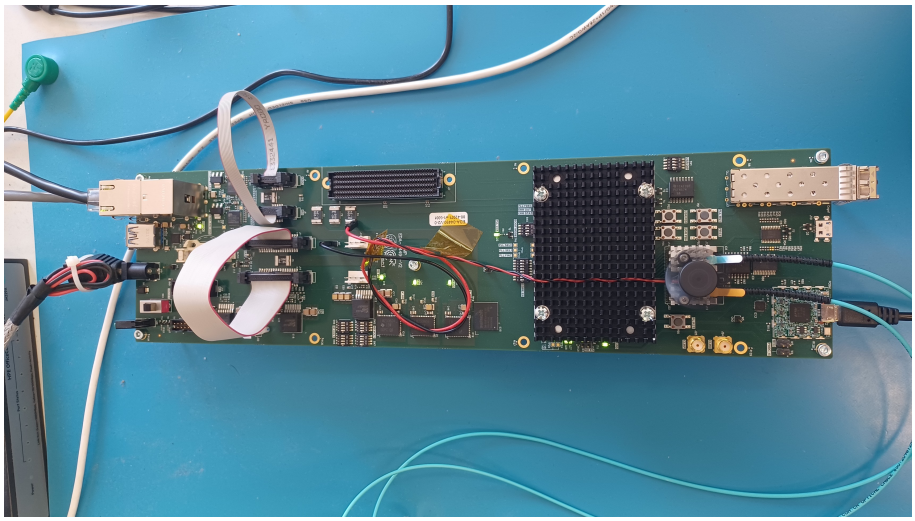

Figure 2.3: A picture of the physical EMP.

a transceiver, by having one be a transmitter and the other a receiver. Since the EMP can connect with up to 12 EMCIs and each VTRx+ can transmit up to 12.24 Gb/s, the Fireflys have to be able to receive almost 147 Gb of data pr. second among their 12 channels. This is, surprisingly, not a problem as the Fireflys can handle a data rate of up to 20 Gb/s pr. channel [17].

The EMP is used be used to communicate with the lpGbt and will house the software that can change the settings on the lpGbt. Thus all software mentioned in this report will run on the EMP but be used to configure the lpGbt on the EMCI. Hence this software library being developed is called lpGbtSw.

Problem Statement

How can software be developed for the EMP-EMCI system, that makes it easy for the user to configure the lpGbt, while keeping the reduction in flexibility of the lpGbts settings to a minimum.

3 | Low Power Gigabit Transceiver

The heart of the EMCI is the low power Gigabit transceiver (lpGbt), which is a radiation tolerant and application-specific integrated circuit (ASIC) developed at CERN [6]. It's purpose is to provide a high speed data link between the front end sensors in the detectors and the back end control system. Some of the internal circuits and intended connections of the lpGbt are illustrated in Figure 3.1.

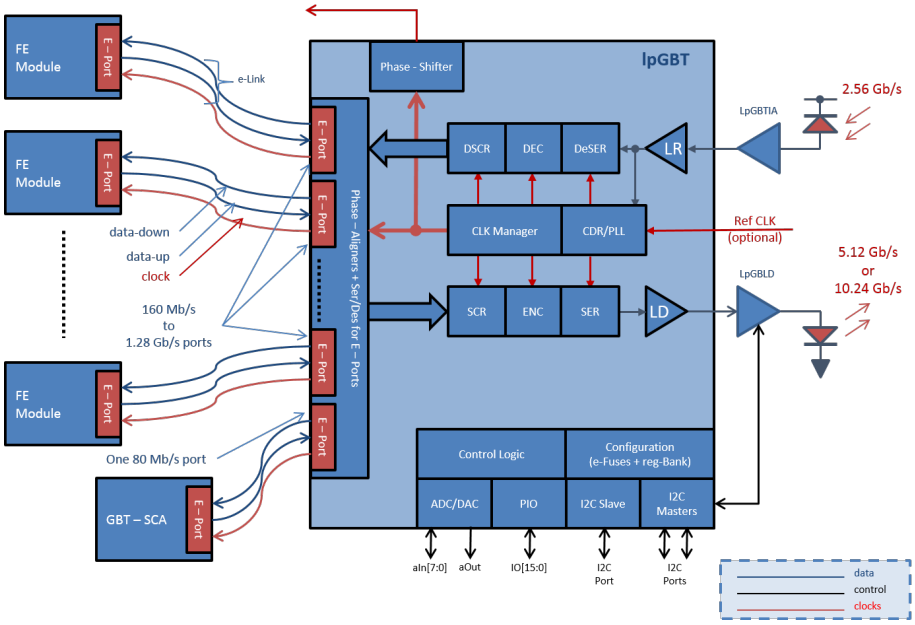

Figure 3.1: Architecture of the lpGbt, [image: 6, p. 5].

The lpGbt features a lot of possible settings related to the electrical circuits that it contains, such as ADC/DAC, GPIO and I^2C. These settings can be controlled through a set of byte-sized registers [6, p. 18], of which there are 493 in total. The EMP can read from or write to these registers by using the commands

writeReg(Address, byte), readReg(Address)

in the software that runs on the EMP. As this report is conCERNed with designing an interface which can configure the various circuits on lpGbt, some basic theory on circuits and digital electronics is introduced.

3.1 Basics of Digital Electronics

In analog electronics continuous signals such as current or voltage are utilized directly. Contrary to this, digital electronics are conCERNed with binary states represented by 1s and 0s, hence the countinuous signals has to be quantized into either of these binary states. This is usually done on the voltage, with +5V representing a High state and 0V a Low state, i.e., 1 and 0 respectively. However, the exact voltage values can depend on the application. Sometimes individual bits are used to control binary settings in a circuit, that is, settings which only have 2 states. The states are most commonly On and Off, but could be anything that is binary in nature. When a bit controlling such a setting is High, it is said to be 'set'.

The advantage of using digital electronics is that it allows for easier storage of information and use of logic, i.e., circuits build on logic gates [18, p. 717-718]. Digital electronics are also highly

flexible, as the ease of data storage entails an easy way to reprogram the circuits, and they are also inherently noise robust, due to the use of quantization and the ability to use error-correcting codes.

3.1.1 Voltage Levels and the Power Supply

Circuits are powered by a power supply, which acts as a voltage source for the circuit. Since the circuit is a load that is connected to the voltage source, it draws an amount of current determined by its resistance due to Ohms law,

$$I = \frac{V}{R}.$$

Different electrical components can vary their resistance, which in turn changes the amount of current drawn when the voltage has to be kept constant. One of the simplest examples is a switch, which has a very high resistance when open, but drops to a lower level when the switch is closed. The voltage level that different component receive can also be varied by using resistor in series, or kept the same locally by instead using resistors in parallel.

To transmit data in a digital circuit, current has to be sent through a conductor, which is simply called the line unless labeled otherwise. Digital devices connected to the line then measure the voltage on the line, by comparing it to a local reference which is usually a ground (GND). If a device is connected to voltage source separate from the communication line, it can connect it to the line which is called driving the line High. Similarly if a device can connect the line to a GND, it is called driving the line Low.
Another used method is to have the line being driven by a voltage source, and then allowing connected devices to pull the line Low on demand. A circuit using this output strategy is called open drain, as it allows the connected devices to 'drain' the voltage from the line. This type of communication circuit will be discussed in Chapter 5, as it is commonly used in I^2C communication. Sometimes lines in digital circuits are not driven, in which case pull resistors are commonly used to enforce a known logic state on the line.

3.1.2 Pull Resistors

When a communication line is undriven it is said to be floating. A floating line could results from the line being connected to unused pins on a printed circuit board (PCB). As digital circuits are designed to function at low current, it is possible for electromagnetic noise to induce enough currents in an undriven line to drive it High temporarily, which is ofcause undesired. To counter this behavior an open line can be driven either High or Low by connecting it to a voltage source or ground, respectively. However, this also makes the line unusable as trying to drive the line to the opposite state would result in a path from a voltage source to a ground, i.e., a short circuit. If a resistor is added between the line and the source or ground, the line can still be actively driven, while also remaining at a known state when undriven. This type of connection is called a pull resistor, as the resistor allows the line to be 'pulled' High or Low safely. If the resistor is combined with a voltage source it is called a pull-up resistor, and if it is combined with a ground it is called a pull-down resistor [18, p. 779]. Pull resistors are also commonly used in combination with switches, as this allows them to be enabled or disabled. Switches also allow a line to feature both types of resistors, as they will not counteract each other.

Equipped with these basic concepts from digital electronics, some of the circuits on the lpGbt can now be discussed.

4 | General Purpose Input/Output

A general purpose input/output (GPIO) is a peripheral component that can connect a controller unit, like an IC, to several other devices through a set of two-way channels. The lpGbt features a 16 pin GPIO with some simple functionality that can be controlled by the lpGbts registers.

4.1 The lpGbt GPIO

All the settings of the GPIO are binary, which means that each pin only requires a single register bit per setting. Since the GPIO features 16 pins and each lpGbt register contains 8 bits, it possible to represent a setting across all pins with 2 registers. This grouping is called a Port in the lpGbt manual, and the register which control pins 0-7 is called the Low Port while the other, controlling pins 8-15, is called the High Port [6, p. 89]. There are a total of 6 Ports responsible for controlling the 6 functionalities that the GPIO features. A diagram of the type of circuit that controls each pin on the GPIO can be seen in Figure 4.1.

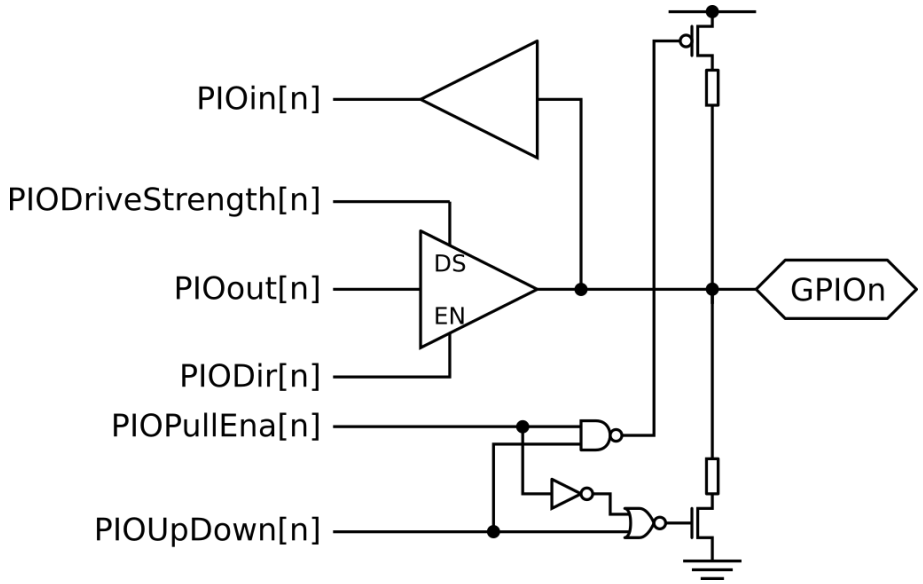

Figure 4.1: Circuit diagram of the GPIO, [6, p. 89].

The figure shows the 6 ports on the left, each of which determines the setting of a functionality for pin n. Thus it should be understood that the circuit in Figure 4.1 only controls a single pin, and thus is duplicated 16 times, once for each of the respective GPIO pins.
The GPIO functionality can be split into categories of basic and advanced as

$$\text{Basic functionality} = \{\text{input, output, direction}\},$$
$$\text{Advanced functionality} = \{\text{drive strength, pull resistors}\},$$

with each of these categories having 3 ports dedicated to them. A short description of the functionalities follows, starting with the basic functionalities.

4.1.1 Basic Functionalities of the GPIO

The basic functionalities are those that are core to the function of the GPIO. Together they control the communication on each pin.

The **input** functionality (PIOin) is used to read the current state of a pin. This allows the lpGbt to receive data from the peripheral components connected to the pins, by checking the state of the bits in the Port. It should be noted that the input functionality can always read the state of a pin, irrespective of whether the pin is being driving by an external device or by the pin itself.

The **output** functionality (PIOout) is used to transmit data on any pin. By setting the bit High in the lpGbt Port, the related pin will be driven High by the GPIO. Conversely setting the bit Low will drive the pin Low.

The **direction** functionality (PIODir) allows the lpGbt to control the data direction of the line between a pin and its GPIO controller, effectively determining if the pin should act as an input or an output. This is done by enabling or disabling the output driver. Table 4.1 shows the possible combinations of PIOout and PIODir, and what pin state results from these.

PIODir	PIOout	Pin state
0	0	X
0	1	X
1	0	0
1	1	1

Table 4.1: Truth table showing the state of a pin, given the binary states of its direction and output bits. X implies that the pin state is determined externally.

From the table, the purpose of the direction functionality is clear, as a GPIO pin cannot act as an input while PIOout is actively driving it.

4.1.2 Advanced functionalities of the GPIO

The **drive strength** functionality (PIODriveStrength) allows the lpGbt to control the rise time of the signals on the GPIO pins, also known as the slew rate. When a bit the this Port is set, the drive strength is set to High, which allows for faster communication on the GPIO pins at the trade-off of increased electromagnetic radiation [6, p. 90]. The ability to increase the drive strength is also relevant if the device connected to the pin imposes high load capacitance, as this can increase the voltage rise time resulting in slower communication.

The **pull resistor** functionality (PIOPullEna & PIOUpDown) allows the lpGbt to pull floating pins to a known, and constant, state. The GPIO circuit features both a pull-up and pull-down resistor, as can be seen in Figure 4.1. Similarly to the output and direction functionality, this gives more than 2 possible configuration for the pull resistor functionality, hence 2 Ports are dedicated to the control of this functionality. The state of these Ports, for a given pin, are then directed to a small logic circuit, that disconnects at least 1 of the resistors. Table 4.2 shows which resistor(s) is disconnected for all possible combinations.

	PIOPullEna	**PIOUpDown**	**Pull down resistor**	**Pull up resistor**
	0	0	OFF	OFF
	0	1	OFF	OFF
	1	0	ON	OFF
	1	1	OFF	ON
Logic	a	b	NOR(NOT(a), b)	NOT(NAND(a, b))

Table 4.2: Truth table for the pull resistor switches in the GPIO.

The table also features the logic imposed by the circuit in the last row.

With the features of the lpGbts GPIO explained, the design of the software on the EMP, that is able to set the register values in the lpGbt that control the GPIO, can be discussed.

4.2 Software Architecture of the lpGbt GPIO

A core design ideal of the LpGBtSw is to keep it as simple as possible. The software should, as best as possible, only feature basic interfacing functionality, that future users can then use to build their own software, which the users can then decide to shape more to their requirements. The GPIO circuit is quite simple and hence it was possible to implement the software in this manner.

The GPIO software is split into a back-end, consisting of functions which are private to the Gpio class in cpp, and a front-end, consisting of public user functions, that then call functions from the back-end. As all functionality of the GPIOs is controlled by bit settings, most front-end functions use get/set functions to retrieve or transmit register values. The register values are changed between these two operations, by a bit changer function that picks out the bit corresponding to the target pin, and sets its state to reflect the desired functionality.
The public user functions that were developed are listed in Table 4.3, together with their allowed inputs and the GPIO Ports that they interface with.

User function	GPIO Port	Function inputs
readPin	PIOin	pinNumber
driveOutputPin	PIOout	pinNumber, voltageLevel
setPinDirection	PIODir	pinNumber, direction
setPinPullOption	PIOPullEna	pinNumber, pullState
setPinPullDirection	PIOUpDown	pinNumber, pullDirection
setOutputPinSlewRate	PIODriveStrength	pinNumber, driverStrength

Table 4.3: User functions developed to interface with the GPIO Ports.

In the table it can be seen that every user function takes the pinNumber variable as an input, which is the number of the target GPIO pin. This variable is assigned to the class PinNumber, which upon creation of the pinNumber variable creates the following member variables

$$\text{pin}, \qquad \text{port}, \qquad \text{pinAdj}. \tag{4.1}$$

The PinNumber class also includes a check, that will throw an exception if the given pinNumber does not corresponds to a physical pin, i.e., if it is not in the range $(0, 15)$. The member variables in (4.1) ensures that the correct register in the Port is picked and that the correct bit in this register is changed, by saving the pinNumber to pin, the Port type (Low/High) to port and the bit position in the correct register to pinAdj. A code snippet of the setPinDirection user function, which contains all the discussed software architecture, can be seen in Figure 4.2.

```
void Gpio::setPinDirection(PinNumber pinNumber, Direction direction)
{
    LOG(Log::INF) << "Setting the direction of the pin " << pinNumber.getOriginalPin() << " to "
    << std::string((static_cast<bool>(direction)) ? "OUTPUT" : "INPUT");
    Port port = pinNumber.getPort();

    RegisterValue registerValue = getPinsDirection(port);
    changeBit(registerValue, pinNumber.getAdjustedPin(), direction);
    setPinsDirection(port, registerValue);
}
```

Figure 4.2: Code snippet featuring the body of a front-end user function.

In the code snippet, it can be seen that a local port variable is first assigned with the value of a "class getter" function that accesses the port variable of the PinNumber class. After that the value of the register that controls the direction functionality of the target pin is fetched with a get() function. Then the target bit is changed to match the direction that the user declared, and the updated register value is written to the lpGbt with a set() function. The direction variable can be interpreted as a bit by the changeBit() function, because it belongs to an enum class and thus is related to an underlying number.

The software also features a demonstrator functionality, which allows the user to test the lpGbt functionality through the console.

4.2.1 GPIO Demonstrator

The demonstrator program allows a user to interface with the GPIO front-end functions. The user is expected to develop further software on top of the provided GPIO functionality, hence a short demonstration of how the front-end functions can be called was deemed useful. The demonstrator can be run in a console by putting flags after the command to execute the program, as

./lpGbtGpio -a lpgbt-uio://emp_lpgbt_3 -p2 -d output -w high.

The flag parsing is handled by the program_options component of the <boost> library, which looks for predefined flags in the input, and then saves both the flag and the following argument to a variable map. The possible flags and arguments can be seen in Table 4.4.

Flag	Shorthand	Input(s)	Functionality
--help	-h	none	see Figure 4.3
--address	-a	1-12	sets target lpGbt
--pin	-p	int	sets target pin
--read	-r	none	PIOin
--write	-w	low/high	PIOout
--direction	-d	input/output	PIODir
--slew	-s	low/high	PIODriveStrength
--pullEnable	none	off/on	PIOPullEna
--pullDirection	none	down/up	PIOPullDir

Table 4.4: Possible flags and arguments for the GPIO demonstrator.

Using the demonstrator a user can set any GPIO functionality for any pin on any connected lpGbt. The demonstrator has been made robust against invalid arguments and will print an error message if a user inputs a wrong argument for the given functionality. The <boost> library comes with build in error handling for the flags. The -h option also describes the allowed flags and arguments, as can be seen in Figure 4.3.

```
Options:
  -h [ --help ]                          show help
  -a [ --address ] arg (=lpgbt-simulator://emp_lpgbt_1)
                                         LpGbt address, one of:
                                         lpgbt-simulator://<>
                                         lpgbt-uio://<emp_lpgbt_num>


Demonstrator options:
  -p [ --pin ] arg (=0)  set the target pin that other settings affect [0-15]
  -r [ --read ]          read state of pin
  -w [ --write ] arg     set output state of pin [low/high]
  -d [ --direction ] arg set direction of pin [input/output]

Advanced demonstrator options:
  -s [ --slew ] arg     set drive strength of pin [low/high]
  --pullEnable arg      enable pins pull resister [off/on]
  --pullDirection arg   set direction of pins pull resister [down/up]
```

Figure 4.3: Console output for the demonstrator --help option.

With the software presented, the test that were used to validate the softwares functionality can be presented.

4.3 GPIO Testing

All functionality of the lpGbt is programmed by software, which the EMP interprets and translates to commands for the lpGbt. Thus it must be ensured that the software works, meaning that the commands affect the correct registers on the lpGbt, and also that the hardware works, meaning that the setting of a register value does what it is supposed to do. Thus both software and hardware tests are described. Not every functionality could be tested in both software and hardware, but between the two methods all functionality was tested, aside from the pull-down resistor which could not be tested due to lack of tools.

4.3.1 Software Tests

The objective of the software tests is both to see if the register mapping provided in the lpGbt manual is correct and to gauge whether the setting of the registers activates the expected functionality. In the software tests, a setting that affects the GPIO line for some pin is changed and the state of the line is then read using PIOin.

Output Test

In this test the functionality of PIOout and PIODir was tested. First PIODir was set to output, and it was observed that PIOout was able to correctly drive the line Low or High when the appropriate bit was set in the register. Afterwards PIODir was set to input and PIOout was set to drive the line High. The line was observed to be Low, which confirms that the output driver is correctly disabled when a pin is configured as input.

Pull Resistor Test

In this test the functionality of the pull resistors were tested. First the pull resistors were enabled and PIOUpDown was tested. It was discovered that the line was Low when the pull-up bit was set and vice versa for the pull-down bit. After correspondence with the lpGbt support team, this was concluded to be caused by an error in the manual, which at that time stated that

$$1'b0 \Rightarrow \text{pull up}, \qquad 1'b1 \Rightarrow \text{pull down},$$

i.e. the bit map was inverted. The bit combinations stated in Table 4.2 are the corrected ones, which both the manual and the lpGbtSw was updated to reflect.
Afterwards the pull resistors were disabled and it was observed that the line was Low when the pull-up bit was set. This indicates that the functionality to disable the pull resistors works correctly.

Output Driver Priority Test

As the main functionality of the pull resistors is to counteract electromagnetic noise in floating pins, it should be possible to drive the pins to the opposite state of an enabled pull resistor. This was tested by enabling the pull resistors and then pulling the line Low while driving it High and vice versa. In both cases it was found that the output driver could control the line despite the pull resistor being enabled and trying to pull against the output driver, which is the intended behavior.

4.3.2 Hardware Tests

The objective of the hardware tests, is to observe that all the software implemented GPIO functionality does what it promises to do, in the physical world. These tests require the use of different tools that can probe the physical GPIO lines on the lpGbt. As the lpGbt itself is quite small, all tests utilized a previously developed splitting board. This board was developed because the layout of the EMCIs FMC is custom, and hence a mapping between this layout and a standard FMC layout was required. The splitting board allows the the EMCI to be connected to an "AMD Virtex 7 FPGA VC707 Evaluation Kit" board, which was also used in one of the tests. Furthermore, the splitting board features external pins, which are large enough to connect cables to and probe

with a multimeter, both features that proved useful for the GPIO testing. The full setup with the EMCI connected to the VC707 through the splitting board, can be seen in Figure 4.4.

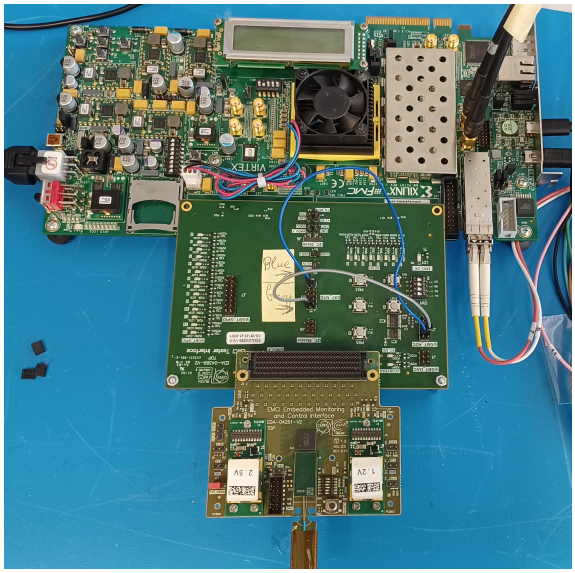

Figure 4.4: Full setup used for testing GPIOs of the lpGbt.

All functionality of the lpGbt GPIO module was tested, except the pull down resistor, as no tools were available to test this functionality.

Output Test

To test the output functionality, the signals from the GPIO pins were mapped to the VC707 and read back using Vivado. Vivado is a program that can translate VHDL code into a bitstream, which can then be loaded into a processing unit that understands programmable logic. The Field-Programmable Gate Array (FPGA) housed on the VC707 is such a processing unit. Vivado also has a tool called the internal logic analyzer (ILA), which allows for including probes in the bitstream, that tell the FPGA to monitor certain lines on the board and take a snapshot of these lines when a preset condition, called a trigger, occurs.

For this test all the GPIO pins were monitored, and the trigger was set to be a bit transition on pin 0. It was found that the correct output signal from any GPIO pin could be observed by the VC707 (receiver), when the setting to drive a pin High was configured in the software.
A picture of the Vivado ILA can be seen in Figure 4.5. In the picture pins 5 and 11 have pre-emptively been driven High and pin 0, the trigger pin, is being driven High, which is illustrated with the appearance of the horizontal green line right after the vertical red 'trigger' line.

Input Test

To test that inputs to the GPIO pins can be read by the lpGbt, the splitting board, which is the board seen in the middle in Figure 4.4, was used in combination with a set of jumpers. A jumper is a small piece of plastic with a metal plate inside, which can be placed between two adjacent pins to provide a connection between them. Since the GPIO pins are mirrored to a set of bigger pins on the splitting board, the jumpers can be used to carry signals between GPIO pins.

For this test half of the pins are configured as input pins and the other half as output pins, in such a fashion that a jumper can connect each output to an input. Then the output pins are driven High individually, and the input pins are being read through a console running on the EMP, to confirm that the signals are received correctly and mapped to the correct registers. From the test it was concluded that the read functionality works correctly. An example of this loopback test can be seen in Figure 4.6, where headers are used to connect pins as

$$\text{pin } 0 \Rightarrow \text{pin } 1, \qquad \text{pin } 10 \Rightarrow \text{pin } 11.$$

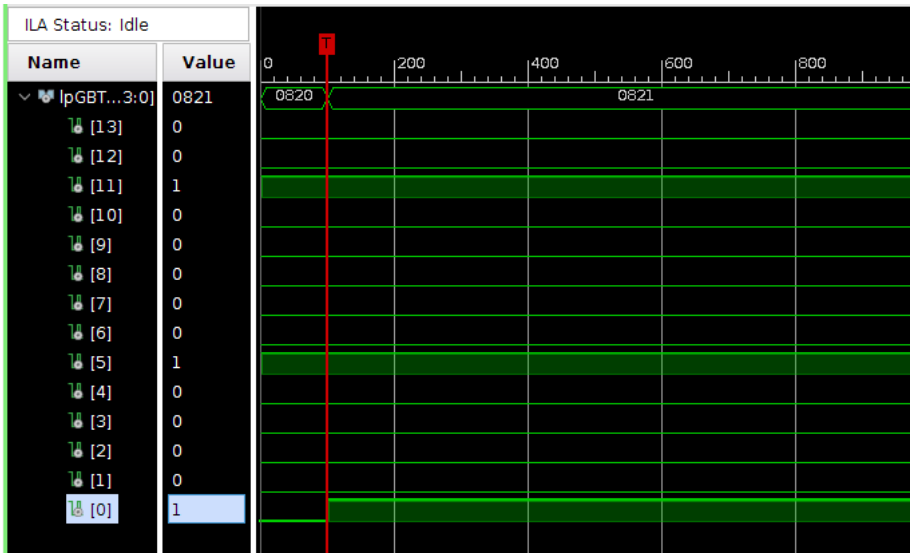

Figure 4.5: Screenshot of the output test in the Vivado ILA.

```
Drive pin high or low? Syntax h1, l5, etc.
h0
2024-05-10 14:48:48.325936 [Gpio.cpp:240, INF] Setting the voltage level of the pin 0 to HIGH
2024-05-10 14:48:48.326810 [Gpio.cpp:115, DBG] Read output pin voltage level register content: 0b00000000
2024-05-10 14:48:48.327348 [Gpio.cpp:129, DBG] Write output pin voltage level register content: 0b00000001
2024-05-10 14:48:48.337628 [Gpio.cpp:280, INF] Reading the value of pin 0
2024-05-10 14:48:48.338326 [Gpio.cpp:100, DBG] Read input pin voltage level register content: 0b00000011
2024-05-10 14:48:48.338510 [Gpio.cpp:280, INF] Reading the value of pin 8
2024-05-10 14:48:48.339205 [Gpio.cpp:100, DBG] Read input pin voltage level register content: 0b11000000
```

```
h10
2024-05-10 14:54:17.099044 [Gpio.cpp:240, INF] Setting the voltage level of the pin 10 to HIGH
2024-05-10 14:54:17.099935 [Gpio.cpp:115, DBG] Read output pin voltage level register content: 0b00000000
2024-05-10 14:54:17.100473 [Gpio.cpp:129, DBG] Write output pin voltage level register content: 0b00000100
2024-05-10 14:54:17.110723 [Gpio.cpp:280, INF] Reading the value of pin 0
2024-05-10 14:54:17.111461 [Gpio.cpp:100, DBG] Read input pin voltage level register content: 0b00000011
2024-05-10 14:54:17.111648 [Gpio.cpp:280, INF] Reading the value of pin 8
2024-05-10 14:54:17.112343 [Gpio.cpp:100, DBG] Read input pin voltage level register content: 0b11001100
```

Figure 4.6: Example of loopback test.

In the upper screenshot of the Figure 4.6, it can be seen that pin 0 is first driven High by setting the first bit of the "output pin voltage register" to 1. Then the values of all GPIO pins are checked by reading a pin from each register in the Port, in this case pins 0 and 8, and it is seen that pin 1 has also become High. The same thing is done in the second picture, but for pins 10 and 11. Note that pins 14 and 15 are always High, as they are being used internally on the EMCI to control the FEASTMPs, as was mentioned in Section 2.1.

Pull-up Resistor Test

To test the pull-up resistor, the external pins of the splitting board were used again, this time to be probed with a multimeter. This method was chosen instead of Vivado probing, because the lines on the splitting board, that connects to the lpGbt GPIO pins, features pull-down resistors. These somewhat counteracts the lpGbts pull-up resistors, resulting in an on-line voltage level smaller than the 1.2V required for a signal to be considered High. However, the voltage level could be measured with the multimeter and was large enough to be distinguishable from electrostatic noise.

When measuring the pins with the multimeter, a pin with an activated pull up resistor had a voltage of about 0.272V, in contrast to a floating pin which fluctuated close to 0 with a maximum observed deviation of +0.002V. The voltmeter was used to measure the effect of the pull-up resistor

on all pins, and it was observed to function correctly. The usage of the multimeter can be seen in Figure 4.7, where the readout of 0.272V is shown.

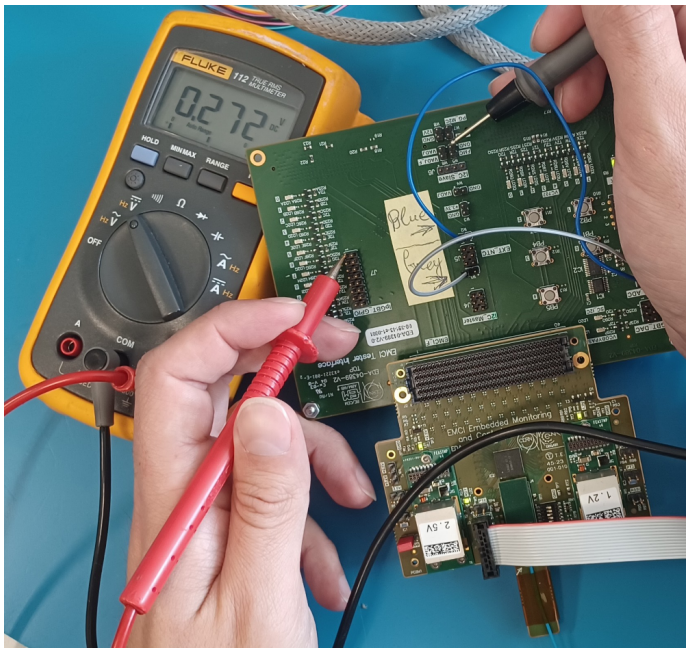

Figure 4.7: Using a multimeter to measure a pin with an active pull-up resistor.

Drive Strength Test

To test the drive strength of a pin, an oscilloscope was used. This tool is able to measure electrical signals and sample them of a rate of 40G samples/second, which corresponds to one sample each 250ps. For this test the trigger condition on the oscilloscope was set to be a rising edge above 360mV, and the persistence setting, which allows for plotting several signals on top of each other, was used.
The test was done by connecting the oscilloscope to a pin and then driving this pin High with both normal and increased driving strength enabled. It was found that the driving strength functionality worked correctly for all pins. An example of the output from the oscilloscope can be seen in Figure 4.8. It can be seen that when using High driving strength the signal reaches the 1.2V target faster, concretely around 2.5ns faster, than with normal driving strength. The signal with High driving strength also overshoots the target voltage by around 500mV before dropping back down towards the 1.2V target.

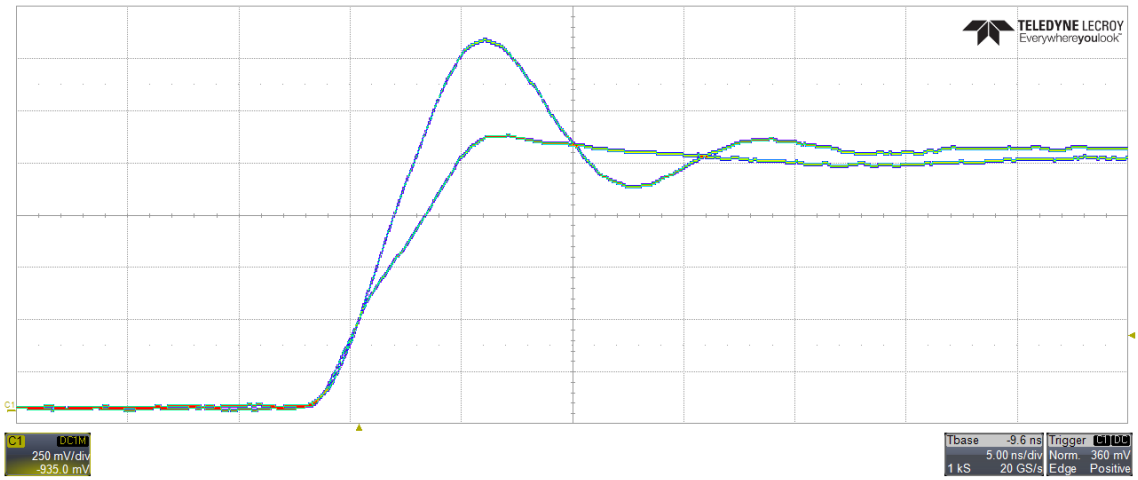

Figure 4.8: Signals going HIGH with both Low and High driving strength, recorded by an oscilloscope.

5 | Inter-integrated Circuit

Inter-integrated Circuit (IIC or I^2C) is a communication protocol, developed by NXP (previously Philips) Semiconductors, that is used in many different electronics today, due to its flexibility and simplicity. An I^2C bus features a single half-duplex communication line that connects several devices. Any device that follows the protocol can then 'clipped' onto the I^2C line, allowing it to immediately communicate with other devices on the bus. This flexibility makes designing, modifying and upgrading systems much easier. Furthermore, the standardized protocol alleviates the designer from having to craft a custom communication protocol each time a board is made, and the manufacturing of the PCB also becomes cheaper as the I^2C bus, which consists of only 2 lines, minimizes the amount of interconnections on the PCBs [16, p. 3-5].

The LpGbt features an I^2C-bus following the NXP standard. Thus this section will primarily be based on the NXP I^2C-bus specification manual [16].

The I^2C-bus features two bi-directional lines, one for communication and the other for a synchronised clock signal. These lines are called the serial data (SDA) and serial clock (SCL) lines, respectively. The protocol requires all devices connected to the bus to act both as both senders and receivers, but categorizes them into controllers and targets, historically called masters and slaves. The controller devices initiates all data transfers and control the clock line. The target devices are only allowed to answer when addressed. This addressing method is core to the I^2C protocol, as it is exactly what allows so many devices to be connected together. However, to understand the addressing, and any data transfer on the bus, it first needs to be explained how the binary states are created in the circuit.

5.1 Physical Layer

The lines on the I^2C bus are serial and the SDA line is half-duplex, meaning that all messages has to be split into bits that are sent on each clock cycle, and that only one message can be transmitted at any time. Both lines are connected to a positive voltage source V_{DD} through a pull-up resistor, making the passive state of the lines High. The connection scheme for the devices on the bus depends on the type of transistor used, so we will only discuss the scheme for NMOS transistors here. In this scheme each device connects the lines to a GND through an NMOS transistor acting as a switch. The switching is controlled inside the devices, allowing each device control to the lines by pulling them low and releasing them high, independently. This method of creating outputs is called open-drain [20, p. 5].
An illustration of the open drain connection can be seen in Figure 5.1. In the figure, some I^2C compatible device is shown in the inner square box. The SCL and SDA lines are connected to this device through the simple circuit in the dashed square, consisting of an amplifier and a transistor switch. Both lines are connected to such a circuit, respectively, which allows the device to pull either line low by closing the switch, in turn creating a route from the the line to ground.

The main advantage of open-drain is that the line follows AND logic, since the line will always be Low when any device pulls it Low. The I^2C protocol utilizes this, by dictating that each controller must release the SDA line if the state of the SDA does not match what the controller is currently sending. The combination of this arbitration protocol, the wired AND connection and the soon to be discussed Start/Stop conditions, allows for non-destructive bus interference. Thus the line is able to run at full informational capacity when two devices tries to transmit simultaneously, as one message is guaranteed to be sent unobstructed.
The wired AND connection is also used for the SCL line, but here the I^2C protocol specifies that the controllers must try to synchronize with the clock cycles on the SCL. This is done by having the controllers that releases the clock first enter a wait state until the SCL goes High, at which point all controllers in contention have synchronized their High clock period timers. The controller with the shortest High period then determines when the SCL goes Low again. Thus when controllers

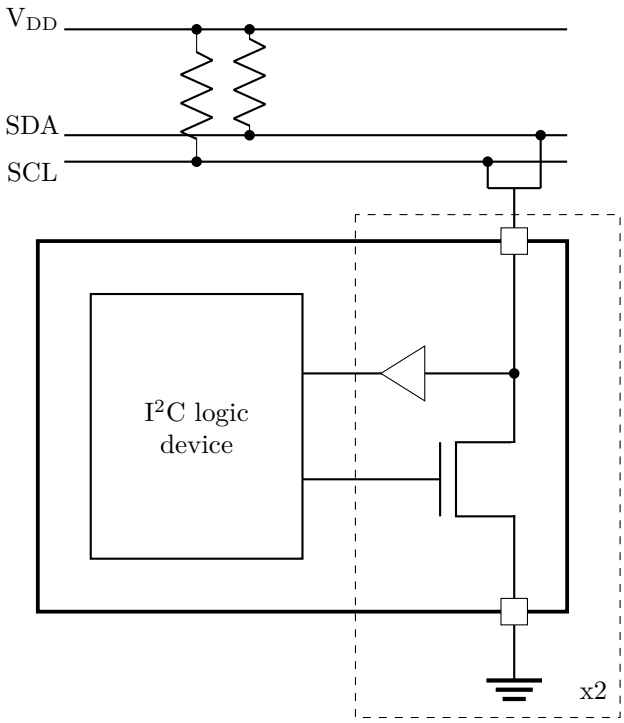

Figure 5.1: Illustration of a device with an open-drain output connected to the I^2C bus. Inspired by [20, p. 5].

with different clock periods are in contention, the SCLs Low period is determined by the controller with the longest clock low period and the SCLs High period by the controller with the shortest High period. The SCL contention ends when a controller loses arbitration on the SDA, after which it must release the SCL at the end of the current frame.

Another advantage of an open-drain connection is that it makes the connection from the line to a device simple, which means that the load on the electronics is known and stable. [20, p 6] gives an example of an undriven line with two devices that can both push and pull, and mentions that a contention could potentially damage the devices. Such a push-pull output is sometimes used by a serial peripheral interface (SPI), where an additional non-data line has to be dedicated to arbitration.

With some understanding of the how the I^2C bus works electrically, we can move onto the communication protocol.

5.2 The I^2C Protocol

The I^2C specifies a simple protocol for communication on the bus, which is comprised of Start/Stop conditions, device addresses, data frames and acknowledgements. The standard also suggests more functionality, which individual implementations can decide to implement in addition to the base functionality.

5.2.1 Start and Stop Conditions

Whenever a controller wants to communicate on the I^2C bus, it has to take ownership of the bus for a period of time. To do this the controller has to send a Start condition while the lines are High, by pulling the SDA line Low and then subsequently the SCL line. When both lines are Low the controller has ownership of the bus and can start transmitting the clock signal and the data bits. After the controller has finished its communication, it sends a Stop condition by releasing the SCL line High followed by the SDA line, i.e., in the opposite order of the Start condition. Both

conditions are illustrated in Figure 5.2.

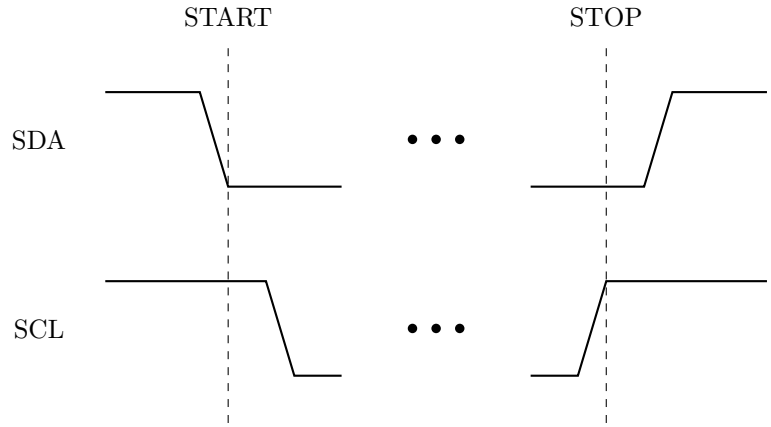

Figure 5.2: Illustration of the Start and Stop conditions on the I^2C lines.

In the special case when a controller needs to communicate with multiple targets, it has to do so sequentially. To not lose ownership of the bus the controller can send another start condition, after finishing its initial communication, by pulling the SDA low during a clock cycle instead of sending a Stop condition. This operation is called a Repeated Start and allows the controller to immediately communicate with another target. Otherwise all communication happens between the Start and Stop conditions.

5.2.2 Address Frames

After a controller has ownership of the bus it can starts its communication. All communication on the bus happens in 9-bit frames, which consists of 1 byte of communication data and 1 acknowledgement (ACK) bit. The communication data is sent by the transmitter and the ACK is sent by the receiver. Each bit of a frame has to be sent while the clock on the SCL is High, and the SDA must be stable whenever the SCL is High. This is required as changes on the SDA while the SCL is High are interpreted as Start or Stop conditions.

On the first clock cycle after a Start condition, the controller has to start transmissions of an Address frame. Every device on the bus is associated with an address, which is usually set internally in the registers of the device, that the device responds to. The addresses are 7 bits long, and the address frame appends an 8th R/W bit which tells the target whether it should transmit or receive data. It should be noted that the R/W bit vocabulary is designed w.r.t. the controller, meaning that the target should receive data when receiving a write bit and vice versa for a read bit. This discrepancy is illustrated in Table 5.1.

Frame vocabulary	Target meaning	bit
write (w)	Receive	0
read (r)	Transmit	1

Table 5.1: Meaning of read/write frame bits for the target.

The 9th bit is the acknowledgement. When a device registers it's address being sent on the SDA after a Start condition, it transmits an ACK. This communication happens in the opposite direction of the addressing, thus the controller has to release the SDA line for 1 clock cycle, in which the target can pull it Low. If the target fails to pull the SDA Low, it is seen as an not acknowledged (NACK) bit and the controller concludes the communication with a Stop or Repeated Start condition.

All extra functionality that requires the I^2C bus is also communicated using the address frame, by reserving certain addresses to indicate that the functionality is to be used. The I^2C standard specifies a total of 16 addresses that can be reserved for additional functionality, all of which can

be be found in [16, p. 16]. However, this functionality only works if the target devices are designed with it in mind, allowing custom implementations to use the reserved addresses for other purposes.

One of the extra functionalities related to addressing is the ability to use extended addresses for devices. This functionality works by reserving the address string (11110XX), as a bus code that tells all devices that 10-bit addressing is being used. The full format of the 10-bit addressing mode can be seen in Table 5.2.

Frame 1				Frame 2	
Ext. Addr. call	2 MSB	Write bit	ACK	8 LSB	ACK
11110	XX	0	0	XXXXXXXX	0

Table 5.2: Frame format for 10-bit addressing on the I^2C bus.

From the table it can be seen that the 10-bit address of the target is split over 2 bytes, with most of the first byte being used for protocol data. Note that the 8th bit is always a write bit (0). This is because the R/W bit does not hold any information when using 10-bit addressing, but is instead used as a redundant part of the frame. If the controller wants to write, it can start the transmission of data right after the 2nd address frame. If it wants to read, it has to perform a Repeated Start followed by a third frame, that is identical to Frame 1 in Table 5.2 with the exception that the R/W bit is now a read bit (1). After this 3rd frame the targets starts its transmission.

5.2.3 Data Frames

After a controller has addressed a target, the transmitter can start transmission of data frames. Each frame contains a byte of data and an ACK from the receiver. The I^2C standard defines no limit on the amount of frames transmitted, thus this should instead be determined by devices participating in the communication. From the controllers side any communication is concluded by a Stop or Repeated Start condition. However, in the case of the read operation, a NACK is first transmitted to immidiately stop the target from continuing its transmission. From the targets side communication can only be concluded by transmitting a NACK.

The conditions for concluding communication are important, because most I^2C devices will try to transmit all their available data bytes when receiving a read bit. In cases when the controller does not need all the bytes, it can use the NACK after the last needed bit to stop the communication.

The I^2C standard does not specify a way to R/W to a single register on a target device, but some implementations do include this functionality, [see examples in 20, p. 11-13]. This is usually done by letting the first data frame be a register pointer, and then continue with the standard writing operation. In case the controller wants to read the register, the register pointer frame can be followed by a Stop+Start or Repeated Start condition, after which the target will remember the set register and transmits the registers data upon receiving another address frame with a read bit. Functionality for using register pointers is also inlucded in the LpGbt [6, p. 109].

5.2.4 Bus Speed

The I^2C-bus was initially limited to a rate of 100 kb/s, but this limit on rate has since been increased. The new speeds has been quantized into the different modes shown in Table 5.3, which are specified by the manual [16].

Mode	Standard	Fast	Fast Plus	High-speed	Ultra Fast
Speed [kb/s]	100	400	1000	3400	5000

Table 5.3: Different speed modes of the I^2C-bus.

All the speed modes are backwards compatible except the Ultra Fast mode, which is a unidirectional write-only mode and hence does not use ACKs. The bus speed is usually set in the controllers internal registers, which allows for a controller to communicate at different speeds with different devices. It

should, however, be noted that the modes are not forward compatible, thus communicating above the minimum speed limit of devices on the bus could result in undefined behavior. The issue of some target devices not supporting the higher speed modes, can be solved by using multiple busses or adding a bridge section to the bus, that can disconnect the slow devices when a high bus speed is needed [16, p. 39].

The higher bus speed modes impose greater requirements on the bus lines and the controllers electronics, due to the requirement of shorter rise time. They may also change the communication protocol, as happens in High-speed mode which changes the bus Low state voltage to half the High state voltage, as opposed to GND, and also removes arbitration and clock synchronization when running at 3.4 Mb/s. Instead any High-speed transaction must be initiated by a the frame shown in Table 5.4, which is sent in either Normal or Fast mode.

High-speed startup frame			
Start condition	Controller call	Controller ID	NACK
S	00001	XXX	1

Table 5.4: Frame that indicates the initialization of High-speed mode.

The controller code, consisting of the controller call and ID, uniquely determines which controller wins an arbitration in case of bus contention, after which High-speed mode is enabled until the controller transmits a Stop condition. Ultra fast mode also changes the protocol due to the unidirectionality previously mentioned.

With the core concepts of the I^2C bus explained, the software interface for the I^2C circuit on LpGbt can be presented.

5.3 LpGbt I^2C Interface

The lpGbt features 3 I^2C controllers, M0, M1 and M2, of which M2 is reserved for controlling the VTRx+ connection on the EMCI. These controllers are related to a set of registers, some of which are available to the lpGbt, and other which are internal to each controller. The I^2C registers available to the lpGbt are thus called external to distinguish them from the registers internal to each controller.

The controllers can be controlled through their respective external command registers, in which 4 bits are used to give the controller one of 16 possible commands. The commands can tell the controller to read or write to a target, or to read/write to its own internal registers. All writing operations depends on the controllers external data registers, of which each controller has four called Data0 - Data3.

When doing a single write, the controller will take the pull the data from the external Data0 register. When doing a multibyte write, however, the controller will instead use the data in its internal Databyte registers for transmission. This is done because each controller is capable of either reading or writing up to 16 bytes before it has to drop control of the I^2C-bus, which is more bytes than there are external data registers. Hence data must be loaded sequentially into the the data registers, and the controller told to pull this data into its internal Databyte registers inbetween each loading iteration. When all the data has been pulled into the controllers internal registers, a multibyte write command can be issued to make the controller begin the transmission.

For the read operation there are 17 external registers which the controller will store the data in. Thus both the single- and multibyte operations only require a single command to the controller, after which the data can be read from external registers.

Each controller also features an external address register, in which the target for an upcoming I^2C transaction is set. Hence the address of the desired target device must be loaded into this register, for the master to correctly transmit data to the target.

C++ code has been written to handle most of these read/write operations. However, at the time of writing it has not been possible to find or create a target device, that follows the I^2C protocol.

Hence it has not yet been possible to test and debug the code, so the implementation of the code will not be discussed.

6 | Conclusion

In this project CERN, its Atlas detector system and the idea of electronic interfaces for data communication was discussed, as well as the EMP-EMCI system, an interface currently in development at CERN. The GPIO and I^2C systems on the lpGbt of the EMCI were explained, and software was developed to configure these systems through the EMP. All functionality of the lpGbt GPIO was implemented and tested. The core communication functionality of the I^2C was implemented but remains to be tested. In addition, some non-communication functionality of the I^2C, such as setting the rate of a controllers clock signal, also remain to be implemented.

For future work the full functionality of the I^2C system should be implemented. Furthermore, the lpGbt features settings related to systems smaller than GPIO and I^2C which were not discussed in this report. These settings should still be configurable by software on the EMP. As the internship at CERN does not conclude with the turn-in of this report, most of this future work is expected to be completed.

Bibliography

[1] Atlas Collaboration at CERN. The atlas experiment at the cern large hadron collider. Journal of Instrumentation, 3(08):S08003, August 2008. Find at https://dx.doi.org/10.1088/1748-0221/3/08/S08003.

[2] D. Blasco. Electronics specification component or facility: Emci, version 1.2. https://edms.cern.ch/document/2332346/2, April 2021. Document in work, publicly accessible.

[3] D. Blasco, P. Moschovakos, P. P. Nikiel, V. Ryjov, and S. Schlenker. Electronics specifications component or facility: Emp, version 1.3. https://edms.cern.ch/document/2631648/1, November 2023. Document in work, currently only internally accessible.

[4] Luis Roberto Flores Castillo. The Search and Discovery of the Higgs Boson. 2053-2571. Morgan & Claypool Publishers, 2015.

[5] CERN. The atlas detector control system. In Proceedings of ICALEPCS2011, pages 5–8, Grenoble, France, October 2011. Proceedings hosted by European Synchrotron Radiation Facility (ESRF).

[6] Manual for lpGbt V1. CERN, April 09 2024. V1 = https://lpgbt.web.cern.ch/lpgbt/v1/lpGBT.pdf is currently only accessible with CERN account, V0 is publicly available on https://cds.cern.ch/record/2809058/files/lpGBT_manual.pdf.

[7] CERN. Cern discovers higgs boson. https://home.cern/science/physics/higgs-boson/how, Accessed 01/05, 2024.

[8] CERN. Cern invents the world wide web. https://home.cern/science/computing/birth-web/short-history-web, Accessed 01/05, 2024.

[9] CERN. Cern creates antihydrogen. https://timeline.web.cern.ch/first-antiatoms-produced-antihydrogen-cern, Accessed 01/05, 2024.

[10] CERN. Cern mission statement. https://home.cern/about/who-we-are/our-mission, Accessed 01/05, 2024.

[11] CERN. High-luminosity lhc. https://home.cern/resources/faqs/high-luminosity-lhc, Accessed 06/06, 2024.

[12] CERN. Trigger and data acquisition (of atlas). https://atlas.cern/Discover/Detector/Trigger-DAQ, Accessed 06/06, 2024.

[13] CERN. The atlas collaboration. https://atlas.cern/Discover/Collaboration, Accessed 20/05, 2024.

[14] CERN. The atlas detector. https://atlas.cern/Discover/Detector, Accessed 20/05, 2024.

[15] P. Moschovakos, V. Ryjov, S. Schlenker, D. Ecker, and J. B. Olesen. The embedded monitoring processor for high luminosity lhc. Journals of Accelerator Conferences Website, ICALEPCS2023:1470–1474, December 2023.

[16] I^2-bus specification and user manual. NXP Semiconductors, 46, HTC 46, 5656 AE Eindhoven, Netherlands, 7 edition, October 1 2021. Can be found at https://www.nxp.com/docs/en/user-guide/UM10204.pdf.

[17] Samtec. Samtec firefly uec5 product page. https://www.samtec.com/products/uec5-1, Accessed 05/06, 2024.

[18] Paul Scherz and Dr. Simon Monk. Practical Electronics for Inventors. McGraw-Hill Education, 4th edition, March 24 2016.

[19] D. Blasco Serrano, M. Haahr Kristensen, A. Bay Madsen, P. Moschovakos, P.P. Nikiel, V. Ryjov, and S. Schlenker. Description and status of the emci-emp interface. Journal of Instrumentation, 17(06):C06012, June 2022.

[20] Joseph Wu. A Basic Guide to I^2C. Texas Instruments, November 2022. `https://www.ti.com/lit/pdf/sbaa565`.

Image references

[21] CERN. Cern's accelerator complex. https://cds.cern.ch/record/2800984, Accessed 20/05, 2024.

[22] CERN. Computer generated image of the whole atlas detector. https://cds.cern.ch/record/1095924, Accessed 21/05, 2024.

[23] CERN. Computer generated image of the atlas inner detector. https://cds.cern.ch/record/1095926, Accessed 22/05, 2024.

[24] CERN. How atlas detects particles: diagram of particle paths in the detector. https://cds.cern.ch/record/1505342, Accessed 22/05, 2024.

Appendices

A | Copyright of CERB Document Server

All images from CERN used in this report, are used in an educational context and are provided "as-is". Furthermore, CERN is credited as the source and a link is provided to a location where the image can be found in the "Image references" bibliography. This use is in accordance with CERNs terms, as stated on `https://copyright.web.cern.ch/`.