

UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE FÍSICA



Development of the ATLAS-High Granularity Timing Detector Interlock Monitoring System

Alexandre Domingos Parreira

Mestrado em Engenharia Física

Dissertação orientada por:
Doutora Helena de Fátima Nunes Casimiro dos Santos
Professor Doutor José António Soares Augusto

Acknowledgments

Esta dissertação demonstra vários meses de esforço e dedicação da minha parte, mas certamente não seria possível tê-la terminado sem a ajuda de um grande grupo de pessoas à qual devo os meus profundos agradecimentos. Havendo dois públicos alvos que pretendo atingir, gostaria que os respectivos agradecimentos não fossem perdidos na barreira do idioma, usando o idioma mais apropriado para cada um.

Primeiro que tudo, aos meus pais, Angelina Domingos e António Parreira por todo o apoio que sempre e incondicionalmente me deram e que só assim me foi possível chegar onde estou e ser o que sou.

Aos meus orientadores, gostava de agradecer à Doutora Helena Santos por me introduzir à extensa rede do CERN sempre com interesse e admiração e ao Professor José Augusto pela disponibilidade e todas as tardes que passou a tentar resolver os problemas comigo e pela partilha do seu extenso conhecimento e mestria em todo o tipo de assuntos que nem sabia que era possível atingir. Ao Rui Vieira pelo companheirismo durante esta jornada e ao Professor Alexandre Cabral por me auxiliar na análise de resultados e pelo seu valioso *feedback*.

Também gostava de agradecer ao grupo do LIP-HGTD do qual fui envolvido, por toda a ajuda e apoio dado. Mas gostava de agradecer especialmente ao Filipe Martins por toda a ajuda e tempo que me deu no que toca ao WinCC OA em tão em cima da hora.

Um agradecimento também à minha amiga Leonor Mendes, que me acalmou ao dizer que o avião de ida para o CERN não ia cair e por todos os desabafos e vivências.

Second, this work would certainly not be possible without the help of the ITk Interlock team. From the Mon-FPGA and PYNQ board, I am sincerely grateful to Chen Wen Chao and to Jean-Pierre Martin for the continuous help throughout this project; from the monitoring interface, I would like to thank to Simon Koch and to Hans Joos for introducing me and helping me navigate in such complex system; and I would like to express my thanks to Peter Kind and to Marius Wensing for helping me in understand the electronics of the LISSY cards.

Resumo

O Grande Colisor de Hadrões ([LHC](#)) irá estar sujeito a uma série de melhoramentos divididos em duas fases. Estes melhoramentos irão permitir injetar um feixe de prótons com uma luminosidade superior à atual. A maior luminosidade implica um aumento no número de colisões de partículas simultâneas, o que dificulta a seleção e a distinção de colisões com interesse para a Física. A experiência [ATLAS](#) (Um Instrumento Toroidal do LHC) é uma das quatro experiências acopladas ao [LHC](#) e, assim como as restantes, terá que lidar com aquele aumento de colisões simultâneas, através do melhoramento generalizado dos seus instrumentos e sistemas de monitorização e controlo. Atualmente, o sistema de seleção de colisões com interesse do [ATLAS](#) não consegue lidar com o aumento previsto de luminosidade. Face a este desafio, foi proposto o desenvolvimento do Detetor de Alta Granularidade Temporal ([HGTD](#)), um detetor adicional que pretende auxiliar no processo de seleção ao fornecer informação de alta precisão sobre o instante das colisões que, juntamente com as três coordenadas espaciais em que ocorrem, permitirá distinguir com maior clareza as colisões de interesse.

Dada a elevada importância do [HGTD](#) para o melhoramento do [ATLAS](#), é essencial garantir a correta operação deste detetor com a criação de diversas camadas de segurança. O Sistema de Interlock (Sistema de Bloqueio de Segurança) é uma delas. Também designado por última linha de defesa, irá lidar com as temperaturas dos discos do detetor e com sinais genéricos de perigo, tais como inundações ou problemas no sistema de refrigeração. Na ocorrência de sinais de perigo, o sistema desliga as Fontes de Alimentação ([PS](#)) que alimentam os diferentes setores dos discos do detetor. O corte da alimentação é realizado de acordo com os sinais de perigo recebidos, podendo ser apenas de um setor, de vários setores ou mesmo geral para todos os discos, havendo quatro discos no [HGTD](#).

O Sistema de Interlock é um sistema eletrónico baseado numa arquitetura modular e contém várias placas, cada uma delas dedicada a uma certa função. A colheita de dados do sistema local é feita pelo módulo da FPGA de Monitorização ([Mon-FPGA](#)). A decisão de desligar as fontes de alimentação dos diferentes setores dos discos do detetor é realizado pelo módulo da FPGA de Interlock ([ILK-FPGA](#)). A ligação entre os dados de temperatura dos sensores de temperatura instalados nos discos do detetor e o Sistema de Interlock é feita através da placa de Temperatura para Interlock ([T2I](#)). Os sinais genéricos externos de outros sistemas de segurança do detetor são enviados para a placa de Sistema de Segurança Global ([GSS](#)). A ligação entre as Fontes de Alimentação dos discos do detetor e o Sistema de Interlock é feita através das placas de Saida (OUT), que desligam as fontes de alimentação quando a [ILK-FPGA](#) assim comandar.

O Sistema de Controlo de Decisão ([DCS](#)) do [ATLAS](#) monitoriza não só todos os sistemas do [HGTD](#), como também os de outros detetores da experiência [ATLAS](#). O Sistema de Monitorização do Sistema de Interlock é a ligação entre o Interlock e o [DCS](#) e permite que um utilizador do [DCS](#) aceda, visualize e controle o Sistema de Interlock. No protótipo do sistema no decorrer deste último ano, o Sistema de Monitorização inclui três partes: (i) a [Mon-FPGA](#); (ii) uma placa PYNQ-Z2 que aloja um servidor, onde são carregados os dados reunidos pela [Mon-FPGA](#); (iii) e finalmente a interface gráfica que exhibe os dados do sistema para o utilizador do [DCS](#).

A implementação de um protótipo do Sistema de Monitorização é o foco desta dissertação e as atividades desenvolvidas podem ser divididas em três partes: instalação da cadeia do Sistema de Monitorização; implementação da interface de monitorização; teste do Sistema de Interlock e do seu Sistema de Monitorização.

Como já referido, uma das partes corresponde à instalação e configuração de ambas as placas do Sistema de Monitorização: a [Mon-FPGA](#) e a placa que aloja o servidor. A ligação entre estas duas placas é designada por cadeia do Sistema de Monitorização. A [Mon-FPGA](#) é essencialmente constituída por uma [FPGA](#), que é um circuito integrado digital reconfigurável de uso genérico que permite realizar tarefas e implementar funcionalidades no domínio da eletrónica digital. Neste caso, a sua função principal é reunir e transmitir os dados das placas anteriormente mencionadas. A placa que aloja o servidor é uma placa de desenvolvimento PYNQ-Z2, que possui um circuito integrado versátil que agrega uma [FPGA](#) a um microprocessador. Esta placa permite, no Sistema de Monitorização, não só executar o servidor, como também reconfigurar a [FPGA](#) da [Mon-FPGA](#). A reconfiguração da [Mon-FPGA](#) é feita através do carregamento de um ficheiro criado a partir de uma linguagem de descrição de sistemas digitais eletrónicos. Este carregamento é, portanto, realizado pela placa PYNQ-Z2. Após reconfigurar a [Mon-FPGA](#), a placa de desenvolvimento encarrega-se de executar o servidor, que recolhe os dados periodicamente colhidos pela [Mon-FPGA](#).

Outra tarefa consistiu na implementação da interface de monitorização que exhibe os dados do Sistema de Interlock a um utilizador. A interface gráfica foi realizada em WinCC Arquitetura Aberta ([WinCC OA](#)), um sistema ou linguagem de programação própria para o desenvolvimento e criação de interfaces gráficas com o intuito de visualização e controlo de sistemas remotos, usando, neste caso, os dados provenientes do Sistema de Interlock e armazenados no servidor, transferidos através de processos de “pedido e disponibilização”.

Tanto o Sistema de Interlock, como a interface gráfica e a cadeia do Sistema de Monitorização incluem muitos elementos migrados de um outro detetor da experiência [ATLAS](#) já existente, o detetor de Trajetórias Interno ([ITk](#)). As equipas envolvidas no desenvolvimento deste detetor produziram as placas do Sistema de Interlock e todo o seu sistema eletrónico. Dadas as semelhanças entre os Sistemas de Interlock de ambos os detetores e a flexibilidade inerente ao projeto do Sistema de Interlock, o Interlock do [ITk](#) será adaptado ao [HGTD](#) de acordo com os requisitos de projeto e a funcionalidade próprios deste detetor.

A outra tarefa inserida no presente trabalho é o teste do Sistema de Interlock e, ao mesmo tempo, do Sistema de Monitorização, através da simulação de situações de temperatura fora do limite normal de operação, e do teste com sensores de temperatura numa câmara climática. Ao provocar situações de risco de temperatura no sistema protótipo, foi possível verificar a sua reação com o auxílio de alguns circuitos eletrónicos e através da interface de monitorização construída. Para se poder projetar os procedimentos de teste foi necessário estudar a eletrónica das placas. Esta dissertação focou-se no teste da placa [T2I](#), que possui, em cada canal de transmissão de sinal dos sensores de temperatura, Amplificadores Operacionais ([OP-AMPs](#)), que comparam a temperatura dos sensores com temperaturas de referência correspondentes aos limites do intervalo de operação normal. Sinais fora deste intervalo, representam temperaturas que podem colocar em risco o funcionamento do detetor. Neste caso, os sinais são transmitidos à [ILK-FPGA](#) para que ela ordene o corte de alimentação das Fontes de Alimentação do setor dos discos em risco. Paralelamente, a temperatura dos sensores que está associada a uma tensão elétrica analógica, é lida por Conversores Analógico-Digital ([ADCs](#)), que a convertem numa palavra digital, sendo depois transmitida à [Mon-FPGA](#) e posteriormente carregada no servidor.

O teste de ambos os sistemas precede a sua futura implementação no [CERN](#) (Conselho Europeu para a Pesquisa Nuclear) como sistema de proteção de uma Placa Eletrónica Periférica ([PEB](#)). Estas [PEBs](#) destinam-se a serem instaladas na periferia da circunferência dos discos do [HGTD](#). A placa [PEB](#) usada nesta implementação preliminar é designada por "Demonstrador". Este tipo de placas é responsável pela transmissão de sinais entre os discos e os sistemas externos, tais como as temperaturas lidas pelos sensores transmitidas para o Sistema de Interlock. Este sistema, com o auxílio do Sistema de Monitorização, permitirá usar em segurança o "Demonstrador" em testes e simulações de partes do detetor ainda em desenvolvimento.

O trabalho apresentado nesta dissertação insere-se na colaboração portuguesa do Laboratório de Instrumentação e Física Experimental de Partículas (LIP) no projeto [ATLAS/CERN](#).

Palavras chave: CERN, Experiência ATLAS, HGTD, Sistema de Interlock, WinCC OA.

Abstract

The [LHC](#) is under modifications that will allow the injection of a beam with a larger luminosity. However, this advancement increases the number of simultaneous particle collisions. The selection of the collisions of importance is a major concern for the experimental studies. The [ATLAS](#) experiment is one of the [LHC](#) four experiments, and its current collision selection system is not enough to handle that increase of luminosity. Hence the need to develop the [HGTD](#). This new detector will provide an additional high-precision timing information ensuring an improved selection of the interesting events.

The Interlock System is one of the several safety layers ensuring the [HGTD](#)'s proper operation. It processes temperature data from the detector's disks and generic danger signals sent by other safety systems. Within this hardware system, there is the [Mon-FPGA](#) that monitors every signal received from or transmitted to the Interlock System. The [Mon-FPGA](#) is a part of the Monitoring System that deals with the monitoring and access of the hardware system's data. The Monitoring System is the subject of this thesis.

The work presented in this thesis was developed in three parts. One includes the configuration of the monitoring hardware and the server hardware that allows to load the data collected by the former.

In addition to this hardware-based system, there is the development of an user interface which accesses the server and displays the data loaded in it. These two parts together form the Monitoring System as a whole.

The third part refers to the Monitoring System's testing by simulating temperature tests at FCUL, Lisbon, and real-time temperature tests at [CERN](#). These tests were conducted in a way that the entire Interlock System would be tested, by verifying its reaction to out-of-limit temperatures. If implemented correctly, the Monitoring System will allow to supervision the system's reaction.

Keywords: CERN, ATLAS experiment (acronym of A Toroidal LHC ApparatuS), HGTD (acronym of High Granularity Timing Detector), Interlock System, WinCC OA.

Contents

Acknowledgments	iii
Resumo	v
Abstract	ix
List of Figures	xv
List of Tables	xvii
List of Acronyms	xxiii
List of Symbols	xxv
1 Introduction	1
1.1 Motivation and Objectives	1
1.2 Methodology and Resources	1
1.3 Outline	2
2 The ATLAS Experiment at CERN	5
2.1 The Large Hadron Collider (LHC)	5
2.2 The ATLAS Experiment	6
2.3 The High-Luminosity Large Hadron Collider (HL-LHC)	8
2.3.1 Luminosity and the "pile-up problem"	8
2.4 The High Granularity Timing Detector (HGTD)	9
3 The HGTD Interlock System	11
3.1 The Interlock System	11
3.1.1 The Main Interlock Crate (MIC)	13
3.1.2 The Local Interlock & Safety SYstem (LISSY)	14
3.1.2.1 The ILK-FPGA	14
3.1.2.2 The Mon-FPGA	15
3.1.2.3 The T2I module	16
3.1.2.4 The OUT module	18
3.1.2.5 The GSS module	18
3.1.3 The Detector Control System (DCS)	19
3.2 The ITk Interlock System	20
4 The hardware of the Monitoring System	21
4.1 The FPGA	21
4.2 The Mon-FPGA version's history	21
4.3 The VHDL language	22

CONTENTS

4.4	Firmware Requirements	24
4.5	Firmware Implementation	24
4.5.1	The "FromLink" module	25
4.5.2	The "Main_CTRL" module	26
4.5.3	The "I2Cslots" and "ReadDataBlocks" modules	27
4.5.4	The "ToLink" module	27
4.6	The PYNQ board	27
4.6.1	The implemented setup	28
4.6.2	On-System Configuration and Application	28
4.6.2.1	The server configuration file	32
4.6.2.2	System integration	34
5	The interface of the Monitoring System	37
5.1	WinCC OA architecture	37
5.2	The monitoring project	38
5.2.1	ITk project migration	38
5.2.2	The JCOP framework	40
5.2.3	The Quasar's Cacophony tool	41
5.2.4	The LISSY panels	46
5.2.4.1	The Mon-FPGA "alive" alarm	50
5.2.4.2	The web end user interface	51
5.2.4.3	Database configuration	53
6	The preliminary testing of the system	57
6.1	The T2I board	57
6.1.1	The input conditioning circuit	58
6.1.2	The U_NTC to Bit circuit	59
6.1.3	The U_NTC to T-BC and T-HI circuit	60
6.1.4	The T2I channel's tests procedure	63
6.1.4.1	The ADC reading's derived temperatures	65
6.1.4.2	The resistance and voltage measurements' derived temperatures	66
6.1.4.3	The temperature regimes	70
6.2	The GSS and OUT boards	71
6.3	The T2I test signals	71
6.4	The GSS test signals	73
7	The Demonstrator implementation	75
7.1	The real-time temperature tests	75
7.1.1	NTC board tester	76
7.1.2	The climate chamber testing	76
7.2	The Demonstrator PEB	80
7.3	The Interlock System strategy	82
7.3.1	The power supply sub-type monitoring	83
8	Conclusion and Future work	85

References	87
Appendices	93
A OUT module logic	95
B Mon-FPGA firmware	97
B.1 Encoding 4b10b	97
B.2 Command packet sequence order	97
B.3 Command packet code	98
B.4 The top VHDL module	98
C The OPC UA server configuration	105
D Design of the HIS interface build	111
D.1 The "CreateDpts.ctl" script	111
D.2 The panel of the creation of the DPs	112
D.2.1 The "createDpts.ctl" button	113
D.2.2 The "configParser.ctl" button	113
D.2.3 "DPs delete" button	114
D.3 The DPs' values display functions	115
E Uncertainties	117
E.1 Steinhart-Hart's temperature	117
E.2 Hysteresis voltage	118
E.2.1 "Broken Cable" hysteresis	118
E.2.2 "ERROR" hysteresis	118
E.3 Voltage divider	119
E.3.1 $U_{NTC}(R_{NTC}, RA, V_{bias})$	119
E.3.2 $R_{NTC}(U_{NTC}, RA, V_{bias})$	119
F Electronic components and cards	121
F.1 The ADS1119 ADC	121
F.2 The LT3042 voltage regulator	122
F.3 T2I OP-AMP's block	123

List of Figures

2.1	Accelerator complex at CERN with the LHC experiments	6
2.2	Schematic of the ATLAS detector [35] with its parts and main sub-detectors magnified	7
2.3	Illustration of the HGTD and its two vessels, each one with two disks with two sides	10
3.1	General Interlock System diagram. The ILK-FPGA shuts downs the PS after receiving danger signals and the Mon-FPGA monitors the system and enables an user interface for debugging and testing	12
3.2	Possible setup of the MIC and LISSY crates of the Interlock System	13
3.3	Representation of the NTC distribution over a quadrant of the front disk	17
4.1	Illustration of the overall monitoring operation of an OUT-20 card	25
4.2	The "Main_CTRL" state machine diagram	26
4.3	Diagram of the Monitoring System	29
4.4	Workbench of the Monitoring System chain	30
5.1	Illustration of the DP creation process inside the "configParser.ctl"	43
5.2	Illustration the Quasar functionality for the Monitoring System	44
5.3	WinCC panel for the DPT and DP creation	44
5.4	Monitoring project's main "datapoints" tree scheme	45
5.5	Example of the LISSY's implementation panel's deconstruction and reference panels procedure	48
5.6	Example of the individual slot panel's deconstruction and channel's reference panels procedure	49
5.7	Example of a LISSY crate implementation panel with the local system layout and the I/O card's test panels	49
5.8	Web end user interface configuration	52
5.9	Archive panel deconstruction	54
6.1	Illustration of the electronics of one input channel of the T2I board.	58
6.2	Temperature hysteresis graph for the T2I channel's input	63
6.3	HI "0"→"1" resistance variation by steps, reflecting a temperature increase on channel 2 ("T2I_S3_C02_temp").	64
6.4	HI "1"→"0" resistance variation by steps, reflecting a temperature decrease on channel 2 ("T2I_S3_C02_temp").	64
6.5	Monitoring panel's and digital word readings' derived temperatures comparison	66
6.6	Monitoring panel's and measurements' derived temperatures comparison	69
6.7	Monitoring panel's temperature as a function of the measured resistances for the T2I channel 2	70
7.1	Active board's sensors temperature readings	78

LIST OF FIGURES

7.2	Ch09 test run of the NTC's and the respective active board sensor's temperatures at each T step	79
7.3	Adapter box illustration	81
7.4	The adapter box and the OUT connector pins	81
7.5	Panel implementing for the final layout	81
7.6	Installed rack setup	81
A.1	OUT output channel schematic	95
B.1	Illustration of the command packet format, received from the backend server	98
C.1	Illustration of the one T2I input channel and the path to the ADC when the channel is open	109
C.2	Right shift on a 8-bit bus	110
F.1	Schematic of on-board T2I ADS1119 with the first four channels	121
F.2	An LT3042EMSE voltage regulator typical application and the T2I application	122
F.3	T2I OP-AMP's schematic	123

List of Tables

- 4.1 Naming convention of I/O cards of the OPC UA configuration file 34
- A.1 Logic inverting process behind the OUT signals 95
- B.1 Table with 4b10b encoding 97
- B.2 "Main_CTRL" process association to the "cmd_code" of the command packet and the emitter enable flags 98

List of Acronyms

ADC Analog-to-Digital Converter.

ALICE A Large Ion Collider Experiment.

ALTIROC ATLAS LGAD Timing Integrated ReadOut Chip.

ARM Advanced RISC Machines.

ASCII American Standard Code for Information Interchange.

ASIC Application-Specific Integrated Circuit.

ATLAS A Toroidal LHC ApparatuS.

BIS Beam Interface System.

CAN Controller Area Network.

CERN Conseil Européen pour la Recherche Nucléaire.

CFM Configuration Flash Memory.

CLB Configurable Logic Blocks.

CMS Compact Muon Solenoid.

Cont_A Continental A.

Cont_B Continental B.

CPU Central Processing Unit.

CSR Control and Status register.

DAQ Data Acquisition System.

DCS Detector Control System.

DHCP Dynamic Host Configuration Protocol.

DNS Domain Name System.

DP DataPoint.

List of Acronyms

DPE DataPoint Element.

DPM Dual-Port Memory.

DPT DataPoint Type.

DRAM Dynamic Random-Access Memory.

DSS Detector Safety System.

E_EXIT Emergency Exit.

EEPROM Electrically Erasable Programmable Read-Only Memory.

EMP Embedded Monitoring Processor.

FDR Final Design Review.

FLEX FLEXible flat tail connecting the detector module with PEB.

FPGA Field Programmable Gate Array.

FSM Finite State Machine.

FSR Full Scale Range.

GND GrouND.

GSS Global Safety System.

GUI Graphical User Interface.

HDL Hardware Description Language.

HGTD High Granularity Timing Detector.

HIS HGTD Interlock System.

HL-LHC High Luminosity Large Hadron Collider.

HMI Human-Machine Interface.

HV High Voltage.

HVP High Voltage Present.

HVP Master High Voltage Present Master.

I/O Input/Output.

I²C Inter-Integrated Circuit.

IC Integrated Circuit.

IDE Integrated Development Environment.

IEEE Institute of Electrical and Electronics Engineers.

ILK-FPGA InterLock-FPGA.

IP Internet Protocol.

ITk Inner Tracker.

JCOP Joint COntrols Project.

JTAG Joint Test Action Group.

LAr Liquid Argon.

LED Light Emmiting Diode.

LEP Large Electron-Positron.

LGAD Low Gain Avalanche Diode.

LHC Large Hadron Collider.

LHCb Large Hadron Collider beauty.

LINAC LINear ACcelerator.

LIP Laboratório de Instrumentação e física experimental de Partículas.

LISSY Local Interlock & Safety SYstem.

LSB Least Significant Bit.

LUT Look-Up-Table.

LV Low Voltage.

LV Stage 1 Low Voltage Stage 1 sub-type.

LV Stage 2 Low Voltage Stage 2 sub-type.

List of Acronyms

MAC Media Access Control.

MIC Main Interlock Crate.

module FLEX small PCB placed at the end of FLEX on the side of the detector module.

Mon-FPGA Monitoring-FPGA.

MSB Most Significant Bit.

NTC Negative Temperature Coefficient.

OP-AMP Operational Amplifier.

OPC UA Open Platform Communications Unified Architecture.

OpenOCD Open On-Chip Debugger.

OS Operating System.

P-MOS P-channel Metal–Oxide–Semiconductor.

PCB Printed Circuit Board.

PEB Peripheral Electronics Board.

PL Programmable Logic.

PLC Programmable Logic Controller.

PLL Phase-Locked Loop.

PS Proton Synchrotron.

PS Power Supply.

PSB Proton Synchrotron Booster.

RAM Random-Access Memory.

RISC Reduced Instruction Set Computer.

SCADA Supervisory Control and Data Acquisition.

SCK Serial ClOcK line.

SCP Service Communication Proxy.

SDA Serial DAta line.

SFTP SSH File Transfer Protocol.

Slot Int Slot Interlock.

SMD Surface-Mount Device.

SNR Signal-to-Noise Ratio.

SoC System on a Chip.

SoM System on Module.

SPS Super Proton Synchrotron.

SSH Secure SHell.

SVF Serial Vector Format.

T2I Temperature to Interlock.

TM Transfer Module.

UFM User Flash Memory.

UPS Uninterruptible Power Supply.

URL Uniform Resource Locator.

USA15 Underground Service ATLAS 15.

USIEC USer Interface for Event Control.

VHDL VHSIC Hardware Description Language.

VHSIC Very-High-Speed Integrated Circuit.

VPN Virtual Private Network.

WinCC OA WinCC Open Architecture.

XML Extensible Markup Language.

List of Symbols

Component's specifications

β	NTC manufacturer B value
R_{25}	NTC nominal impedance at 25 °C

Greek letters

σ	Standard deviation
----------	--------------------

Mathematical expressions

BC	Low temperature ("Broken Cable") interlock signal
HI	High temperature interlock signal
A	Steinhart-Hart equation coefficient A
B	Steinhart-Hart equation coefficient B
R	Generic temperature sensor's resistance
R_{eq}	NTC's equivalent hysteresis resistance
R_{NTC}	NTC's resistance
T	Sensor's temperature
T_{BC}	Low temperature ("Broken Cable") threshold
T_{HI}	High temperature threshold
U_{NTC}	NTC's voltage
V_+	OP-AMP's non-inverting input
V_-	OP-AMP's inverting input
V_{BC}	Low temperature ("Broken Cable") threshold voltage
V_{HI}	High temperature threshold voltage
V_{bias}	T2I input circuitry's reference voltage
V_{CC}	T2I component's voltage supply
V_{in}	T2I R_REF circuitry reference voltage
V_{ref}	T2I ADC's reference voltage

1 Introduction

1.1 Motivation and Objectives

CERN has been in the front-line of science progress and particle physics studies since its formation. Its most powerful particle accelerator, the LHC, is set to undergo modifications, which will empower its performance. The ATLAS HGTD is part of this new upgrade phase, dealing with this power enhancement, causing an increase of beam luminosity. Given the critical role of the HGTD in the new HL-LHC, ensuring its safe and reliable operation is vital for the correct operation of the ATLAS experiment. For that, multiple safety layers secure the overall system. One of them is the Interlock System, considered the last line of defence of the detector, dealing with the temperature monitoring of the HGTD disks and other generic danger signals. It processes those signals, ordering the shut down of the full detector's Power Supply (PS) or just specific sections, while communicating the control station about its status.

This control station, designated as DCS, controls and monitors the status of the detector's disks, in order to ensure its proper operation. The DCS supervises several parts of the detector, beyond the Interlock System. This supervision is possible through the Monitoring System's chain of the Interlock System. This chain gathers information about the status of the system and loads it into a running server, enabling DCS requests of that data to evaluate the current situation of the detector. This evidences three main blocks and concurrent processes: the hardware that collects the local data, the data uploading to the server and the data request for analysis. These processes require a strong and reliable link connection. Only this way, the chain can operate properly.

The main focus of this thesis is the implementation and testing of the Monitoring System, enabling a local control station to supervise an Interlock System's hardware section. It will deal with the base blocks of the system which, by being modular, can efficiently be replicated into more complex systems, such as the final HGTD implementation.

1.2 Methodology and Resources

The work presented in this thesis was done in the framework of the Portuguese ATLAS Group, a LIP association with CERN.

This thesis will be split in order to cover the Monitoring System's three main components. It integrates the Monitoring-FPGA (Mon-FPGA), the monitoring module that gathers the local system's data and loads it into the server. The configuration of the hardware that runs the server where the data is loaded is also a significant part of the work done throughout this thesis. Then this data is accessed by the user on a dedicated project, providing an Human-Machine Interface (HMI) and the control station supervision. The overall system was subsequently scheduled for implementation in an operational and realistic setup, following preliminary hardware testing.

Several elements of the Monitoring System result from a migration of projects from another CERN detector, introduced in Section 3.2. This led to a deepening of the knowledge of the developed hardware and firmware, as well as software proficiency. For the Mon-FPGA integration, the study of its operation had to be expanded, involving FPGA scripting and a dedicated language structure for its firmware development. The same is true regarding the server hardware and the user interface developed for monitoring. Both elements' integration in the local network and their desired configuration demanded knowledge

1. INTRODUCTION

about different operating systems development and dedicated tools, such as Linux Ubuntu and CentOS, SCP communication to move files between systems remotely, the Quasar tool to run a specialized server and a SCADA software.

In addition to the Monitoring System, the Interlock System's hardware needed a dedicated implementation and testing. This required the study of its electronics and of the most suitable methods for testing it, with the available resources.

1.3 Outline

In the following chapters, the tasks related to the implementation of the aforementioned Monitoring System will be presented and explained. Its implementation will be used in the future to, eventually, develop a more complex and suitable system as the final design.

Chapter 2 begins by introducing the broader background of the implementation of the Monitoring System. From a broad spectrum down to a more specific scheme, it will allow to understand where the implemented system fits under such complex experimental environment. This includes the CERN, the LHC and the ATLAS experiment, with a more detailed analysis of the planned upgrades of the LHC, known as the HL-LHC, on which the HGTD will be installed. This detector is where the system, addressed by this thesis is implemented.

In Chapter 3, it is described a layered background to the thesis case-study, the HGTD Interlock System (HIS), providing a detailed explanation of each part. It is split into two blocks, the central and the local systems. Their role and operation on the HIS are explained. The main or central system gathers external signals corresponding to danger situations for the detector. The local systems receive those signals and apply them to the PS of the detector. Additionally, these systems deal with the temperature status of the detector's disks. Both blocks contain a monitoring module, the Mon-FPGA, that executes the aforementioned operations of local data gathering and of loading that data into a server.

In Chapter 4, the Mon-FPGA's implemented firmware is explained as well as its loading process into the FPGA. Although it is still in development by the ITk team, the firmware is considered stable for the present HGTD implementation case. The firmware is described briefly, given focus to other blocks of the Monitoring System. However, it was necessary to study how the Inner Tracker (ITk) Monitoring System's core operates, in order to safeguard any unexpected need of adaptation to the HGTD. This was the case of a requirement's update during the development of this thesis (covered in Section 7.3). This chapter also deals with the server hardware configuration needed to run two functionalities. It not only runs the monitoring server, but also configures the Mon-FPGA. This server hardware was developed to be a replacement for the final board, which is still in development.

Chapter 5 describes in depth the implementation of a SCADA software (WinCC OA) interface that enables the DCS operation for the supervision of the Interlock System. This interface is developed within a project connected to the monitoring server and constantly requests the data loaded in it.

Chapter 6 describes the study and the development of testing procedures for the provided Interlock System's hardware, while using the previously implemented monitoring module and developed interface. These tests not only allowed to test the hardware, but also the implemented Monitoring System and the overall system's unified operation.

And finally, in Chapter 7, a real-case scenario is described, which used the newly implemented local Interlock System and the Monitoring System. It required realistic tests, focused on the temperature safety response. It also details a new strategy approach for the monitoring of the final HIS implementation, due to a new proposal for the system setup. This chapter is the culmination of the previous ones, including the Mon-FPGA's operation, the server link to the Mon-FPGA and to the monitoring interface, the interface

itself and the understanding of the system's hardware. The implemented system is set to achieve a set of requirements for the real-case scenario, such as a thresholds of danger-normal operation accuracy of 1°C, a **NTC** (temperature sensor) accuracy of 1% and a **ADC** accuracy of 0.5 °C.

2 The ATLAS Experiment at CERN

Established in 1954, the [Conseil Européen pour la Recherche Nucléaire \(CERN\)](#), also known as European Laboratory for Particle Physics, resulted from the post-war European science decline, yet from the desire of a united science organization and international effort. This would allow to share the increasing cost of nuclear physics facilities and to advance nuclear science and physics without military and weaponry intentions or politics, others than internal regulations, to stop or decelerate its progress. For that, one of its first directives was an open-access policy, portraying an image of an ethical and collaboration based organization, which still holds until today. It also emphasizes the perpetuation of science and knowledge by training new generations of physicists, engineers and technicians [75]. In 2017 it had more than 17500 people from around the world engaged in a collaborative work environment [56]. Located between the French-Swiss border, [CERN](#) has achieved the title of the world's largest particle physics centre.

[CERN](#) was involved in science and technology's largest advancements in the current and last centuries, such as the creation of the World Wide Web, antimatter observations and the latest, the Higgs boson's discovery [43]. Portugal became a member state of [CERN](#) in 1986.

The [Large Hadron Collider \(LHC\)](#) is one of the engineer marvels of [CERN](#) and is a part of its complex of particles accelerators, working together to unravel the fundamental questions about the Universe.

2.1 The Large Hadron Collider (LHC)

The [Large Hadron Collider](#) is the largest and the most powerful particle accelerator of [CERN](#) complex and of the world. It consists of a ring with 27 kilometres at a depth of 100 meters of superconducting magnets, in order to speed injected particles at nearly the speed of light and to collide them. The collision data is used to study their behaviour and the nature of subatomic particles. It is housed in the former [Large Electron-Positron \(LEP\)](#) collider tunnel and it is under upgrades to pave the way to the [High Luminosity Large Hadron Collider \(HL-LHC\)](#), in which this thesis work will be applied.

Two beams of particles are accelerated in opposite directions. The control of their trajectory and their acceleration boost are executed with unprecedented precision. The particles guidance is mainly controlled by a strong magnetic field, maintained by superconducting electromagnets. Their superconductive nature is accomplished with constant chilling of the magnets down to $-271.3\text{ }^{\circ}\text{C}$ [77], while the [CERN](#) complex ensures the particles proper boosting. It includes many particle accelerators, including the [LHC](#). They all work together to achieve the predefined specifications. Different architecture accelerators are used in different approaches and for different occasions. For instance, linear accelerators allow an initial particle boost, since it depends from a single acceleration pass, but the increase of energy is proportional to the length of the accelerator. While circular accelerators benefit from the repetitive particle trajectory, allowing to constantly boosting until the desirable particle speed, which is why particles repeat the same circuit as long as necessary. Their limitation is mainly the power of the magnetic fields, since the increase of energy of the particles and speed needs more field strength to keep them in orbit. As an example, the [CERN](#) complex of accelerators houses the [Proton Synchrotron \(PS\)](#), [Proton Synchrotron Booster \(PSB\)](#), [Super Proton Synchrotron \(SPS\)](#) and [LINear ACcelerator \(LINAC\)](#) accelerators. The particles are boosted in this chain of accelerators before being injected with the desired energy in opposite directions into the [LHC](#), with the [SPS](#) being the last one [7]. The [LHC](#) is designed to accelerate the

2. THE ATLAS EXPERIMENT AT CERN

protons to 7 TeV, which causes a collision with a centre of mass of 14 TeV.

The **LHC** beams are composed by bunches of protons, achieving a luminosity of $10^{34} \text{ cm}^{-2}\text{s}^{-1}$ [78]. These bunches are then collided at four beam crossing points, where the four **LHC** experiments are settled: the **ATLAS** (A Toroidal LHC ApparatuS), the largest **LHC** experiment; the **CMS** (Compact Muon Solenoid), a multi-purpose detector like **ATLAS**; the **LHCb** (Large Hadron Collider beauty), dedicated to the study of asymmetry between matter and anti-matter, and **ALICE** (A Large Ion Collider Experiment), which studies heavy-ion physics [31].

Figure 2.1 is used to illustrate the set of accelerators and the four **LHC** experiments, also representing its geographical location on the French-Swiss border.

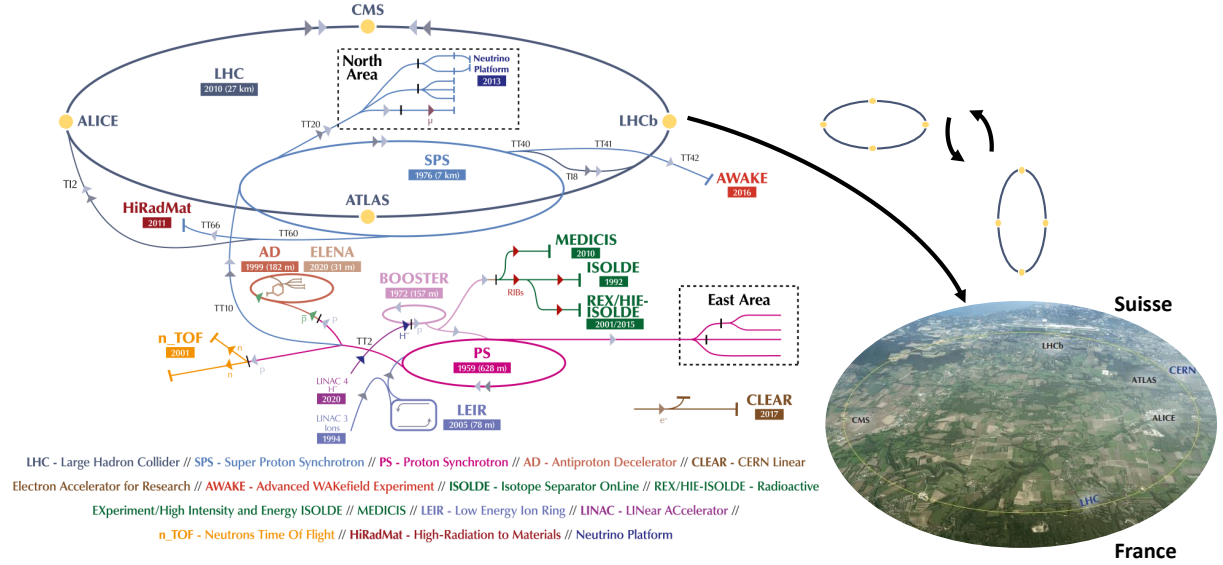


Figure 2.1: Representation of the accelerator complex at **CERN** with the **LHC** experiments. Below the illustration is the legend of the accelerators [74]. At the figure's lower right side is the aerial view of the LHC Tunnel and its position in the French-Swiss border [46]. The rotation between these two schemes is indicated on the upper right.

2.2 The ATLAS Experiment

The **A Toroidal LHC ApparatuS** (**ATLAS**) experiment will be the only **LHC** experiment addressed in this thesis. It is the largest **LHC** experiment and one of the most complex in the world. The detector is located in Point 1 of the **LHC** circumference, in Meyrin, Switzerland and was designed to exploit the maximum potential of the **LHC**, in terms of nuclear physics progress, engineering and electronics.

The detector of the experiment has a diameter of 25 m, a length of 44 m and it weighs 7000 tonnes. It has a cylindrical structure with the particles interaction point at its centre. The interaction point is where the desired particle collisions are forced and the produced particles are characterized, delivering about 1 billion collisions per second, producing 60 million megabytes of data per second, with only a small portion of it representing *interesting characteristics that might lead to new discoveries* [79]. The complex collision selection is executed in real time by the Trigger and Data Acquisition system of **ATLAS**, which picks events with specific characteristic worth of analysis. It is a two-stage process that stores the current data in buffers and executes detailed analysis on it, selecting about 1000 events per second at the end of the last stage [79]. The particles can be characterized by their energy, momentum and charge. By bending the trajectory of the produced particles, it is possible to deduced and measure those characteristics.

The four main sub-detectors of the **ATLAS** can be observed in Figure 2.2. At the figure's upper

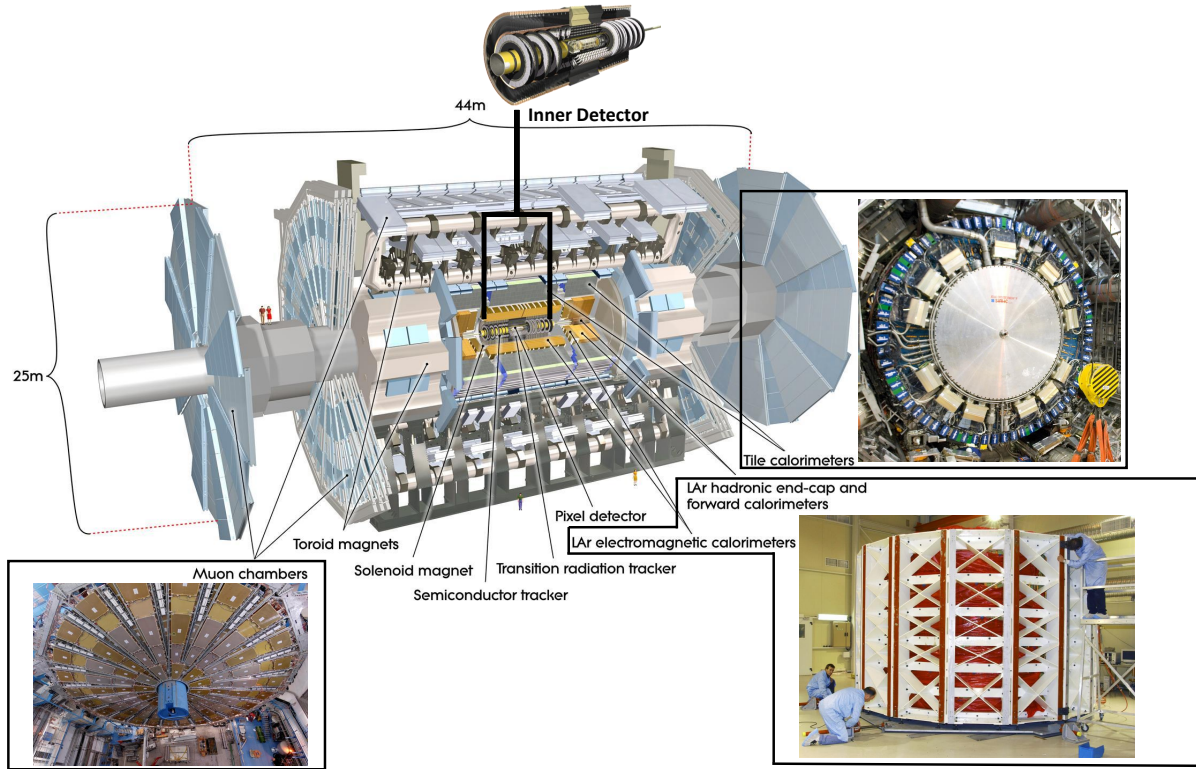


Figure 2.2: Schematic of the [ATLAS](#) detector [35] with its parts and main sub-detectors magnified. At the figure's lower left side is the Muon Spectrometer [49], on the lower right are the [LAr](#) calorimeters [50], on the upper right are the Tile Hadronic Calorimeters [17] and on the upper centre side is the Inner Detector [59].

centre is an illustration of the Inner Detector, located at the centre of the [ATLAS](#) around the interaction point. As its name suggests, it is the innermost section of the [ATLAS](#) detector, itself composed of three complementary sub-detectors: the Pixel Detector, the Semiconductor Tracker and the Transition Radiation Tracker. It has approximately 6.2 m of length and 2.1 m of diameter and in its centre it is applied a magnetic field of 2 T of strength, by a solenoid magnet, which is used to cause the bending of the charged particles' trajectory. Being the most internal system of the [ATLAS](#) detector and the one directly in contact with the collision, it functions as the primary subsystem of the main detector, playing a crucial role on the reconstruction and measurements of the trajectories, as a way to characterize the produced particles [76].

At the figure's lower left side, it is represented the Muon Spectrometer, composed by the different types of Muon Chambers. It is considered the outer layer of the [ATLAS](#) detector, it surrounds the calorimeters and it is responsible for the identification and measurement of the momentum of muons [53].

Then, at the figure's right side, the two calorimeters, the sub-detectors of [ATLAS](#) are represented in charge of the quantification of the energy dissipated of the detected particles. The one below is the [Liquid Argon](#) Calorimeters or [LAr](#) Calorimeter, which accommodates electromagnetic, end-cap and forward calorimeters. It surrounds the Inner Detector and measures mainly the energy of electrons and photons. It measures these particles energy with the current caused by the ionization of the liquid argon between the layers of metal that compose the sub-detector, which absorb the incoming particles. The one above is the Tile Hadronic Calorimeter that surrounds the [LAr](#) Calorimeter and measures the energy of hadronic particles that are not fully absorbed by it. Unlike the [LAr](#), it produces current due to photons coming from plastic scintillators [18].

2. THE ATLAS EXPERIMENT AT CERN

2.3 The High-Luminosity Large Hadron Collider (HL-LHC)

The LHC is subject to a series of upgrades, resulting in the High Luminosity Large Hadron Collider (HL-LHC) which will be its designated successor. It will deal mostly with an increase of luminosity and of the number of simultaneous particle collisions, enhancing the LHC collision occurrence efficiency and, consequently, its potential for new discoveries. Therefore, the ATLAS experiment is on the work to sustain this increase of number of events, which causes the "pile-up problem". This includes the improvements of already existing sub-detectors, such as the Liquid Argon (LAr) and the Tile Calorimeters, as well as of the Trigger and Data Acquisition system. But also includes the addition of new systems, for instance of the High Granularity Timing Detector (HGTD), addressed in Section 2.4. The HL-LHC is projected to have an integrated luminosity 10 times higher than the current LHC, and it is planned to become operational at the beginning of 2029.

2.3.1 Luminosity and the "pile-up problem"

The instantaneous luminosity, $\mathcal{L}$, measures how tightly particles are packed into a given space, that is, the particles density in that space, for instance, in the LHC injected beams. It is one of the measurements used to characterize the beams, since higher luminosity and particle density on the beam result in larger particle collision probability, increasing the chance to observe the desired interaction.

The number of events per second, $\frac{dN}{dt}$ is proportional to the instantaneous luminosity and the total cross-section, σ_i , which represents the likelihood that a desired event will happen [24]:

$$\frac{dN}{dt} = \mathcal{L} \times \sigma_i \quad (2.1)$$

So, it is clear, from equation 2.1, that to increase the number of events, which increases the probability of detecting an event of significance, the luminosity shall increase. Alternatively, it is possible to characterize the number of events with the integrated luminosity, that addresses the total number of events during a period of data-taking [24].

Another measurement used to characterize the experiment is the statistical uncertainty, σ_{stat} , which is proportional to the factor $\frac{1}{\sqrt{N}}$, where N is the number of detected events [84]. The statistical uncertainty, as the name implies, is related to the likelihood of unsuccessful detections. This demonstrates that an increase in the number of events causes a decrease of statistical uncertainty, reducing the probability of missing a desired detection.

Hence, by using these concepts to define the detection capability of the detector, there would be two key approaches to achieve an overall improvement: increasing the instantaneous luminosity, given its proportionality to the rate of observed events (equation 2.1), or by increasing the run time of the detection, which would allow a larger number of collisions to happen. By observing, in both cases, a higher number of collisions, it will be possible to decrease the statistical uncertainty.

As a drawback to a higher luminosity, is the consequential increase of the number of simultaneous particle interactions at the interaction point of the detector. The increase in number of events is known as the "pile-up", meaning a build-up of simultaneous events. As already said in Section 2.2, not all particle collisions correspond to the desired events, thus the need of complex and quasi-optimal data selection ensured by the Trigger and Data Acquisition system of the current ATLAS experiment.

However, the present [ATLAS](#) is not suited for the challenging increase of luminosity, requiring a series of upgrades in order to withstand the analysis and study of much more collisions. The [HGTD](#) is one of the upgrades, which will directly support the [ATLAS](#) experiment resilience regarding the overlap of simultaneous events, by adding a new degree of depth to the collisions' reconstruction.

2.4 The High Granularity Timing Detector (HGTD)

The [HL-LHC](#) is programmed to reach a peak luminosity of $7.5 \times 10^{34} \text{ cm}^{-2}\text{s}^{-1}$ and 4000 fb^{-1} of integrated luminosity, when in its last years of operation the [LHC](#) achieved about 160 fb^{-1} [39]. To cover this desired luminosity, the [ATLAS](#) experiment will endure a succession of improvements split between the Phase-I and the Phase-II upgrades. The [HGTD](#) is a Phase-II upgrade.

The [High Granularity Timing Detector \(HGTD\)](#), based on [Low Gain Avalanche Diode \(LGAD\)](#) sensors, has been in development in order to improve the [ATLAS](#) experiment and support its operation when coping with the most of the upgrades of the [LHC](#).

As the Inner Detector is being improved with a new tracking system, giving it the new designation of [Inner Tracker](#) detector, the [HGTD](#) mitigates the effect of "pile-up" by measuring high-precision timing information of the collisions as a new coordinate, in addition to the 3-D spacial coordinates, to discriminate the events when they occur close in space, but well-separated in time. This will improve the ability to reconstruct the events in the forward region. The expected time resolution at the start of the [HL-LHC](#) lifetime is 30 ps, increasing to 50 ps at the end of its operation [67, 73].

The detector will have 2 hermetic vessels, which will be installed in the region between the barrel and the end-cap calorimeters at the [ATLAS](#) detector (forward region). Each vessel will be about 3.5 m from the interaction point, in opposite directions, with a thickness of 125 mm each [67]. Figure 2.3 illustrates the location of the [HGTD](#), with its two vessels, inside [ATLAS](#). Each vessel has two cooling/supporting disks with two sides each. The sides designations are labelled as front side and back side, for each disk, similar to the nomenclature of the disks themselves, with one front disk and one back disk per vessel. The vessels are referred as A-side vessel and C-side vessel, due to their proximity to specific key marker locations (C-side for Charly pub and A-side for Airport).

The active area of each side is divided into three modular rings, which allows the replacement of the sensors at a small radius, that suffer a higher damage from the radiation of the beam of the upgraded [LHC](#). The replacement of each ring is planned to be done at each 1000 fb^{-1} for the innermost ring (120 mm to 230 mm radius) and 2000 fb^{-1} for the middle ring (230 mm to 470 mm radius). The outer and bigger ring (470 mm to 640 mm radius) will not be replaced [73]. Figure 2.3 demonstrates the magnified active area of one side disk with the modular rings described before and their overlap percentage.

The [LGAD](#) sensors will be installed in each supporting disk, on both sides, and will be read out by dedicated [ASICs](#), designated as [ALTIROC](#), which were specifically designed to be radiation hardened and have low jitter [67]. About 16000 [LGAD](#) sensors will be installed in the two vessels, which will have 3.61 M channels to read. Each sensor is bump bonded to the [ALTIROC](#) readout chip, forming the bare module. Then two bare modules are glued and connected by conductors with a [PCB](#), called [module FLEX](#). Each [module FLEX](#) is connected by a flexible flat tail called [FLEX](#) to a [Peripheral Electronics Board \(PEB\)](#), which is installed on the edge of each side of the disks [42]. Each row on the modular rings will have the [ALTIROC](#) readout chips installed.

The structure of the peripheral boards ([PEB](#)) will be discussed in Section 3.1.2.3.

2. THE ATLAS EXPERIMENT AT CERN

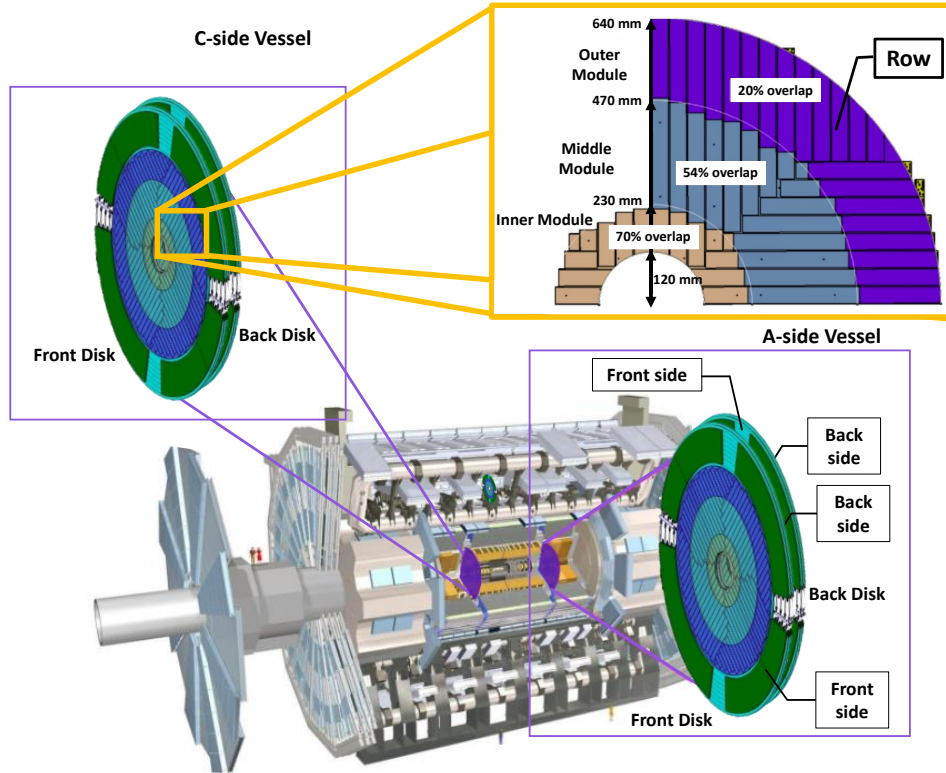


Figure 2.3: Illustration of the [HGTD](#), at the [ATLAS](#) detector, [73] and its two vessels. Each vessel has two disks, each one with a front side and back side. The name of the disks follow that same pattern, having the front disk and back disk, per vessel. The name of the vessels are A-side and C-side, given their proximity to specific key marker locations. To resume, there are, in total, two vessels and four disks each one with a front and back side. On the upper right side is a magnified active area of one side disk with the modular rings described before and their overlap percentage.

The [HGTD](#) transmits two separated data streams: the main is the [Data Acquisition System \(DAQ\)](#), dedicated to the timing data and the second stream is named Lumi, conveying the luminosity data transmitted by a subset of [ALTIROC](#) chips and to the collisions summary regardless of the later selection by the trigger system [73].

This thesis addressees the Interlock System of the [HGTD \(HIS\)](#), which is a safety system for the detector against various type of risks, such as high temperatures and problems with the cooling system. This system will be detailed in the next chapter (3). It does secure the detector by controlling the [Power Supply \(PS\)](#) systems that power different parts of the detector. Each type of [PS](#) system is in charge of specific elements. In order to allow an optimal operation, it was establish the use of individual adjustable voltages. It resulted in two types of systems: the [High Voltage \(HV\) PS](#), rated with 800 V and 6 mA, with the purpose of powering the active area modules of the disks. Then the [Low Voltage \(LV\) PS](#) - stage 1 ([LV Stage 1](#)) and stage 2 ([LV Stage 2](#)) - should power the front-end and peripheral electronics, delivering about 20 kW and it was proposed as a multiple stage system [73]. Both are controlled by the [HIS](#) and the [Detector Control System \(DCS\)](#), introduced in the next chapter.

3 The HGTD Interlock System

The [HGTD Interlock System \(HIS\)](#) is designated as the last line of defence against overheating of some sections of the disks of the [HGTD](#) and against various hazardous situations that can disrupt the correct operation of the detector, such as cooling complications, fire or flooding.

This chapter is supposed to describe the operation of the Interlock System, a system also implemented in the [ITk](#) detector, along with evidence of the similarity between the Interlock Systems of both sub-detectors.

3.1 The Interlock System

The Interlock System is a safety and control system for the different sectors of the disks of the detector. It is split into two hardware systems: the [Main Interlock Crate \(MIC\)](#), a single crate working for the entire detector, and the [Local Interlock & Safety SYstem \(LISSY\)](#) crate, of which several are used. The former is mainly responsible for the distribution of external signals, related to hazardous situations, and the latter includes the processing of local temperature information of each sector of the disks and the safety actions applied to the [Power Supply \(PS\)](#) systems of the disks. Both types of signals, external and local temperature, are then processed by the core of the Interlock System, a module¹ designed around a [FPGA](#), the [InterLocK-FPGA \(ILK-FPGA\)](#), installed in each one of the [LISSY](#) crates. Upon receiving triggers of danger, its programmed interlock matrix proceeds to shut down the necessary [PS](#) powering the disks of the detector. In this way, unsafe conditions to both detector and humans are avoided. Concurrently, all exchanged signals are monitored by another module also designed around a [FPGA](#), the [Monitoring-FPGA \(Mon-FPGA\)](#) module, installed in the two types of crates. It is in charge of gathering and feeding those signals into a server. The connection to this server allows to access the data and to display it for the control station ([DCS](#)) user. This provides a [HMI](#) to the [HIS](#), making possible status updates and control over the system, otherwise unavailable. It also ensures proper testing of the modules in the crates. One of the differences between the MIC and LISSY crate is that, in the first, there are no actions to be applied on any part of the detector, since it is mainly responsible for the clustering and later distribution of the external signals.

The concept of the Interlock System is illustrated in Figure 3.1, with the received interlock signals on the left of the diagram, the interlock matrix at the centre, the monitoring of the system in parallel and the controlled devices on the right [42].

It is important to mention that the Interlock System must be always running, without having to disconnect its components. It must be able to operate even when the detector is powered down when ordered by the [ILK-FPGA](#) or by the [DCS](#). Afterwards, when re-powering the detector or sector, it should be ready to carry on with its function. Also, the operation of the Interlock System should work independently from the Monitoring System, as well as from the [DCS](#), in such a manner that an interlock order sent by the Interlock System can not be overwritten by any [DCS](#) decision. Thus, there should be a reaction delay from the Interlock System, in order to provide enough time for the [DCS](#) to react in coordination. This will be discussed in Section 3.1.3. The interlock logic should be negative, meaning that any open-loop (0 V) should lead to a "INTERLOCK" state [42].

¹A module is a hardware element with a specific function in the system. It can be composed by multiple cards, if they all execute that function.

3. THE HGTD INTERLOCK SYSTEM

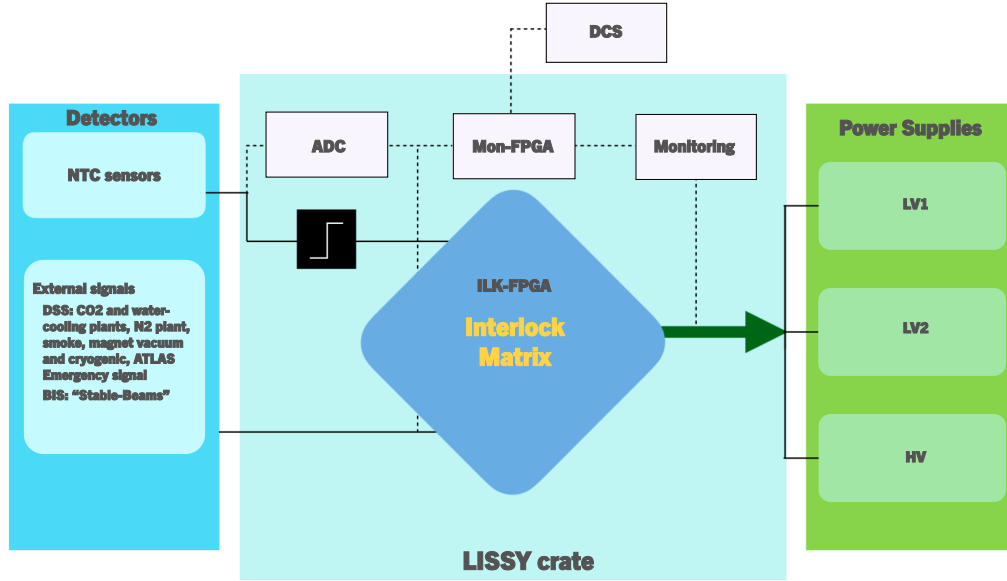


Figure 3.1: General Interlock System diagram[42]. Upon receiving signals of danger from either the local temperature sensors, installed in the disks of the detector, or the external signals, the interlock matrix configured by the **ILK-FPGA** module reacts by commanding the **PS** of the detector to switch off, according to the acknowledged signal. A Monitoring System, controlled by the **Mon-FPGA** is watching the behaviour of the system at all times, with the purpose of debugging through a **DCS** user interface.

The interlock detectors, which correspond to the blocks on the Figure's 3.1 left side, can produce two types of signals. One of them consists on the temperature readings of the disks of the detector, representing a major concern when using silicon based disks. This is accomplished by installing thermistors, **Negative Temperature Coefficient (NTC)** resistors, covering the whole surface of the disks. These were chosen because of their high radiation hardness and the large signal they can produce, allowing their reading over large distances. Thermistors are semiconductor components, which react to temperature variations, changing their electric resistance. When overheated, the **NTC** sensors reduce drastically their resistance. Naturally, when the temperature decreases, the resistance increases. By reading their voltage drop, it is possible to determine their temperature². In this way, the undesirable increase of temperature and possible overheat can be detected by simply reading the voltage drop on the sensors. The analog signals from this reading are then converted into binary signals, describing the two possible states: "INTERLOCK" state, meaning danger for the detector and "OK" state, meaning normal operation [42]. The maximum allowed temperature for a secure system operation is 40 °C.

The other interlock signals are designated as external signals. Apart from the temperature of the disks, there are safety signals related to the state of the detector, transmitted by the ATLAS **Detector Safety System (DSS)** and **Beam Interface System (BIS)**. In the event of fault signals coming from the CO₂ cooling plant, external safety or environmental alarms such as signals from the cooling system, smoke or flammable gas detection, magnet vacuum or failures of cryogenics, like risk from water due to condensation melting, ATLAS emergency signal, flooding and ATLAS wide safe-for-beam interface [42], the Interlock System will receive interlock signals, suspending the activity of the detector. The configuration of these signals according to the sector of the disks where it is associated with will be defined by the **ILK-FPGA** interlock matrix.

For this purpose, there is a distinction to states and corresponding signals, with "OK" and "INTERLOCK" as the states of no danger and danger situation for the detector from any type of origin, respectively, and "ERROR" and "Broken Cable" (Section 3.1.2.3) being types of interlock signals for the

²This will be addressed in Section 6.1.

"INTERLOCK" state. The interlock signal designation refers to any signal coming from the interlock detectors, meaning danger or no danger status.

3.1.1 The Main Interlock Crate (MIC)

Following the signal progression through the Interlock System, it starts with the MIC, the stage where the external signals are grouped and transmitted to the LISSY crates. The crate has standard 19 inches of length and 3U³ height [21], meaning its has approximately 48.26 cm of length and 13.34 cm of height and its design enables communication between cards⁴ via the backplane, allowing faster internal communication, while minimizing unnecessary external cabling. This is one of the benefits of using a crate, besides having a better setup organization, providing storage optimization and reduction. A crate is a metallic or plastic box that can house different types of Printed Circuit Board (PCB) cards. In contrast of the routing of signals through the backplane, each card has front panels allowing external communication with the outside world, with desirable ports. The crate is powered by common voltage outlet, facilitating simultaneous and individual power to each one of the installed cards. The backplane connectors for the cards are mostly generic to serve placement of multiple cards types.

The signal exchange with the **LISSY** crates is done with the **Transfer Module (TM)** module. The **MIC** is composed by that module, along with a **DSS** module, a **BIS** module, **High Voltage Present (HVP)** and **High Voltage Present Master (HVP Master)** modules, as well as with a **Mon-FPGA**. The signals are distributed through the connection of the **TM** module with the **Global Safety System (GSS)** module, installed in the **LISSY** crates, which is described later in Section 3.1.2. Figure 3.2 illustrates the connections between the **MIC** and **LISSY**, in a possible setup. This crate does not take any actions in the Interlock System [21].

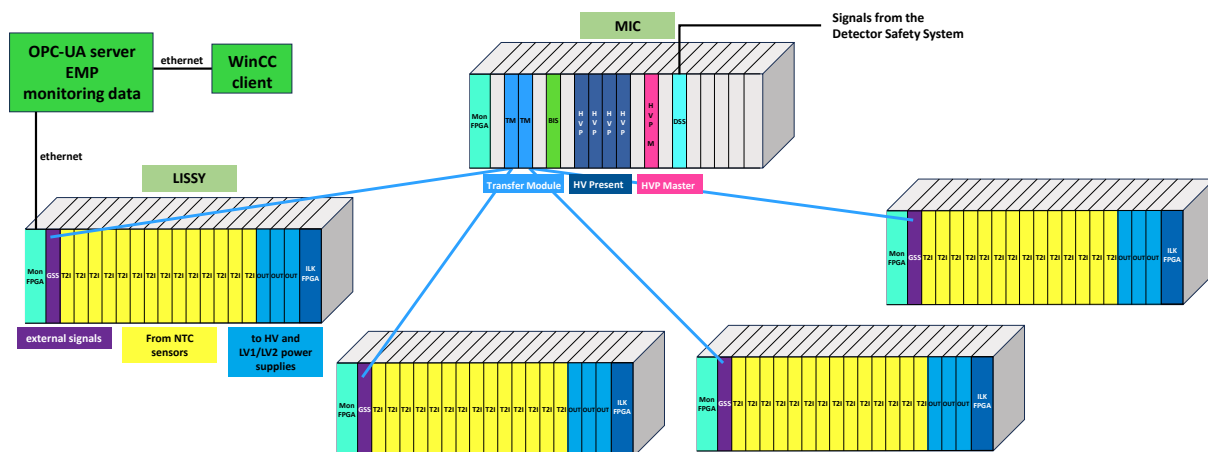


Figure 3.2: Illustration of a possible setup of the **HIS** with one **MIC** and four **LISSY** crates. The connection between the **MIC** and the **LISSY** crate is achieved by the **TM** and **GSS** cards, allowing the transmission of external interlock signals.

To reduce the risk of operation failure of the Interlock System, the MIC should have backup power sources and **Uninterruptible Power Supply (UPS)** connection for continuous operation [20].

The external systems establish communication through the DSS and BIS modules. The DSS module can receive six signals from the DSS⁵. The interlock signal transmission is accomplished by a current

³U is a standard measurement for height of computer enclosures, also known as racks. It stands for rack unit and is equivalent to 1.75 inches (44.45 mm). It is used to characterize the height of racks of 19 inches of length [23].

⁴A card refers to an individually **PCB** which is installed in one of the crates. It is not generic as the term module.

⁵There are cases where the name of the system has been used to designate a specific module. It is specific of modules belonging to only that system, sharing the same functions with it.

3. THE HGTD INTERLOCK SYSTEM

loop controlled by the **DSS** card, which converts it into binary signals, corresponding to the two states available: open-loop means "INTERLOCK" (danger signal) and closed loop means secure situation. Following that, it transfers the interlock signal to the **TM** cards via the backplane. The **BIS** module sets the connection to the signals related to the beam of the detector. Similarly to the **DSS**, the **BIS** of ATLAS updates the Interlock System with "stable-beams" signal, meaning that all sub-detectors of the ATLAS are secure from the beam interference, explicitly the **HV PS**, which are required to be shut down, until the beam is stable. Otherwise, proper signals are sent to the **BIS** module to ensure that the Interlock System stays in "INTERLOCK" state [21].

As already stated, both type of modules send their respective interlock signals to the **TM** card via the backplane of the crate. The **TM** card can receive one signal from **BIS** card, six signals from the **DSS** card and receives three signals from one **GSS** card (six in total, from two **GSS** cards), as update signals [21]. Therefore, it can send 6 signals from the **DSS** and one from the **BIS** to two **GSS** cards simultaneously, and additionally three spare signals to them, such as the **Emergency Exit (E_EXIT)**, **Continental A (Cont_A)** and **Continental B (Cont_B)**. This will be elaborated in Sections 3.1.2.1 and 3.1.2.5 [20].

Unlike the **BIS** module, the **HVP** module receives signals that allow to acknowledge the presence of specific **HV PS**, controlled by the OUT cards in the **LISSY** crates, in order to produce an "Allow-Beam-Injection" signal. This is to be sent to the **BIS** of the ATLAS. The purpose is to notify that the sub-detector has its **HV** channels off, allowing the secure injection of the beam. The **BIS** is then responsible to send "stable-beams" signals to the **BIS** module, as previously mentioned. The **HVP Master** gathers the signals from all **HVP** [21].

The **Mon-FPGA** present in the crate should monitor the operations, as well as send test signals to all of the modules to ensure their proper operation and for debugging [21].

3.1.2 The Local Interlock & Safety SYstem (LISSY)

The main function of the **LISSY** crate is to house the core of the Interlock System, centralized in the interlock matrix, where the interlock commands are send when it receives signals that represent risk for the detector.

The **LISSY** crate is a **MIC** alike crate, with the same dimensions, providing a generic use for the designed crate box. Additionally to the generic **Mon-FPGA**, it has one more module designed around a **FPGA** core, the **ILK-FPGA** module. Besides those, it houses three more types of modules: the **Temperature to Interlock (T2I)**, **GSS** and OUT modules. The number of cards of the **T2I** and OUT modules in the **LISSY** can vary according to the desired layout, whereas the **ILK-FPGA**, **Mon-FPGA** and **GSS** are established to one per **LISSY**. This is one of the reasons that the slots have common connectors in the backplane of the crate, allowing different placements and number of these cards, aside from the **ILK-FPGA** and **Mon-FPGA**. For the two **FPGA** cards there are specific slots for their placement, which are the slots 20 and 1, respectively. The first uses two slots in comparison to the other types of cards and, as a result of that, the **LISSY** can house one less card than the **MIC**: from 21 to 20 slots, of which 18 are generic. As to the generic backplane connectors, each one (dedicated to one slot) can transmit 32 interlock signals dedicated to the OUT card, 16 **T2I** signals and 13 **GSS Input/Output (I/O)** signals [21].

3.1.2.1 The ILK-FPGA

The **ILK-FPGA** is a module with only one card per **LISSY** crate. It controls the reaction of the Interlock System when facing a hazard situation. The card carries a **FPGA** programmed with an interlock matrix, which defines the mapping of the interlock signals to the OUT channels and sends shut down commands to those, powering down the **PS**. The signals it deals with are the local temperature signals, coming from the **T2I** modules, and the external system signals, coming from the **GSS** module. The

majority of the **GSS** received interlock signals, communicated through the **DSS**, are configured by the interlock matrix to be specified by channels in the OUT cards. This is useful to shut down the supply of specific regions of the disks. The transmission of the interlock signals to the **ILK-FPGA**, as well as the interlock matrix commands, is executed through an **I²C** bus from port expanders (MCP23017 [52]), in the **GSS** and OUT cards. Essentially, each input of the MCP23017, operating in the parallel-to-serial mode, in the **GSS** card, receives a slice of interlock signals and then outputs them through only two lines: the **SDA**, the serial data link of those input signals, and the **SCK**⁶, the clock which controls the data transmission. In contrast, the **ILK-FPGA** commands are sent as **I²C** signals and then undergo deserialization by the port expanders in the OUT cards. Thus, there is a serialization and deserialization process of the interlock signals and **ILK-FPGA** commands, respectively. This process avoids an excessive number of **I/O** lines to be routed in the backplane.

Additionally, there are the **Slot Interlock (Slot Int)** signals, the **E_EXIT**, **Cont_A** and **Cont_B** signals, that do not rely on the port expanders. They are still communicated from the **GSS** module to the **ILK-FPGA** module through the backplane, but the **E_EXIT** is not processed by the matrix and the three are not read by port expanders, having direct lines in the backplane and cards. The **E_EXIT** signal orders to shut down all **PS** and by bypassing the matrix, it ensures a faster distribution on all crates and a higher reliability, in case of a matrix, **FPGA** or port expander malfunction, representing an important but last case situation signal. The same as the Continental signals, that bypass the port expanders, ensuring a higher reliability.

The local temperature readings are communicated by the **T2I** module to the **ILK-FPGA**. By reading the high temperature and open channel signals, the interlock matrix proceeds to shut down the corresponding **PS**, through the port expanders and the backplane, similar to the **DSS** configurable signals. The front panel of the **ILK-FPGA** module has two pairs of **LEDs** (a red and a green). One of the pairs corresponds to the Global Interlock signal status. The red turns on when the **E_EXIT** signal is activated and the green turns on if this signal is deactivated, even if the system is in "INTERLOCK" status, by some other signal. The second pair corresponds to the **FPGA** operation. An Ethernet port is also implemented for access to the **FPGA** in order to communicate with it, using developed tools, and a **JTAG** port is below it for firmware reload. At last, a Sub-D9 female connector is used as **CAN** port for crate-to-crate communication, between **LISSYs**.

3.1.2.2 The Mon-FPGA

The **Mon-FPGA** module has two main requirements: the monitoring of the crate it is installed in and the testing of the cards connected to the backplane, whether installed in the **MIC** or in a **LISSY** crate. The centralization of the firmware for both type of crates allows an efficient update process and continuity for later projects.

For the **LISSY** crate, the **Mon-FPGA** reads the **I²C** bus of the **T2I**, **GSS**, OUT cards and direct **Slot Int** lines, through the backplane, in order to load the current information to a server. Additionally, it also reads the **LISSY** power supply status⁷. From the **T2I** card it reads the analog temperature values from the sensors and the interlock signals. The interlock signals from the **GSS** signals are also read. The **ILK-FPGA** commands are read before being applied to the **PS**, in the OUT card. Each card has AT24CS02 [10] **EEPROM** chips (including the cards in the **MIC**), which are non-volatile memory chips capable of storing and saving data until later overwrite. Each **EEPROM** chip has information of the card it is installed in. It is configured with that data and should be able to be reconfigurable as intended. Besides

⁶The **SCK** can also be referred as **SCL**.

⁷This specification is not going to be addressed in this thesis.

3. THE HGTD INTERLOCK SYSTEM

the current data reading, the **Mon-FPGA** should also read these chips and load the information they store in the server for easier identification. The readout of **EEPROM** chip follows an established procedure. The first byte of the chip indicated the type of card it is reading. This could correspond to a **T2I**, **GSS**, **OUT**, **TM** or **HVP** card type. The second byte should correspond to the card variant, usually being the number of channels the card has. Additionally, for the **OUT** card, the second byte has space to specify which **PS** the card is associated to, in the case of using of an **OUT** card per sub-type (**HV**, **LV Stage 1** or **LV Stage 2**) [70]. The data from the crate loaded in the server is then accessed by the **DCS** user.

The tests should be accessible by the **DCS** user and applied in specific testing periods, mainly when there is no data being taking in the experiments. They are sent in an **I²C** bus to separated **I/O** expanders in the receiver card. The **T2I** test signals are transmitted as 16-bit commands from the **Mon-FPGA**, where the **LSB** corresponds to the input channel 1 and the **MSB** to the input channel 16. Each bit of that sequence controls the test signal value: bit "0" to a deactivated test signal and bit "1" to an activated test signal. The **Mon-FPGA** should test each **T2I** channel by setting "ERROR" signals in any of the input channels and the user should be able to verify the system response to it, by switching the interlock signal in that **T2I** channel and switching off the corresponding **OUT** channel. The test signals of the **GSS** card are transmitted through a 10-bit command and based on the same method as the **T2I**, by forcing "INTERLOCK" status on the signals communicated to the interlock matrix and, consequently, the user should be able to verify the interlock signal and the **OUT** channel reaction.

There were developed various versions of this module, mainly changing the core **FPGA** being used. In result, there are different firmware versions for them (Section 4.2). The front panel of the module has one **JTAG** port to reconfigure the **FPGA**, with two red **LEDs**. Both keep turned on with the crate powered on and the upper **LED** blinks once when the **FPGA** is being loaded with the firmware. Below it has two rotating switches, to define a crate ID number (ID with two identifiers) and one Ethernet port for communication with the backend hardware running the server, the **Embedded Monitoring Processor (EMP)**. Its dimensions are standard 3U of height and 160 mm of length (Eurocard **PCB** standard dimensions⁸).

3.1.2.3 The T2I module

The **T2I** module assembles the temperature sensors (**NTCs**) readings, ensuring protection for the **LGAD** sensors and other parts of the disks when overheated. There will be one **NTC** per **module FLEX**, which is connected by a **FLEX**. The **FLEX** is a flexible flat tail that allows to connect the **module FLEX** to a **PEB**, located on the edge of the supporting disk [42], which then transmits the sensors' signal to the **T2I** cards. This chain⁹ makes possible the readings of the temperature sensors by the Interlock System, through the **T2I** module. There are two variants of this card: a **T2I-12** and **T2I-16**, with 12 and 16 input channels for the **NTCs** readings, respectively. Each input channel should carry only one **NTC** reading. The desired configuration would require 896 **NTCs** for the **HGTD**. This would be equally distributed per disk and per side [42]. With two vessels, each one with two disks of two sides, there would be 448 **NTCs** per vessel, 224 per disk and 112 per side. Section 2.4 details the **HGTD** vessel arrangement and Figure 2.3 illustrates the structure behind the vessels.

This configuration favours equal **T2I** channelling distribution for each disk side. The distribution of **NTCs** per **PEB** can vary, according to the most optimal solution for every part of the system. Figure 3.3 exemplifies the **NTC** distribution over a quadrant of the front disk. Each disk has two sides with the same **NTC** distribution: over 112 sensors per side, meaning 28 per quadrant of a side. On the left is represented a quadrant of the front disk with each side against each other, as it will be implemented. On

⁸Eurocard **PCB** dimensions is an European standard format for **PCB** development.

⁹The indicated chain goes from the **NTCs**, installed in the **module FLEXs**, which are then connected to **FLEXs**, that makes the connection to the **PEB** possible.

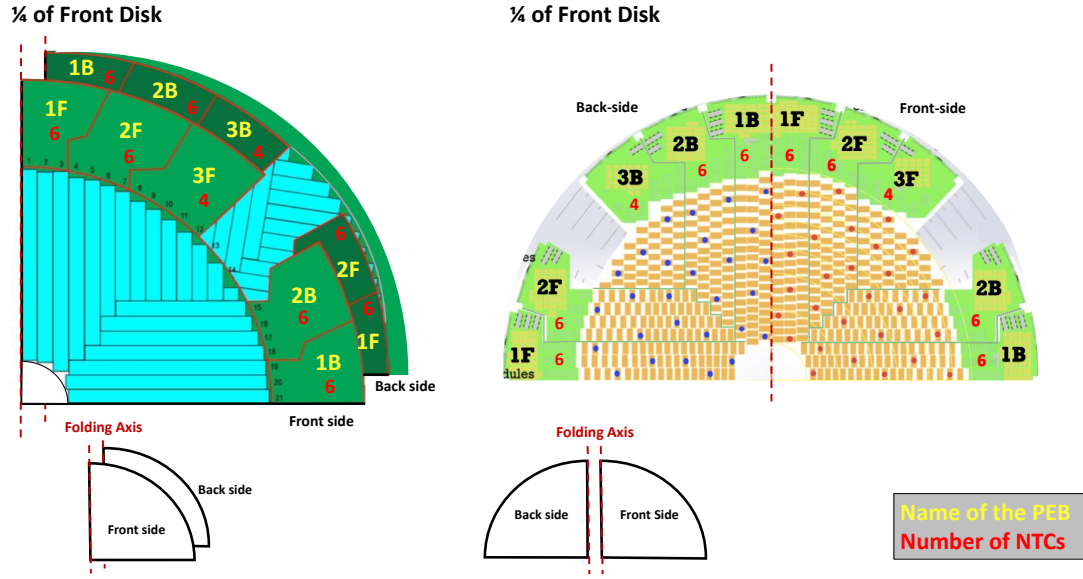


Figure 3.3: Representation of the NTC distribution over a quadrant of the front disk, on both sides. On the left [48] is the quadrant of the disk with both sides against each other, similar to what will be implemented. On the right [42] is the unfolded representation of both sides of that same quadrant, mirroring each other. Each PEB has a maximum of 6 associated NTCs and a minimum of 4 NTCs. Each side has 112 sensors installed, meaning that each quadrant has 28 sensors on each side. At the figure's bottom is a generic representation of the folding of each side quadrant of the above representations and the legend of the illustrations on the bottom-right.

the figure's right are represented again both quadrants but unfolded, mirroring each other. Each PEB has a module name with a number and a letter. PEBs with the same design have the same name designation, as it can be verified by mainly the right representation, with 1F, 2F, 2B and 1B having the same amount of sensors associated and the same PEB design. The amount of necessary cards for the proposed 224 sensors per disk is subjective to the variant of T2I card being used (14 T2I-16 cards or at least 19 T2I-12 cards, leaving 4 unused channels). The final implementation of the work described throughout this thesis will be with a PEB board, designated Demonstrator, addressed in Chapter 7.

The comparison of the local temperature signals states is executed in the T2I card. Each channel has saturated OP-AMPs that define an upper and lower value for the normal state of operation of the disks. The higher temperature value admitted is 40 °C and the lower is at -60 °C [21]. It is worth mentioning that not even the chosen NTCs go below -40 °C, meaning that they are not able to achieve this lower value defined by the T2I card. This was implemented using the equivalent voltage drop that the sensor theoretically would need to detect that temperature. This is a way to detect anomalies in the circuit in the implemented electronics, leaving a safety margin. It is believed to be a good practice in the development of such electronics. Hence, when the temperature of the sensors go above 40 °C, the corresponding off-limit T2I channel, sends the "ERROR" signal to the interlock matrix, that way triggering the interlock¹⁰ in the respective OUT channel. The Interlock System is also prepared to detect open-loop channel, designated as "Broken Cable". Thus, each T2I channel has two types of interlock signal, the "ERROR" for high temperatures and another called "Broken Cable", for open-loop channel, meaning a loss of communication or when it detects an anomaly causing an equivalent temperature below -60 °C. Both mean danger to the system. These signals are read by two sets of port expanders, which transmit them independently and simultaneously to both Mon-FPGA and ILK-FPGA. The Mon-FPGA should be able to force an "ERROR" status in the OP-AMP of the upper limit temperature to test the operation of the

¹⁰The interlock of a channel is applied when the system reacts in a danger situation in that channel, thus shutting down the corresponding PS.

3. THE HGTD INTERLOCK SYSTEM

system. This verifies the **T2I** channel being tested, which should trigger a chain reaction, reaching the corresponding OUT channel. The hardware electronics will be studied in-depth in Section 6.3, where tests on a **T2I-12** card were conducted. At the same time, the sensors are also read by **ADCs** installed in the **T2I** card. This converts the analog signals of each sensor into digital word to be read by the **Mon-FPGA** module as the local temperature values, providing complementary information to debug out of limits situations. The **T2I-12** has four **ADCs** installed and the **T2I-16** has five. Both card variants have one **ADC** dedicated to the reference temperature thresholds and to the temperature of the **T2I** board itself, through a specific temperature sensor installed in the card. Each one of the other three and four **ADCs** installed, respectively, reads the voltage drop of four **NTCs** through the **T2I** channels.

The front panel of the **T2I** module is based on Sub-D connectors¹¹, with 3 Sub-D9 or 1 Sub-D37 for the **T2I-12** or **T2I-16**, respectively, and are only used for the temperature sensors signal transmission.

3.1.2.4 The OUT module

The OUT module routes the received **ILK-FPGA** interlock commands to the corresponding **PS**, through its output channels. When in "OK" state, it provides at least 3.0 V to each output signal. When it receives signals equivalent to "ERROR" or "Broken Cable" from the **ILK-FPGA** matrix, it drops the voltage to below 0.45 V [21]. This logic control is executed with individual OR logic gates for each output channel. One of the gate's input is from interlock matrix's signals and the other is the external signal that affects the whole card (**Slot Int**). When both signals mean no danger, the gate does not drive. When at least one is at "INTERLOCK" state, the gate drives 3.3 V. The gate output has positive logic, meaning that when there is no danger signals and the channel is at "OK" state, the gate delivers 0 V. When at least one of the signals mean danger, the gate delivers 3.3 V, meaning "INTERLOCK" status. However, the Interlock System has negative logic, which is characterized by the association of 0 V to the bit "1" and 3.3 V to "0", when "1" means "ERROR" and "0" means "OK" state. Mainly due to the cut of power of the **PS**, meaning "INTERLOCK". To invert the OR gate logic back to negative, its output is controlled by a current limiting circuit implemented with an inverting buffer transistor (**P-MOS**), that limits the output current to approximately 35 mA [21]. That way, when the gate drives 3.3 V at its output (for the "INTERLOCK" status), the remaining circuitry cuts the current at the channel output, forcing the corresponding **PS** to shut down. When the gate drives 0 V, the channel output provides at least 3.0 V. The logic inverting process of the output of the OUT channel is presented in Appendix A.

Simultaneously, the **Mon-FPGA** should be able to read the state of the channel at the output of the OR gate. There are 3 variants of the OUT card: the OUT-20, OUT-24 and OUT-32, with 20, 24 and 32 output channels, respectively. Their major difference is the number of channels as well as the connectors it uses and, consequently, the number of pins dedicated to a common **GND**. The OUT module also has Sub-D connectors for each variant in the front panel: 1 Sub-D37, 3 Sub-D9 and 2 Sub-D15 for the OUT-32, OUT-24 and OUT-20 cards, respectively. The sequence of signals from the interlock matrix are received as **I²C** buses and read by port expanders, causing a deserialization and, consequently, an individualization of the commands from the **ILK-FPGA**.

3.1.2.5 The GSS module

The **GSS** module is the interface to the **MIC**. It establishes the communication between crates, through cabling with the **TM** module on it. It can transfer the external system signals to the rest of the cards on the **LISSY**. As previously explained, the external signals are related to generic danger situations. Each card has a Sub-D15 connector to 1 **TM** card, allowing 10 input channels, 3 output channels

¹¹The name comes from the D shape of the connector and from the number of pins of the connector.

and the remaining 2 pins for a common **GND**. The 10 inputs handle the external signal transmission from the **TM** and each one is connected to optocouplers with 2.2 k Ω resistors. The inputs follow the negative logic of the Interlock System, meaning open channel (below 0.8 V for inputs) corresponds to "ERROR" and bit "1" [21]. The first input, designated as ISO-IN1 is dedicated to the **E_EXIT** signal. The second (ISO-IN2) and third (ISO-IN3) are dedicated to the **Cont_A** and **Cont_B** signals, respectively, and ISO-IN4 to ISO-IN10 are dedicated to the configurable interlock signals.

The **E_EXIT** is the most wide signal, affecting the whole detector. When triggered by the **DSS**, it commands the shut down of all **PS**. Its effectiveness lies on the bypass of the interlock matrix and the port expanders. Similarly, there are the **Cont_A** and **Cont_B** signals. These three signals are designated as **Slot Int**, meaning that when triggered, they shut down an entire slot, which corresponds to an OUT card, independent of its channel distribution per **PS**. When in a general shut down, the **E_EXIT** triggers the **Slot Int** of all connected OUT cards in the crate. Unlike the Continental signals, their signal association to the desired slots is still processed by the interlock matrix. Nevertheless, they are used to bypass the port expanders, being more reliable than the configurable signals, in case of a port expander malfunction, by not depending on them [21]. Unlike the **T2I** module, the **GSS** interlock signals have only one type of signal, this being the "ERROR", even if triggered by open-loop on that channel. The "ERROR" signal is driven by OR gates, with the **TM** and the **Mon-FPGA** test signals as its inputs. The **Mon-FPGA** is, therefore, able to force interlock signals on the **GSS** channels.

In parallel, all interlock signals sent to the **ILK-FPGA** are monitored by the **Mon-FPGA** by separated I/O expanders on the **GSS** card.

3.1.3 The Detector Control System (DCS)

The **DCS** should ensure the coherent and secure operation of the **ATLAS** experiment and its sub-detectors, allowing an interface between all of them. The **DCS** deals not only with the **HIS**, but also with the Interlock System of other sub-detectors, as the **ITk** detector and other systems controlled externally, like the **LHC** accelerator, the **CERN** technical services, the **ATLAS** magnets and the **DSS**. For that, the system should be able to set the detector in question in any desired operation state, to monitor the current state of it, as well as to store any operational parameters. This supports, not only, current monitoring, but also historic tracking to detect any abnormal behaviour on the detector. This should provide enough data to take the correct action on the system. Only through this a smooth and safe operation of all sub-detectors is achievable. To secure the synchronization of the state of the detector, a bi-directional communication between the **DCS** and the running systems should be implemented [16]. Having a generic control system, as the **DCS**, applied to the sub-detectors of the same experiment, facilitates the integration and element standardization of all systems involved.

The relation between the **DCS** and the Interlock System is coordinated by a **DSS** time interval, as it follows: the actions of the Interlock System can not be overwritten by **DCS** decisions and, as a result, the Interlock System is the last line of defence of the **ATLAS** sub-detectors, being independent of the **DCS**. Hence, the interlock actions should be taken after a predefined period established by the **DSS**, enabling the processing of any **DCS** decision before acting. This allows the **DCS** to take action before the Interlock System in order to deactivate the necessary parts of the detector in an organized way, without possibly damaging it by shutting them down in an uncoordinated approach. After that decision time, the Interlock System should act independently of any **DCS** decision and action.

The interface between the **DCS** and the Interlock System should be accomplished with a **WinCC OA** project, that requests the data of the system in the server running on the **EMP**. The data was previously loaded by the **Mon-FPGA** of each crate. The **DCS** should also be able to test and control the Interlock

3. THE HGTD INTERLOCK SYSTEM

System through the [Mon-FPGA](#).

3.2 The ITk Interlock System

The [HIS](#) will be a product of an adapted Interlock System from the [ITk](#) detector, following a fitting strategy to the [HGTD](#) needs and features. The Interlock System was developed by the [ITk](#) team as a versatile system, which can be migrated and adapted to other [ATLAS](#) sub-detectors, like the [HGTD](#). This is exemplified by the generalization of both hardware and firmware applications. For instance, the multiple purpose of the developed crate, being used as [MIC](#) and [LISSY](#), as well as its backplane, allowing different card setups. It also allows different [LISSY](#) crates with layouts singular to the detector in cause. The difference between Interlock Systems lays on the requirements for that specific detector, which impacts all these parts of the system.

The [ITk](#) Interlock System should encompass 60 [LISSY](#) crates and 1 [MIC](#). The responsibility of its development was shared between teams of the Montreal and Wuppertal universities, with most of procurement and production being organized in Germany. The Wuppertal teams are responsible for the [T2I](#), [GSS](#), [OUT](#), [ILK-FPGA](#) cards and backplane of the [LISSY](#), which includes their production and quality control and assurance of all parts. The Montreal team is mainly in charge of the [Mon-FPGA](#) and its requirements [21]. Section 3.1 described the Interlock System as a universal system for both sub-detectors. Both Interlock Systems will be distributed in the service cavern [USA15](#), together with the hardware of other systems [34].

4 The hardware of the Monitoring System

The **Mon-FPGA** is the core of the monitoring of the Interlock System. This chapter will introduce the module focal point, an **FPGA** and its reconfigurable architecture, the **VHDL** language and details about the firmware operation. It requires the reading and transmission process of packets of crate data, while the server is constantly updated with the monitoring information. The firmware was developed by the **ITk** Montreal team in charge of the **Mon-FPGA** development, and is accessible in the centralized gitlab repository [41]. It is still in a development phase, although it was stable at the time writing of this thesis. It was mainly developed with the **VHDL** description language, dedicated to **FPGA** configuration, which is based on a hierarchical module structure, with a top **VHDL** module interconnecting the individual ones. The **VHDL** code is synthesized into a bit-file which is loaded in the **FPGA** by a **JTAG** cable. There is also a centralized gitlab dedicated to the **Mon-FPGA** of the **HIS** [12].

As an alternative of the final hardware - still under development - which will run the server, a board with a **SoC** chip was used to reconfigure the **FPGA** and run the server. The preparation of the **SoC** to execute both functions will be addressed in Section 4.6. The conjunction of these two parts of the system composes the designated Monitoring System's chain, which allows an user supervision of the system through a designed user interface.

4.1 The FPGA

Field Programmable Gate Arrays (FPGAs) are widely use due to their flexibility and processing speed. They are composed by reconfigurable logic blocks that interconnect with each other, making it very versatile and fitting for logic based projects. By being reconfigurable, they are ideal for prototypes' implementation, complex or not, due to their development with incremental modifications, as it is the case of the **Mon-FPGA**. They do not have the limitations that **CPUs** or microprocessors have, since they are already programmed to execute a combination of tasks. A **FPGA** can be reconfigured as intended and by not executing functions that it is not meant to do, allows fast and specific data processing tasks. Its main limitations derive from the number of logic gates it has, as well as their memory capacity and processing speed, even if they can do some processing faster than most microprocessors. These factors can vary with each model, but they can be found in numerous architectures, for instance with embedded chips, such as **ARM** cores, **PLLs** or **DRAM** blocks. Consequently, the Interlock System is a **FPGA**-based system, and the same is true for the Monitoring System. Both **FPGA** modules are required to process a large number of input signals and data transmission packets, therefore requiring flexibility for its development. Thus, the processing speed of the Interlock System is not a limitation, as the **FPGA** is optimally chosen for the design requirements.

The testing of the firmware can be done by simulating the desired signal behaviour in a testbench, through additional code. This avoids investing unnecessary time and costs in firmware tests with physical hardware, before applying it to the final product. Given these factors, **FPGAs** are widely use to project digital systems that required several development stages, speed and specification to certain functions.

4.2 The Mon-FPGA version's history

Through its development were produced several model versions of the **Mon-FPGA**. One of the main causes was the general chip shortage in 2020, due to the COVID-10 pandemic effects. Chip manufactur-

4. THE HARDWARE OF THE MONITORING SYSTEM

ers had reduced or stopped production of certain chips, forcing the use of old stock devices and different versions/models for different purposes, in order to maintain the development of the card. The version used for this thesis was the [Mon-FPGA](#) version 3.0, still a prototype, which has a mezzanine configuration where the [FPGA](#) chip is installed, like the [ILK-FPGA](#). Its connection to the outside and the carrier card is done through the mezzanine connectors, linked to the card where it is installed.

The initial design involved an Altera MAX10 [FPGA](#). Due to the chip shortage and given the uncertainty of its post-pandemic production, also occurring to other chips from various manufacturers, such as Xilinx and Trenz mezzanines, it was developed a backup alternative, with an Altera Cyclone EP1 [\[30\]](#) (first generation Cyclone [FPGA](#)). There was a small stock of them with the [ITk](#) team, a surplus from previous projects. Its long usage had shown consistent results and proven to be a reliable choice for the card pre-production, despite their end of production and support from the manufacturer at that time. This was the version used for the development of this thesis. For this version, the team always had a future plan of an afterwards device improvement. Later it was also developed another pre-production model with another Altera [FPGA](#), the MAX10, with two different models. The final product, still in development, will use an Altera Cyclone 10LP. Until the moment of this thesis publication, there are 3 firmware versions for the Cyclone EP1, for the MAX10 M25 and for the MAX10 M50, [\[51\]](#). The Cyclone 10LP version (models 10CL08/4 [\[25\]](#)) has a strong compatibility with the former developed firmware versions. There were other alternatives, such as using a similar [ILK-FPGA](#) Trenz mezzanine, an industrial fabricated mezzanine with a [FPGA](#) already installed. It would have an AMD Xilinx Artix-35 [\[36\]](#) [FPGA](#) and the firmware migration was considered as simple, but its production was discontinued, so it was considered as a replacement [\[72\]](#).

4.3 The VHDL language

The main description languages ([HDLs](#)) for [FPGA](#) configuration are Verilog and [VHSIC Hardware Description Language \(VHDL\)](#). Their differences lie on the coding structure, with the first being a hardware modelling language, closer to C software language and easier to learn, and the latter with syntax resembling the Pascal scripting [\[81\]](#). [Hardware Description Languages \(HDLs\)](#) languages are known as behavioural languages, since they reconfigure the hardware and are closer to it than other coding languages. They allow to change the hardware through its reconfiguration, instead of being used to program an already configured processor to execute various functions.

The language used to implement firmware for the [Mon-FPGA](#) is [VHDL](#). This [HDL](#) was originally created by the Department of Defense of the United States of America and is widely used until today, being subject to multiple standardizations by the [IEEE](#) [\[40\]](#). The [VHDL](#) code can be written and synthesized with an [IDE](#), usually provided by the [FPGA](#) manufacturer, like Vivado [\[82\]](#), from AMD/Xilinx or Altera Quartus [\[33\]](#) from Intel.

The [HDL](#) code is not necessarily sequential, unlike many common computing languages. Excluding typical loop and conditional structures, its concurrent block operation nature makes the [HDL](#) languages highly depended on the temporal aspects and, as so, strongly rely on delays. Apart from that, it also has a hierarchical architecture, achieved by using the [VHDL](#) modules organization. A [VHDL](#) code module is designated as a segment of code that describes a digital circuit. The hierarchical structure of [VHDL](#) coding allows the use of multiple individual modules, working together and controlled by a top module. This aspect facilitates the connection between circuits and sub-circuits described by code modules, making the [FPGAs](#) a privileged development platform due to their versatility. Besides, digital systems can often use the same blocks multiple times. Also, most manufacturers provide templates for firmware development, which reuses these generic digital blocks, such as multiplexers and counters.

The **VHDL** module is structured with an external and an internal block of code. The external, also designated as **entity**, usually describes the digital circuit interface, by defining the ports of the circuit and their type of signals. It should also define their nature, for instance if they are bidirectional, a buffer port, input or output. This interface can then be used by several cases by internal blocks, the **architecture**, which defines the actual function of the circuit and how it is going to behave.

Another important **VHDL** element when developing large codes with a considerably amount of interconnected modules, as it is the case of the **Mon-FPGA** firmware, is the **component** instance. It is a very common element to manage the hierarchical module structure in the firmware. It allows to break a larger code into smaller parts, integrating them into multiple grouping modules and so on, successively, until the top module is achieved. The **component** is a declaration of an usual **VHDL** code submodule, with its entity and architecture already defined in a dedicated code file. Then, the top module code is used to declare these types of submodules for a coordinated operation, with a port signal type declaration, similar to the **entity** declaration of the each module. The interconnection of the **component** with the rest of the code and the other components is accomplished by instantiating the **component** in an architecture similar code structure. The **component** declaration and instantiation¹ are executed in the top module **architecture**, before and after the **begin**² statement, respectively.

The code that is going to be loaded into the **FPGA** has to be synthesized first, in order to produce a bit-file. This file is then usually loaded by the manufacturer software. There are other loading methods, as was the case of the **Mon-FPGA**, which is detailed in Section 4.6.2. The **JTAG** interface is commonly used for on-board tests, to debug logic behaviour and to configure devices such as **FPGAs** and, therefore, to load the synthesized bit-file in the chip.

In the module's **entity**, the ports and variables should be declared by their type of signal, usually using the standard libraries functionality. This comes in a range of types with restrictions for specific scenarios. For instance, arrays of bits are broadly used and they have 3 formats: the **bit_vector**, which defines an array of only binary values "0" or "1", the **std_logic_vector** and **std_ulogic_vector**, that define a set of 8 and 9 logic values³, respectively, including the binary signals. The additional logic values are used for simulation purposes [32]. The format of the vectors follow the same pattern, using the ranges of (**MSB** downto **LSB**), with **MSB**>**LSB** or (**MSB** to **LSB**), with **MSB**<**LSB** and the length of the vector is given by $|\text{MSB} - \text{LSB}| + 1$. There are other vector types and their use differs from the preference of the developer and the scenarios they are being applied to. Additionally, there are single value types such as the **std_logic** or **std_ulogic**, both to represent one logic value for the standard **IEEE** [32].

On a broad view, the **VHDL** provides code for synthesis and code for simulation and verification. The latter is usually called the testbench. The simulation of the modules is done with additional **VHDL** code, which is not synthesized by the hardware compiler and thus not loaded into the **FPGA**. The simulation of the modules allows to verify their response when receiving the specified inputs and to confirm their behaviour. The files that execute the simulation testbench do not have any entities and are often identified by time dependent instructions, such as clocks and **wait** statements.

¹Declaration and instantiation are understood as the specification of a component's behaviour with **VHDL** code and the subsequent creation of one or several instances of the component, respectively.

²This command starts the architecture's behaviour specification.

³The values of **std_ulogic_vector** can be "0", "1", "X", "Z", "W", "L", "H", and "-", while the **std_logic_vector** additionally accepts "U" as a value.

4. THE HARDWARE OF THE MONITORING SYSTEM

4.4 Firmware Requirements

The **Mon-FPGA** firmware requirements are a result of various meetings with the users of the **ITk** project and they can be found in the technical document written for this purpose [19].

The firmware implements a central state machine, a very common scenario for **FPGA**-based digital systems. It is a behavioural model called **Finite State Machine (FSM)**, as it uses a finite number of states. Each state triggers actions (output signals) and waits for inputs that will define the next state transition. Usually **FSM** starts in the "idle" state, when the machine is waiting for a specific signal to start its operation and take some actions, such as one as simple as turning on a **LED**.

The **Mon-FPGA** state machine encodes specific sequences of **I²C** commands suited for the readout of each type of modules. By reading the information in the **EEPROM** chips of the connected cards, it selects the appropriate **I²C** command for that card type data reading. This procedure is stated in Section 3.1.2.2. Those **I²C** commands are equivalent for cards of the same type, allowing a *plug and play*⁴ readout operation.

Since the **Mon-FPGA** is used for both types of crates and for different crate setups, it should be able to interface with every **I/O** module card (**T2I**, **OUT** or **GSS**) installed and connected to the crate backplane. The connection between the card and the **Mon-FPGA** is done through an **I²C** bus.

Some of the firmware requirements underneath this thesis work, as mentioned before in Section 3.1.2, are the monitoring of all input signals, analog or digital, using independent transmission paths, the monitoring of all output signals, by collecting the reaction of the system on the **OUT** channels, according to the **ILK-FPGA** interlock matrix decision, the ability to test individual channels of the cards with input signals commanded by the **DCS** and, subsequently, to transmit that data to a server.

4.5 Firmware Implementation

As convention for this firmware study, a module is defined as **VHDL** code that is executed to perform a certain function and is shown in the format of "module". A block is a functional piece of firmware whose function is executed by a single or a group of **VHDL** modules and it is represented as *block*.

Figure 4.1 illustrates the overall monitoring process of an **OUT-20** card, as example. This includes the *plug and play*⁴ operation, done by reading of the **EEPROM** chip of the **I/O** card connected through the backplane. This reading informs the **Mon-FPGA** about the card type and sub-type mounted in that specific crate slot. The monitoring consists of an order/response or receive/send packets mechanism. When the **Mon-FPGA** firmware is triggered by sequences of backend command requests, it triggers a chain of events starting with the processing of the command by the "FromLink" module. The commands come as a serial link from the backend. After processing, it is sent to the "Main_CTRL" module. This module orders the exchange of **I²C** scripts⁵ between the "I2Cslots" and the "ReadDataBlocks" modules. The former selects the appropriated **I²C** scripts for that card (only possible with the **EEPROM** information), stored in the **I2CDPM** memory, and the latter executes the readout of the card's monitoring data, such as the status of its output signals, by executing the selected **I²C** scripts. The readout data is then rearranged into response packets to be sent to the backend server. The response building is not represented in the illustration. The **Mon-FPGA** can also update the backend with the card's **EEPROM** information.

The following sections detail briefly on the receiving and sending of data packets firmware blocks, that allow the **Mon-FPGA** to update the server, while also studying the main block.

⁴The *plug and play* is understood as the plug of the module to be read and play as the readout procedure executed by the **Mon-FPGA**. The installation of the cards in the crate is enough to start the monitoring reading process.

⁵An **I²C** script is a sequence of **I²C** commands in the form of program scripts, fixed for each card type [19].

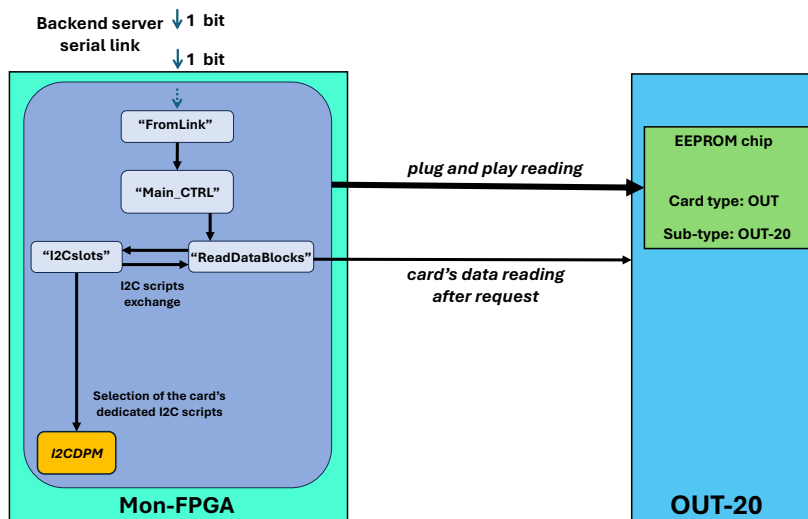


Figure 4.1: Illustration of the overall monitoring operation of an OUT-20 card. The **Mon-FPGA** uses the card's inherent information, stored in the **I/O** card's **EEPROM** chip, to select the appropriated **I²C** scripts, to then, read the interlock data from that card, when requested by a backend serial link. The blocks shown are studied in this section. The read data is then sent to the backend, which is not represented in the illustration.

The overall firmware is assembled by the top module, "monitor_top", exemplified in Appendix B.4, that, essentially, connects every port of the other modules, allowing their communication. The modules interconnections are achieved through the **component** concept explained in Section 4.3.

4.5.1 The "FromLink" module

The "FromLink" module is the first module directly connected to the outside, the server. It receives the commands coming from the backend server and stores them in a local memory. For that purpose, it comprises a sequence of 3 submodules, the "Phase_sel", the "decode_4b10b" and the "packet_decoder". The backend commands processing by these submodules is sequential and on this order.

The "Phase_sel" module is responsible for the deserialization and the optimal signal phase selection process. It receives the serial bit link from the backend server. The serial link is deserialised into a bit sequence, oversampled and synchronized. Oversampling of a signal is the process of taking an instance (a sample) of the same signal at different time instants in the same clock period. It allows to reduce noise and increase resolution of the sampled signal, since the **SNR** is proportional to $\sqrt{N}$, where N is the number of samples. So, by increasing N , the **SNR** improves⁶.

The following module is the "decode_4b10", that receives the previous processed deserialised link. This module decodes it according to a 4b10b encoding, which encodes a sequence of 4 bits into a group of 10 bits. The decoding process will execute the opposite. An encoding process ensures more layers of error detection and noise reduction in the transmitted signal, by introducing predefined bit patterns for parity checks. In this case, it also allows to generate a new smaller bit sequence from a larger and more noise-susceptible one. The Appendix B.1 details the encoding process, based in a 10-bit to 4-bit as said, with an additional control bit, K , that aids in distinguishing types of commands.

The decoded 4-bit commands are received by the "packet_decoder" module. It holds enough decoded commands to build 32-bit structured long words, designated as command packet, to be processed by the main firmware block. After building the command packet, it stores it into a memory slot designed as *cmdDPM*⁷. A detailed description of the command packet sequence is presented in Appendix B.2.

⁶A higher **SNR** provides a clearer signal, since the noise magnitude is smaller compared to the signal magnitude.

⁷DPM stands for Dual-Port Memory which is the type of memory implemented. It has two access ports, one for writing and another for reading data purposes.

4. THE HARDWARE OF THE MONITORING SYSTEM

4.5.2 The "Main_CTRL" module

The "Main_CTRL" is the central module of the [Mon-FPGA](#) firmware. It implements a [FSM](#) that, upon detecting an update of the command packet, stored in the *cmdDPM* memory by the "From_Link", launches the process requested by the command. Each process is associated to different [FSM](#) states. Figure 4.2 represents the diagram of the "Main_CTRL" state machine.

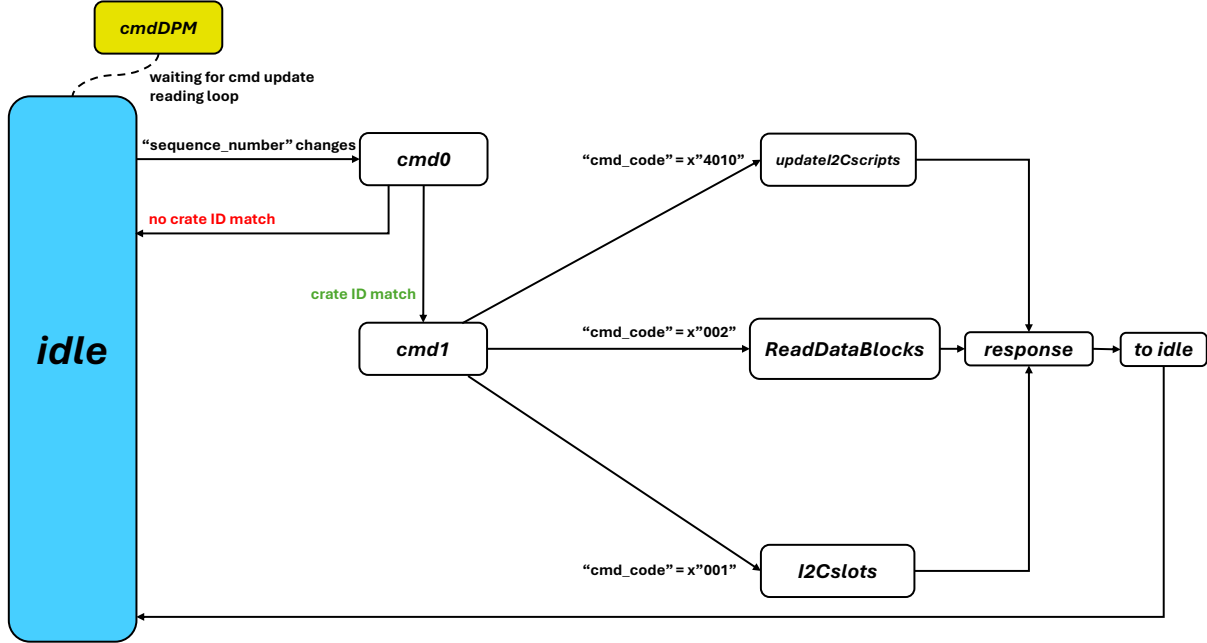


Figure 4.2: The "Main_CTRL" state machine diagram. The *idle* state waits a command update, stored in the local memory, *cmdDPM*. After detecting a change, it moves to state *cmd0*, where a crate ID number matching process occurs. If it matches correctly, it chooses the next state according to the "cmd_code" on the 3rd long word of the command packet (Figure B.1). The next states are associated with enabling of a specific process. These also build a response packet to be sent back to the backend (*response* state) and then move back to *idle*. Although they are not represented, there are several error detection variables which force the [FSM](#) to reset to *idle*.

The [FSM](#), in the *idle* state, monitors in a loop cycle the "sequence_number" byte of the command packet, in the memory address of the command header. The command packet is stored in the *cmdDPM* and is structured as it is presented in Figure B.1. When it detects a change in the "sequence number", it starts an initial state succession. In the *cmd0* state, it confirms the crate number of the command packet header (comparing it to the [Mon-FPGA](#) front-panel's switch - Section 3.1.2.2), as an additional error detection layer. Following the crate number matching, the machine transitions to the state *cmd1*, where it determines the next state according to the command specification in the "cmd_code" of the command packet sequence (Figure B.1). This state is also designated as *branch* state, since it branches the [FSM](#) to specific states, according to what the command orders. The state and process association with the "cmd_code" 16-bit sequence is presented in Table B.2. These types of states end by moving to the *response* state, in order to build a response packet to be sent to the backend and then return to *idle*, through the *to idle* state.

The *updateI2Cscripts* is based on a firmware requirement. It should be able to provide a mechanism to write [I²C](#) scripts from the backend server, later used to read the cards' data by the [Mon-FPGA](#) [19]. At the present moment, in this thesis development, the [I²C](#) scripts are stored directly in the local memory slot for that purpose, the *I2CDPM*, and are not configured from the backend, since this block is not developed yet.

The *I2Cslots* should generate and store the I^2C scripts to allow the reading of the individual crate slots and the *ReadDataBlocks* should execute their reading, by exchanging the correct I^2C scripts for the cards connected to the *Mon-FPGA* through the backplane. These states control specific modules with the same name⁸, as it is presented in the Table B.2. The controlling of the specific processes and blocks are executed with enable flags, that are output from the "Main_CTRL" module and used as inputs on the respective block modules, what leads to the launching their processes.

Initially, in order to cover every *Mon-FPGA* requirement, the "Main_CTRL" FSM had more intermediate states than the ones represented. However, upon development, the firmware was reduced without losing any functionality. The code still contains those obsolete elements, but they were excluded from the logic flow [72].

4.5.3 The "I2Cslots" and "ReadDataBlocks" modules

The "I2Cslots" module is composed by multiple submodules helping to generate I^2C scripts for each type of card. The I^2C scripts allow I^2C communication and, consequently, data exchange between the *Mon-FPGA* and every slot meant to be read; in the case of the *LISSY* crate, from slot 1 to slot 19⁹. It also contains multiplexer submodules in order to access and store the previously generated scripts in the *I2CDPM*. These submodules allow the scripts exchange with the "ReadDataBlocks" module.

The "ReadDataBlocks" module executes the main requirement of the *Mon-FPGA*. It executes the readout process of the *I/O* cards connected to the *Mon-FPGA*, through I^2C lines in the backplane. This block executes the card's readout, whereas the "I2Cslots" generates the scripts that allow the readout to occur. The scripts exchange is done through this module communication with the multiplexers of the "I2Cslots", that collect them from the *I2CDPM* and return them to execute the readout of the cards. The data collected from the readout is then used to build a 32-bit response packet, similar to the command packet.

4.5.4 The "ToLink" module

The *Mon-FPGA* firmware cycle ends with the "ToLink" module, the "FromLink" counterpart. It receives the response packets formatted as 32-bit long words. The module transforms them into 4-bit sequences, which are then encoded to 10-bit sequences and serialized into a link to be send to the backend server. Each singular step is executed through individual submodules, the "CU_packet_gen", the "encode_4b10b" and then the "Serdes10" submodules, sequentially.

4.6 The PYNQ board

The Monitoring System of the crate does not only include the *Mon-FPGA*, but also the server hardware where the gathered data is loaded by the card and later accessed for monitoring purposes. This function is going to be executed with the *EMP*, another *SoM* designed around a *FPGA*. It enables the communication between the *Mon-FPGA* and the *DCS*, by running a server where the crate data will be available when requested by the Wincc project, which sets up an user interface for the monitoring of the system. The *CERN* software platform for the servers is the *Open Platform Communications Unified Architecture (OPC UA)*. An *OPC UA* server provides a machine to machine communication protocol, developed by the OPC Foundation [55]. The data exchange functions of the *EMP OPC UA* server are meant to be used in several *HL-LHC* experiments, such as the *ITk* and the *HGTD*, with multiple boards [29].

⁸The naming was attributed that way specifically for this thesis as a way to simplify the subject. However, in the actual *VHDL* code, they are declared with other designations.

⁹Slot 1 corresponds to the *Mon-FPGA* dedicated slot, which reads the *LISSY* crate power supply status.

4. THE HARDWARE OF THE MONITORING SYSTEM

However, this hardware is still in the development phase, not having enough quantity for the development of the **Mon-FPGA** and Monitoring System. In the meantime, it was proposed as an alternative the development board PYNQ Z2 [14], from AMD/Xilinx. It has a Zynq XC7Z020 **SoC** chip [86], designed to support Python projects development. A **SoC** has more than one **IC** in one package. In this case, the Zynq has a dual-core **ARM** Cortex-A9 processor integrated with a programmable logic equivalent to an Artix-7 **FPGA**, meaning it has a **CPU** portion - Processing System - as well as **FPGA** capabilities - **Programmable Logic (PL)**. Basically, through a Python interface ran by the Processing System, it allows the configuration of the **PL**. The Processing System is able to run a Linux-based system bootable image, accessible through **SSH** connection to its terminal. The Linux **OS** version is burned into a SD Card, which is inserted in one of the PYNQ ports. The latest available image at the development of this thesis was used: a Linux Ubuntu 18.04. The development board already came with this Ubuntu version, but it was possible to choose a specific pre-built image from the manufacturer site [14], develop a dedicated image or use one from the public repositories of other projects. The **ITk Mon-FPGA** team carried the development of the PYNQ board as a substitute of the **EMP**. Its implementation for the local setup addressed in this thesis composes a significant part of the thesis development.

Apart from the **OPC UA** specification, the PYNQ board is also used to load the **Mon-FPGA** firmware into its **FPGA**, after being synthesized. It does that by using the **Open On-Chip Debugger (OpenOCD)** package. The objective of this package is to set up a tool for debugging, in-system programming and boundary-scan testing for embedded devices like the PYNQ board [6].

4.6.1 The implemented setup

Given the two functionalities for the PYNQ board, it was established a hardware configuration for both. Each one of the configurations can be connected at the same time, but not all their processes can run simultaneously. Section 4.6.2 will address this aspect.

The interface between the **Mon-FPGA** and the backend server hardware, being this the **EMP** or the PYNQ board, is executed with a CAT6A cable. To enable the communication between the backend hardware and other server clients, such as the **WinCC Open Architecture (WinCC OA)** project, the backend hardware needs to establish connection with the outside on a local network (through router). This connection is also executed through a CAT6A Ethernet cable. Since the PYNQ board only has one Ethernet port for the development of this project it was added an Ethernet peripheral port to set the **Mon-FPGA** connection (PMOD RJ45). The local network connection was established using the PYNQ original Ethernet port. This setup is designated as the *monitoring setup*.

The other hardware chain is called the *configuration setup*, which allows the reconfiguration of the **FPGA** of the card by the PYNQ board, through a USB blaster. On one side of the blaster, the connection to the PYNQ board is achieved through an USB port. On the other side, the connection to the **Mon-FPGA** is done with a flat **JTAG** cable. The USB blaster was a Waveshare USB Blaster [80]. For the reconfiguration of the **Mon-FPGA**, an access to the PYNQ board is still required, which is achieved through the same communication method with the local network (PMOD RJ45).

Figures 4.3 and 4.4 demonstrate both setups established concurrently. The former illustrates in a diagram the implemented setup and the data transfer directions. The latter, displays the physical setup represented above. It is worth mentioning that the physical setup (Figure 4.4) only demonstrates the green frame of Figure 4.3.

4.6.2 On-System Configuration and Application

The initial contact with the PYNQ board required a familiarization with shell scripting tools, such as bash, Linux system and communication protocols. The PYNQ board can be accessed in two meth-

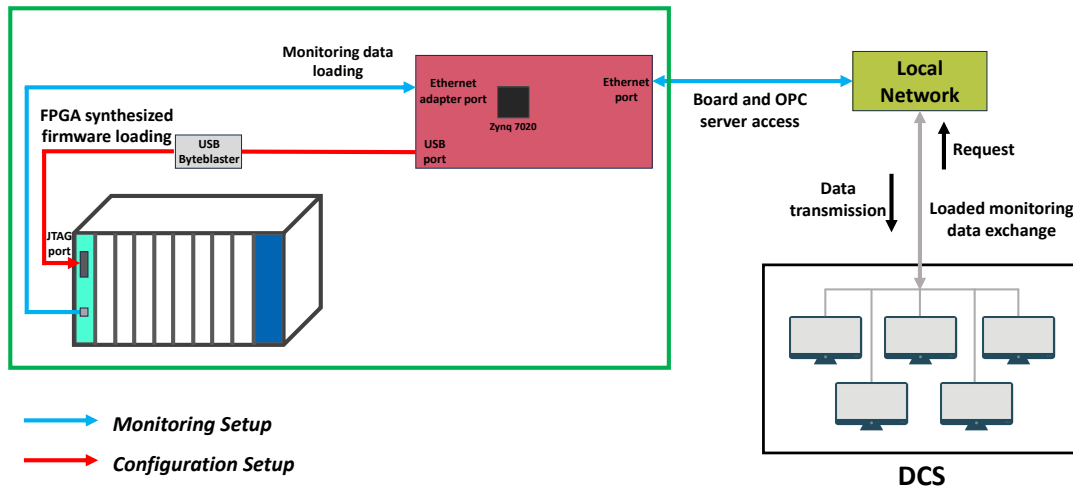


Figure 4.3: Diagram of the Monitoring System. Both setups are represented. In blue is the data exchange performed by the *monitoring setup*. The *Mon-FPGA* loads the monitoring data from the crate, in this case the *LISSY*, into the running server on the PYNQ board, through an Ethernet connection. Then, a connection from the backend hardware to the local network enables the transmission of this data to the *DCS* project, in a client request/server response protocol. The red path represents the *configuration setup*, comprising only the *JTAG* connection between the PYNQ board and the *Mon-FPGA*. A connection to the PYNQ board through the local network is also necessary. The green frame contains the setup shown in Figure 4.4, which encapsulates the conventionally designated Monitoring System's chain. While the system as a whole is the conjunction of this chain with the developed monitoring interface.

ods: by a direct connection with a computer, through an Ethernet cable, or through a connection to the local network of both systems, the PYNQ and the programming computer. The latter method, the local network approach, was selected, in order to emulate the *EMP* backend chain, to approach the final implementation setup and to allow the update of the system with the required tools. The former method does not support automatic updates and tool's downloads for the PYNQ, through internet.

The preliminary familiarization stage required a successful connection to the PYNQ board. Considering that the development of this thesis was conducted in the local laboratory at the FCUL, this demanded working within an institutional and highly supervised network. With this being said, it was necessary to request to the IT Department access the local network, for the PYNQ board and programming computer. This is achieved by providing the *MAC* address of both hardware pieces. The *MAC* address is a unique identifier of network interface control drivers or hardware, such as the typical computer or microcontroller network drivers. On the other hand, each time a network interface control hardware connects to a network, it is assigned an *IP* address, usually different in each connection, unless it was established a static *IP*. The PYNQ *MAC* address is 00:05:6B:03:A8:F4. For the access request it was also required to provide the active Ethernet outlet of the laboratory (B5.068). The access request is required in order to keep track of the connected devices of the network. Ethernet connection usually represents a higher risk for both the network and the device, since the device has access to certain network segments, that, otherwise, through wireless connection it would not have. This open a window that allows untrustworthy devices to access the network and devices using it, thereby facilitating unauthorized use and putting both systems at risk.

Afterwards, it was followed a standard procedure to access the PYNQ board. Its system did not provide a *GUI*. As already referred, the board can work under a Python framework. One of the methods of operation was to use the Jupyter notebook [61] as the interface to the board. This mainly provides Python code execution, accessing this type of files in its system. It also allows to open a bash terminal page in order to navigate in the board system. However, for that purpose, it was preferable to directly access the board through a dedicated PowerShell tool for bash terminal.

4. THE HARDWARE OF THE MONITORING SYSTEM

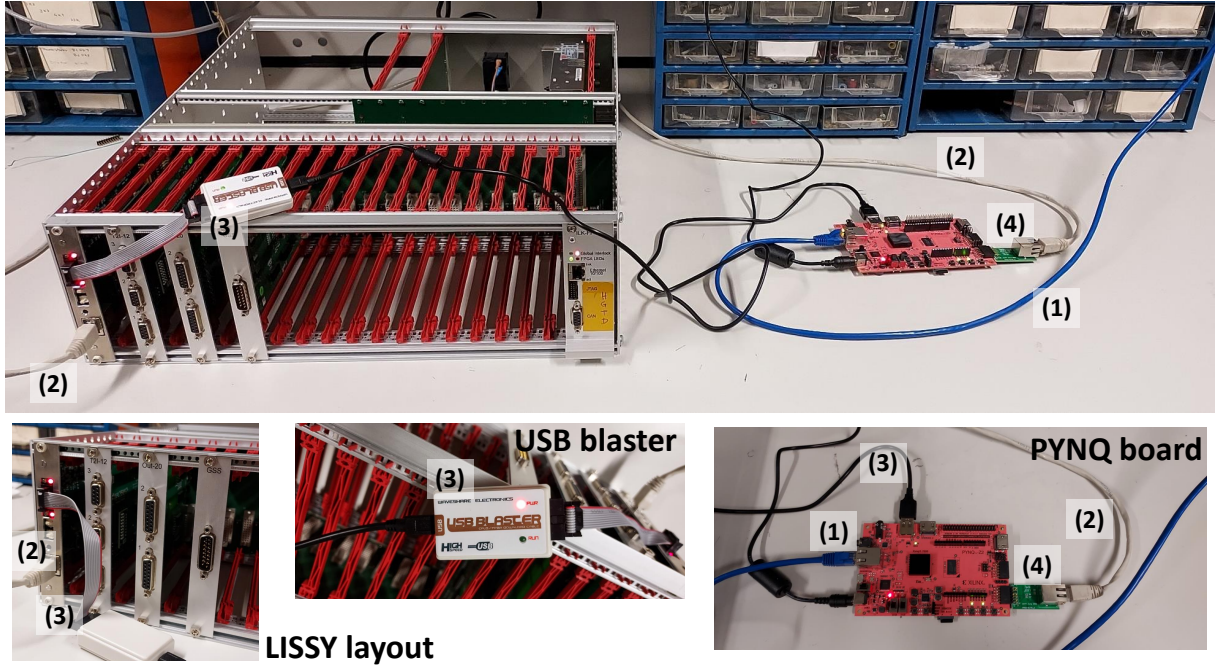


Figure 4.4: Workbench of the Monitoring System chain. Both setups are presented. At the figure's top is the overall setup. At the bottom are specific sections of the setup above. Bottom left is the **LISSY** layout used throughout this thesis work; from left to right: **Mon-FPGA** and its connections (Ethernet and **JTAG**), a T2I-12, an OUT-20 and one **GSS** cards. Bottom centre is the USB blaster used to reconfigure the **Mon-FPGA** and at the bottom right is the PYNQ board and its connections, to the **Mon-FPGA** and to the local network router. The hardware legend is at follows: (1) blue Ethernet cable: establishes the local network connection of the PYNQ board (top and bottom right); (2) white Ethernet cable: enables the connection between the PYNQ board and the **Mon-FPGA** (top twice, bottom left and right); (3) USB blaster: bridge of reconfiguration between the PYNQ board and the **Mon-FPGA**. The flat cable of the blaster connects the **JTAG** ports (top, bottom left and centre) and the other side of the blaster provides a mini-USB to a USB connection to the PYNQ board (top, bottom centre and right); (4) PMOD RJ45: peripheral Ethernet adapter, which allows the **Mon-FPGA**/PYNQ board data communication (top and bottom right). The cards included in the setup were: 1 **Mon-FPGA**, 1 T2I-12, 1 **GSS**, 1 OUT-20 and 1 **ILK-FPGA**.

This was the method implemented, given its reliability and efficiency, providing important skills and competences for future projects considering its applicability on the control of such systems. Nevertheless, both procedures required an **IP** connection to control the board. This was achieved by knowing the predefined credentials of the board from the setup guide platform [63]. For an easier first access to the board, it comes with an established static **IP** address protocol version 4 (IPv4): 192.168.2.99. This access method requires a terminal emulator software, in order to establish the connection through the PYNQ **IP** and to provide a bash terminal to work on its system. The **SSH** protocol is a typical approach for the purpose. It is based on client-server connection, where a **SSH** client connects to a **SSH** server or remote computer using its credentials. It is widely used for secure remote operations, such as the PYNQ board access.

The preferable choice for this tool was the PuTTY [62] software to emulate the system's terminal and the WinSCP [85] for file transfer between systems. The file transfer, which will be addressed shortly, is executed through a **SFTP** connection, which stands for file transfer using **SSH** protocol. In order to establish access with the board, the programming computer used to navigate the PYNQ system either needed an Ethernet connection on the local network with its IPv4 address modified to cover the range of the PYNQ board address (192.168.2.N, $N \in [0, 255]$) or a connection to that network Wi-Fi or its **VPN** connection. The IPv4 modification was required given the complication of establishing connection between two devices of different **IP** ranges. The programming computer **IP** was assigned automatically through **DHCP**¹⁰, unlike the PYNQ board static address. The IPv4 modification implied to change

¹⁰**DHCP** is a very common automatic **IP** address and network client configuration assignment in networks.

the subnet mask¹¹ to the same used by the PYNQ board, 255.255.255.0, and a preferred DNS server, 127.0.0.1.

The PYNQ board configuration for its two assignments required following dedicated instructions provided by the [Mon-FPGA ITk](#) team. The gitlab repository [41] contains all the necessary files to configure the PYNQ board. The first to be executed is the "setup_pynq.sh". It is saved inside the repository, so to run it on the PYNQ system it is required to either transfer it, via [SFTP](#) connection, or by cloning this repository in the PYNQ home directory, using the git command on its terminal (it is necessary to install the git package). The repository contains not only the files which configured the board to execute its functions, but also the firmware detailed before in Section 4.5. The "setup_pynq.sh", when executed, installs all the necessary packages, establishes the required system services and provides important files their execution permissions, to run them properly. This file is a shell type file and uses a bash script, command tool available in Linux systems to execute multiple commands efficiently. It could be executed using shell line commands.

The "setup_pynq.sh" also establishes the "itk_backend.bin" file, stored in the repository, as an [OS](#) service. It can be later found in the /lib/firmware/ PYNQ directory. A Linux service is a program, either controlled by a file or application, which runs in the background, executing predefined tasks. These files are defined with the service extensions (.service) and can usually be managed by the "systemctl" Linux command, such as start, restart, stop and status commands. The "itk_backend.bin" (backend firmware) is an important file to configure the PYNQ board, allowing it to reconfigure the [Mon-FPGA](#) and to run the [OPC UA](#) server. It is a synthesized [VHDL](#) firmware, developed in the Vivado [82] block design software for Xilinx [FPGA](#) products, to configure the [PL](#) segment of the PYNQ [SoC](#) system. Every time the board was restarted, it was necessary to load the backend firmware to ensure a proper operation. It defined a specific [LED](#) pattern to help identify if the firmware was properly loaded and echoes an "itk_backend.sh OK" message on the terminal on successful loading.

With the PYNQ backend firmware loaded, the board was set to reconfigure the [Mon-FPGA](#). This process involved the [OpenOCD](#) package installed by the "setup_pynq.sh" file. The [Mon-FPGA](#) firmware was already synthesized and its corresponding file (each [Mon-FPGA](#) version used a different version of the firmware) is already in the PYNQ system, by the "setup_pynq.sh". For the reconfiguration, it is included another bash script to aid in the loading of the firmware to the [Mon-FPGA](#). This script calls the [OpenOCD](#) to execute the [FPGA](#) interface through the [Mon-FPGA](#) and the USB Blaster. The synthesized firmware is an Altera Quartus [33] file type, dedicated to [JTAG](#) configuration, the [SVF](#). By contrast, the PYNQ board configuration uses a binary file (bin), as it is an on-board configuration and does not uses either Quartus nor [JTAG](#) interface. By running the dedicated bash script with the specific [SVF](#) firmware file, the firmware is loaded to the [Mon-FPGA](#). The USB Blaster has 2 [LEDs](#), red and green. The red one should always be turned on, as an indicator of the blaster operation, and both should switch when the loading of the [Mon-FPGA](#) firmware is being executed. The [Mon-FPGA](#) already carried a functional firmware loaded into its memory, which was set at the card's power-up. This application allowed a proper operation of the card at power-up. In the course of this thesis development, it was recommended an alternative firmware, which was not stored in the card memory. Thus, it was necessary to load the firmware every power-up cycle, using the above procedure.

With the [Mon-FPGA](#) configured, it was necessary to set up the PYNQ board to run the [OPC UA](#) server, in order to conclude the physical system configuration (Figure 4.4). This server protocol is based on a client/server model, fortified with exchange of credentials, and allows re-use of [OPC UA](#) code.

¹¹The subnet mask refers to a section of the network, allowing the data traffic within that subnet to be more efficient.

4. THE HARDWARE OF THE MONITORING SYSTEM

Given the high demand for this type of protocol in the CERN workplace and after successive collaborations, it was developed the Quasar OPC UA framework. This tool allows to easily integrate multiple OPC UA servers schemes within the CERN requirements, reducing the cost of their development and maintenance [64], as it was the case of the HIS server, specifically from the ITk. The Quasar framework includes the WinCC OA integration tool, the Cacophony, which supports the generation of the appropriated scripts for the project and will be later addressed in Section 5.2.3. Besides the brief description of both tools, Quasar and Cacophony, and their testing, neither of them were used to developed the respective products, as they were re-used from the ITk project.

The Quasar can be found in an open-source GitHub repository [66]. It aids in the generation of pre designed schemes to fit the CERN framework, by using Quasar dedicated scripts and git commands. The OPC UA server depends on a series of design files, including a XML configuration file, an object-oriented script that allows to specify the system being run in the server. From the pre defined scheme, it is possible to specify it to set up the used system, by the modification of the design files that composed it. This allows a fast server integration into the system in cause. The server creation form the GitHub repository will build executable files to run the server, in the created directory build/bin/. The ITk Mon-FPGA repository already has the specific server configuration directories and can be found at software/mon_opcua_server_bin_v3/ folder. The configuration file of their server was modified and adapted for this project's LISSY system, as it is conventionalized in Section 4.6.2.1.

Similarly to the "setup_pynq.sh", there is a bash file to configure the PYNQ board as the server hardware. The "setup_pynq_supervisor.sh" installs the necessary packages and establishes the monitoring OPC UA server as a controlled process. The main package is the *supervisor* [71] (providing a Linux service with the same name), which is able to manage the monitoring server through similar service line commands. The server can also be run manually in the executable configuration files folder, but the *supervisor* aids in the debug and maintenance of the server, since it was noticed that the server crashed after about 1 hour and 40 minutes of continuous operation. The *supervisor* ensured that it was properly restarted. Alternatively to the WinCC OA project, the UA Expert client [28] software was frequently used to verify the server status and objects. It incorporates a log¹² panel. This allowed to verify the operation of the server and management by the *supervisor* service, by accessing a cleaner interface than the Linux log files. For historic logs, the log files from the monitoring server stored in the PYNQ board were also accessed. With the configuration of the PYNQ board and server to receive the monitoring data of the LISSY crate, the monitoring project was able to request that data and display it to the DCS user. The HGTD dedicated repository contains direct instructions on how to set up the PYNQ board and the monitoring chain [12].

4.6.2.1 The server configuration file

The OPC UA server is configured through design files. This thesis addressed specifically the adaptation of one of them, the configuration XML file designated as "config.xml". It defines the objects being monitored and updated in the server. This section covers a defined set of naming and structure conventions for this file, targeting the HIS at this stage of development. This was essential due to requirements of the WinCC OA project, which also used this file to set up the monitoring objects. It was additionally applied by the ILK-FPGA to defined the user logic for the interlock matrix. Thus, the file conventions allow to standardize it for the system specifications and for easy future developments.

¹²Log is known as a record of events, in this cause, of the server data status.

By being a **XML** file, it is defined as a tree structure, having the root-variables with its child-variables, each one with its own attributes¹³. The Appendix C analyses the file and key **XML** commands. This section will summarize the conventions defined for it. For the **LISSY** crate, the root-variable is referring to the monitoring set of objects being controlled, designed by "MonitorController". For easier in-server debugging, each root and child-variables have an ID indicator. As a branch of the root, the subsequent system branch would be the crate being monitored; this will represent a more specific set of objects. The crate name format should be "CrateX", where "X" is the crate ID number. The **HIS** is set to have at least 4 **LISSY** crates and this naming convention allows to, in the same configuration file, clearly and numerically specify multiple crate layouts to be monitored by different **Mon-FPGAs** on a same server. Inside the crate variable, every single card installed, meant to be monitored, should be defined individually as child-variables. The **Mon-FPGA**, referred to as "MonFPGA", in addition to its dedicated slot number 1, should only have one specific child-variables - the "alive" variable - related to the status of the **Mon-FPGA**/PYNQ readout cycles. This variable results from a Quasar complementary design file, designated as "CalculatedVariable", a C++ file. It comprises multiple functions to be invoked on the main configuration file. For this variable, the invoked function ("cumulativeDifferenceWithOverflow") calculates the difference between two **Mon-FPGA** "heartbeats". The "heartbeat" is the number of data update cycles that are executed within each server update cycle¹⁴. The function then applies the difference between successive "heartbeats", helping to verify if the current "heartbeat" is within an acceptable range¹⁵. For the system implemented in this thesis, the **ILK-FPGA** variable does not hold any specific attributes other than its name, "ILockFPGA".

Each **I/O** card should include attributes specifying its name, slot number and serial number, with its channels and signal type represented as child-variables. The Table 4.1 resumes the key conventions applied to this type of cards.

The name attribute of the **T2I** cards should follow the "T2I_SX" format, with "X" representing the slot number of the card. Additionally, it should have fixed value variables, "Ca1_A", "Ca1_B" and "Ca1_C", corresponding to $\frac{1}{298.15} - \frac{1}{3435} \times \ln(10000)$, $\frac{1}{3435}$ and 0, respectively, which are then used for the temperature calculations of the **NTCs**. This estimation is executed by invoking, once again, a function of the "CalculatedVariable" file, called "steinhartHart". This function is shown in Code C.3 and detailed in Appendix C. It basically employs the Steinhart-Hart equation, which models the behaviour of the resistance according to the temperature of semiconductors, like the **NTCs**. It can be defined by:

$$\frac{1}{T} = A + B \ln(R) + C \ln^3(R) \quad (4.1)$$

where T (K) represents the temperature of the **NTC**, R (Ω) its resistance at that temperature and A , B and C are coefficients specific of the semiconductor. B is given by $\frac{1}{\beta}$ (1/K)¹⁶. The **HGTD** projected **NTCs** are the Semitec 103KT1608T-1P [45], which at 25 °C (298.15 K) has a 10 k Ω resistance and a β of 3435 K. The A coefficient is obtained directly from the Steinhart-Hart equation at 25 °C. The C coefficient is always considered as 0. So, the "Ca1_B", "Ca1_A" and "Ca1_C" variables clearly correspond to these coefficients, respectively, used as the function's arguments.

¹³Variables' characteristics, such as their names or values. They are designed to contain data of that characteristic

¹⁴There are two distinct cycles: the **data update cycle**, when the **Mon-FPGA** stores the read data of the crate into the PYNQ's memory, after the board requests it to; and the **server update cycle**, when the **OPC UA** queries this stored data, updating the values on the server.

¹⁵This will be properly addressed in Section 5.2.4.1, as it was used as an alarm for the **DCS** user, checking on the **Mon-FPGA** behaviour.

¹⁶ β (K) is the often called as B value, which expresses the shape of the R/T curve of the **NTC**. However, to avoid misinterpretation between the B coefficient of the equation and the B value of the **NTC**, the latter is designed as β .

4. THE HARDWARE OF THE MONITORING SYSTEM

So, the voltage read at each **T2I** channel and converted by the card's **ADCs** should be applied to estimate the sensor resistance, R , according to the circuit equation analysed in Section 6.1.2, and then used in equation 4.1 to obtain its temperature.

In the **T2I** variable group, all used input channels should be classified among three sets of child-variables. One is the temperature of the corresponding reading, in the format of "CXX_temp" with "XX" being the number of that channel in padded numbering¹⁷ with two digits. Their value attribute uses the "steinhartHart" function referred before to estimated their temperature. Besides the input channels, each **T2I** card has three on-board variables transmitted from a specific **ADC**. From these three on-board variables, two of them represent the thresholds' reference temperatures¹⁸ ("hi_temp" and "err_temp") and one for the **T2I** on-board temperature sensor ("board_temp"). The other two types of **T2I** child-variables result from the **Mon-FPGA** requirement of also monitoring the binary signals of those channels. These values are obtained by using the "getBit" function from "CalculatedVariable" file. This function's body is analysed in Appendix C. One of the values sets the "ERROR" bit signals¹⁹ (high temperatures, above 40 °C), specifically in the format of "CXX_hi" and the other sets the "Broken Cable" bit signals (open channel), in the format of "CXX_err".

The **OUT** cards have the name format of "DigitalIO_SX", with "X" being the slot number and each output channel is specified as "CXX_out" ("XX" is the corresponding channel number) and it is associated to the bit value of the output, using "getBit" once more. The **GSS** card is designated as "GSS", with no slot specified in the name. Since there is only 1 **GSS** card per crate, there was no desire to specify in the name designation which slot it was installed. Its 10 inputs get the name "IN_XX" and its 3 outputs "OUT_XX" with the corresponding interlock bit values, applying the "getBit" function.

Table 4.1: Naming conventions of each type of channel signal for every **I/O** module of the **LISSY** crate, for the **OPC UA** configuration file. The **T2I** has 3 different type of variables: 1 for the sensor temperature and 2 for the interlock bit values of both interlock type of signals. The **GSS** card has 2 sets of variable types: one for its 10 inputs and other for its 3 output channels. "X" refers to the card slot and "XX" to the channel number.

Card type	Card variable name format	Channel name format	Type of value
T2I	"T2I_SX"	"CXX_temp"	NTC temperature
		"CXX_hi"	"ERROR" bit
		"CXX_err"	"Broken Cable" bit
OUT	"DigitalIO_SX"	"CXX_out"	Output channel bit
GSS	"GSS"	"IN_XX"	Input channel bit
		"OUT_XX"	Output channel bit

4.6.2.2 System integration

The **PYNQ** development included its integration in the local network and the setup configurations referred in the previous sections. Given that the local network is part of an institutional network, it is constantly updated in maintenance cycles. This affects the data collecting process from the server, since, mostly on weekends, the network suffered multiple restart and clean up sessions affecting the remote access to the board, using the institution **VPN** and its **IP** address. This also affects the server which runs as a background process. To maintain the server/client communication, the board needs a stable connection to the local network. By cutting the connection for these security cycles, the access to the server by a remote user, like the monitoring project or an **OPC UA** client software (like **UA Expert**) is

¹⁷Padded numbering is used for numbering with specific number of digits, filling the unused digits with 0, such as 01 or 02.

¹⁸Each channel has two **OP-AMPs**, each of them with a reference temperature; one for the **upper limit** and other for the lower limit of normal operation for the sensors. Section 3.1.2.3 and 6.1 detail and explore this matter.

¹⁹Bit "0" means "OK" state and "1" means "INTERLOCK" state.

disabled. This represented a **major inconvenience** mainly during long data collecting and server status support periods, in order to check its proper operation. This was necessary as the server is to run for an indefinite period of time, continuously gathering the crate data. It also affected the development of the monitoring interface, which, by being a server client, was constantly requesting access and being denied.

Another major issue observed during the development of this project was a **frequent PYNQ crash**. This persistently happened at two specific occasions: (i) when working on the configurations of both PYNQ setups; (ii) and when restarting the board. The former occasion was observed when running the "itk_backend" service, which occasionally crashed the system. The latter occurred because any attempt to restart the board - whether due to this system crash or the maintenance cycles of the local network²⁰ - resulted in the loss of access to the board.

These crashes were given as permanent, forcing the writing of a new Linux image in a SD card and the restart of the whole PYNQ board configuration process from the beginning. After intensive tests, one of the assumptions of the source of this problem was an overlap of the configuration scripts executions. This is due to the fact that:

- It occasionally happened when running the "itk_backend", as already stated;
- The "setup_pynq.sh" and the "setup_pynq_supervisor.sh", both enable their main Linux services - the "itk_backend" and the *supervisor*, respectively - to execute at the board boot.

This means that, every time that the board is restarted (which is when it was observed such event), it automatically starts both services. This is configured using the Linux "systemctl" commands and the "enable" flag. To test this hypothesis, both services were run individually and sequentially when the PYNQ was operating normally. This testing procedure verified that by running the *supervisor* first and then the "itk_backend", the PYNQ crashed as expected, which explains the occasions on which the crash regularly happened. This specific sequence occurrence is due to the fact that the *supervisor* and the monitoring server are constant processes running in the background, while the "itk_backend" is a momentary loading process. Thus, the reverse procedure did not caused any problem and it was established as the proper sequence of actions (first the "itk_backend" and then the *supervisor* service).

Condensing, the backend firmware could not be loaded when the *supervisor* was running in the background, which means that both services were considered unusable at the same time. By having both services disabled, it was then possible to restart the PYNQ board without major issues, since the services were completely under control by the developer. The disable at boot process was easily achieved using the "disable" flag at the service command line.

²⁰To re-establish the connection it was necessary to restart the board.

5 The interface of the Monitoring System

Alongside with the [Mon-FPGA](#), the Monitoring System's interface that complements it is as necessary as the module to properly monitorize and control the running systems of the [HGTD](#). The goal of this thesis is to, therefore, implement not only the [Mon-FPGA](#), but also to develop a visual set of panels to establish an initial stage [DCS](#) interface to the [HIS](#), focusing on the Demonstrator PEB, addressed in Chapter 7. This chapter will describe the development of the Monitoring System's interface, created with the [WinCC OA](#) software and its communication with the running server, involving a process of loading and requesting of the local data. The interface development required the establishment of the server connection to both the system and to the variables to be monitored and displayed, a web access and a database configuration aimed to a future implementation of the interface.

5.1 WinCC OA architecture

[Supervisory Control and Data Acquisition \(SCADA\)](#) systems are broadly used in industrial process automation systems¹, to build [Human-Machine Interfaces \(HMIs\)](#) and allow communication and control between users and the machine and its processes, by acquiring the data produced from them and using it for supervision. [CERN](#) and the [ATLAS](#) experiment rely widely on this type of systems to remotely monitor and control, if needed, such infrastructures, through the data of actuators and sensors, as it is the case of the [HGTD NTCs](#). The [SCADA](#) framework software used in [CERN](#) is the [WinCC Open Architecture](#) from Siemens/ETM [68], which is one of the multiple WinCC versions. The Open Architecture version is dedicated to large-scale and customized industrial systems.

A [WinCC OA](#) project is a development environment which holds the [HMIs](#) designed interfaces, as well as their configurations and designs. The project needs to be constantly running to enable the access to the interfaces. It is a modularity built system, since each module handles specific project functionalities, which are controlled by managers: modules are specific to the project, whereas managers are WinCC software processes. The number and type of managers can differ among different projects. In some cases, a project can have multiple managers of the same type for concurrent process control. The managers can be distributed in 5 groups regarding its main functions: Communication and Archiving, Processing and Control, Visualisation and Operation, Subproject Connection and Drivers.

Starting from the field² level, the most specific group of managers are the Drivers. This group comprises the Driver modules. *The driver reads current states, measured values or counter values from the field and passes commands and set values to the subordinate controls* [9]. Basically, it handles the collected data from the system's machines and communicates it to the project in order to be processed, also enabling their control through the project.

Then, this data is communicated to the Communication and Archiving type of managers. This group is the processing centre of the project and its main manager is the "Event Manager". This manager deals with all the other managers and their communication. The group also includes archiving data processes and event trigger alarms [9].

¹Process automation systems are systems that control their processes automatically, like most of the industry nowadays. It does not need the constant human intervention, speeding the processes and reducing their costs.

²"field" is understood as the communication level with the system's [PLCs](#), which are, essentially, computers dedicated to the [I/O](#) connections with the sensors and actuators.

5. THE INTERFACE OF THE MONITORING SYSTEM

The Processing and Control group comprises modules which implement algorithms to process the data retrieved by the Driver modules. The Visualisation and Operation incorporates the user interfaces of the project, operating not only for visualization, but also for editing and for user interaction with the systems under control. At last, more typical of large WinCC projects, there is the Subproject Connection. WinCC OA can be implemented as a network of different subprojects, each customized to specific systems, controlling large infrastructures and operating as a unified operational system. The modules and managers comprised into this group are used to set up the communication between WinCC subprojects and to allow the link between dedicated systems [9]. The managers' names will be specifically addressed as "manager name".

The WinCC OA allows to easily scale the project to extend and cover all the necessary machinery, user operation and system's requirements, by enabling individual manager operation (excluding the fundamental ones, like the "Event Manager"). With this, new managers can be easily added, started and stopped in the middle of the project development. It does not depend on the user computer OS and the number of users in concurrent work, this way allowing different managers to operate in different and dedicated computers, without causing conflicts in the project's operation.

As the main manager name implies, a WinCC project is (generally) event orientated, meaning that it only reacts when an event is triggered, such as a value change. Otherwise, it does not communicate or process data, by staying in steady operation, saving resources by processing less data.

In a WinCC project, the values from the sensors and actuators, which are every logic state and every measured value within the system, such as the NTCs temperature or interlock binary signals, are represented as variables of the process, designated as "datapoints" (DPs). The source of the data gathered from the field level is combined as a "device" in WinCC. Each DP is characterized by a tree structure, which includes the value of the DP, specific behaviours and their respective child-variables. At last, the DPs can also be configured with values ranges for alarms, conversion processes of their values and so on. The DPs are also defined by the "datapoint type" (DPT), which configures base parameters for every DP covered by this type of device. A single device can allocate multiple DPTs, which can then have multiple DPs. Both the DP and the DPT need to be created by the project's developer [26].

The WinCC systems developed under CERN's responsibility run inside an internal network and are not accessible without proper credentials.

5.2 The monitoring project

5.2.1 ITk project migration

The WinCC OA is used in a variety of automation systems of both the ITk and the HGTD detectors, including their respective Interlock Systems. The supervisor project, as expected, requests the update of the specified "datapoints" (DPs) values to the OPC UA running on the server hardware (the EMP or the PYNQ board), using the server credentials configured on the project. Figure 4.3 illustrates this monitoring chain and data exchange between the project and the Mon-FPGA, which collects the data from the field level. Basically, the data loaded into the server by the Mon-FPGA updates the values of the project's DPs. These are associated to each server variable with value. Considering this, through various interactions with the involving teams, it was possible to reuse a large portion of projects already developed and tested for the ITk detector. Since at the time the HIS was at an early development stage, the ITk SCADA system had much more functionality than the one needed for the current phase of the HIS. Thus, it was mandatory to, once more, adapt the project to the current requirements of the HIS. It is worth mentioning that the HIS requirements were focused on the Demonstrator PEB implementation,

detailed in Chapter 7.

The WinCC OA version used for the project development was the 3.14. This is an important information as it may lead to incompatibilities in certain parts of the project. The specific instructions for the HIS Demonstrator WinCC project can be found in the HGTD dedicated repository [11]. Since the WinCC OA is a licenced software, CERN already supplies it for authorized developers. So, it was necessary to request a license for continuous work, because the unlicensed version (also only available by requesting the download files) forces the project to stop each 30 mins of operation. One of the requirements for the provided license was that the software had to run on a Linux CentOS version 7 system. At the time and after trying multiple alternatives, it was necessary to use a computer with a dedicated CentOS7 system.

The ITk project [13] was part of a group of WinCC projects, working as subprojects of a larger interlinked system. One of the subprojects was dedicated to their Interlock System and this was the only project necessary to fulfil the HGTD project requirements at the time. The detachment of their project's dependencies with the rest of the system represented a significant portion of both the project migration and the thesis development.

To start with the project development, it was necessary to create a new and empty distributed project. This is achieved with user interface tools, such as the Project Administrator, which enables the control and administration of the projects. It allows to access the WinCC projects and to create new ones, either empty or by the import of an existing one. Then there is the Console tool, dedicated to only one project. This displays all the project's managers and allows their control, such as starting and stopping, individually and when preferred. It also allows to configure the managers with certain specifications, such as ports. Some managers communicate through dedicated ports. One manager can only have one port and it cannot be reused by another running manager. As an example, if a project has multiple user's editor managers for different systems, each one should have a dedicated port. Other managers, like the ones from the Processing and Control group, can be configured with dedicated scripts that are executed as part of their functionality. The usual debugging tool is the Log Viewer, which displays the background processes supporting the project operation, such as the managers initialization, communication processes and so on. The "User Interface Manager", from the Visualisation and Operation manager's group, can be used to control a variety of interface modules, like GEDI, the user graphical editor tool. It is an interface for the project's developers³ and most of the project's development is executed with it. These interfaces usually remain hidden from the end user⁴, who has dedicated modules.

As was noted through the project development, due to certain dependencies inside the project it was necessary to create the project with a specific name and system number, to avoid causing incompatibility between the project configuration and the original ITk scripts. These two parameters serve as unique project identifiers for connection between the project's distributed WinCC systems. Both names should follow the ATLAS DCS Integration Guidelines [15] (Section 3.4) naming convention. So it should start with "ATL" and then 3 capital letters of the subsystem: for the HGTD it would be "HGT". The name ends up as "ATLHGTLISSY". Since the system name is used in the "address name"⁵ of the DPs, which is an important detail for data communication, the original scripts required the modification to carry the new name. It is also possible to create credentials for the project's access at this stage.

The communication and data exchange between the server and the WinCC project are initially established through the introduction of the server specifications to the project, configuring it as a server client.

³There were two students working in parallel in this project: myself, Alexandre Parreira, and a fellow, Rui Vieira, on the ILK-FPGA. These will be referred as "local HIS developers".

⁴"end user" is who ultimately uses the product as intended. The product, in this case the project, is dedicated to this user.

⁵In this thesis, "address name" refers to the name of the DP used as an identifier and location address inside the WinCC system. The term "address" is not exclusively used, as in WinCC it is still referred to as the DP name.

5. THE INTERFACE OF THE MONITORING SYSTEM

Then, each [DP](#) also requires its own configuration. The client configuration can be executed manually through various steps. However, it was used an [ITk](#)-developed [ASCII](#) script to automatically carry that task, designated as "dpList". [ASCII](#) is the name of a character encoding convention, but here refers to a script. The steps of configuring the project as an OPC client include the adding of a new manager, a specific [OPC UA](#) manager called "WCCOAopcua", dedicated to projects that connect to this type of server, by requesting its data. This manager should be created with a driver number specification, that would redirect the manager to control the correct client configuration with that number. For the [HIS](#), the project's client driver number was 2, so the manager specification should be in the format of "-num 2". With this accomplished, the project should be internally configured, through a GEDI tool in the System Management tab, then in the Driver OPC and OPC UA client tab. This will launch a panel to configure the project as an [OPC UA](#) client. In this panel, the client driver number should be specified, as the one introduced before. The manager will, in this way, know that it enables the server/client communication with this configuration. The server endpoint [URL](#)⁶ should also be introduced in this panel, to allow the project, as the client, to locate the server on the local network. This is always necessary when trying to connect to the server, regardless of the used method, like through the UA Expert software. Another necessary specification is the server subscription configuration and the adding of a new internal [DP](#)⁷ for the server.

The [ASCII](#) script already executes all the aforementioned steps, excluding the appending of an OPC manager. The script is exclusively imported by a GEDI's tool called ASCII Manager, which takes the script and executes every line, including the specifications indicated above. The usage of this script for the [HIS](#) required the script adaptation for the local server endpoint [URL](#). Only in this way, the project is configured as an OPC client specifically for the [HIS](#) local server.

5.2.2 The JCOP framework

After the project creation, it was necessary to use another [CERN](#) developed framework. The framework is called [Joint Controls Project \(JCOP\)](#). It was a development from a collaboration between the [LHC](#) and the Industrial Controls Systems Group in the Beams Departments, in order to build control systems for their shared experiments. This standardizes their development, provides long-term maintenance and support for all developers and members and avoids incompatibilities between supervisor and control systems of different departments. The [JCOP](#) framework administers a set of components and tools to build these types of control systems for the [CERN](#) general experiments. It is specifically applied to [WinCC OA](#) [38]. Each [JCOP component should be installed through its dedicated installation tool](#). The tool's and components' versions depend on the [WinCC OA](#) system version. Each WinCC project is based within a system folder which includes all its configurations and developed interfaces.

The [JCOP installation tool](#) is compacted in a zipped file, which mirrors a basic WinCC project folder, allowing to unzip it under the project directory to automatically integrate it into the project, by adding a new tab into the GEDI's interface. The tab enables a new panel for the installation of each component. A panel in this context is an interactive graphical interface. It can be originally from the software, like a specific tool from GEDI's interface, or developed for the project system, such as the monitoring panel of the [LISSY](#) crate where its status is displayed to the [DCS](#) user. The developed panels can be accessed in the panels/ directory of the project. The [JCOP components](#) should be in a temporary folder and selected in the installation's panel. They can be downloaded from the [JCOP](#) web page [37] or from dedicated [JCOP](#) repositories. [CERN](#)'s dedicated [OS](#) also has folders with some [JCOP](#) components

⁶Location address for the server on the local network.

⁷WinCC internal "datapoints" are used by the project itself, without involving external sources, like the field level sensors and actuators.

for installation on those machine's projects. The components can be grouped into 3 main types: the Core, applied to fundamental and reusable functionality shared between other type of components; the Tools, for handling, displaying and storing data; and the Devices, to control common hardware devices of the experiments.

The components that are needed for the [HIS](#) project are the "fwCore", which provides basic functions to be used by the other components. The "fwViewer", "fwXML", "fwTrending" and "fwDeviceEditor-Navigator" components were not strictly required for the development of the current WinCC project, but rather are there as a standard practice for future project implementations. They serve to display and navigate a bi-dimensional scene, to use characteristic [XML](#) functions, to configure predefined plots for the data trending history and to instantiate and interact with [JCOP](#) devices, respectively [37]. Then there is the "fwAccessControl", which is meant for end user access to the monitoring panels. It needs special user configuration from [CERN](#) dedicated departments and it was not possible to accomplish it for this thesis. For the local project developments (unlike the final [CERN](#) remote machine's project for the Demonstrator implementation) it was possible to recreate an end user interface access using an alternative tool. This implementation is discussed in Section 5.2.4.2.

The "fwAtlasinstallation" and "fwAtlas", as the names imply, are components dedicated to [ATLAS](#)'s control systems. They work together to standardize certain elements of the WinCC projects, such as the colours of specific alarms and the removing of unused WinCC [DPs](#). And finally, the components focused on the [DPs](#) database connection are "fwNextGenArchiver", "NextGenArchiverOracle" (if the database is from Oracle) and "fwNgaUpgradeTool". The first supports database and storage technologies for both existing [CERN](#) database backends, the InfluxDB and Oracle. The latter is the recommended database used in [DCS](#) systems. Consequently, the second one is dedicated to the [DP](#) archiving on a Oracle database, which provides the configuration file entries, datapoints and documentation on the component's installation [54]. The third one simplifies the upgrade and integration of older WinCC and achieved versions into newer ones. The database configuration will be addressed in Section 5.2.4.3.

Some components, after being installed, automatically add dedicated managers for their proper operation, such as the "fwDeviceEditorNavigator" which adds an "User Interface Manager" and the "fwNextGenArchiver" that adds a specific Archiving type of manager, the "WCCOAnextgenarch".

5.2.3 The Quasar's Cacophony tool

As stated in Section 5.2.1, in addition to the project configuration as the server client, each [DP](#) required individual configuration to enable their data exchange with each variable of the server.

The [HIS](#)'s Monitoring System covers the [Mon-FPGA](#), which loads the crate data into the [OPC UA](#) server, and the [WinCC OA](#) project, which requests that loaded data, displaying it in a built interface for the [DCS](#) user. This is a common scheme for supervision of most of [CERN](#)'s developed systems. So, due to this large use of [WinCC OA](#) at [CERN](#) and to [SCADA](#) system's integration into [OPC](#) server frameworks, there was the need to develop a suitable tool for easier project integration within the [CERN](#) workplace. The framework is, once more, provided by a Quasar tool, developed by [CERN](#), called Cacophony [65]. After Quasar is executed and the [OPC UA](#) server is created⁸, with all the baseline scripts and files needed to run it, the Cacophony repository should be cloned and used with Quasar dedicated git commands to create a framework for WinCC projects, helping in its development. It creates a new directory for WinCC project configuration under the [OPC UA](#) server folder scheme.

From Section 4.6.2, it is known that the [OPC UA](#) server is configured through its dedicated design files, with the main one being the configuration [XML](#) file, designated as "config.xml". This file specifies

⁸As it is described in Section 4.6.2, [CERN](#)'s [OPC UA](#) servers are created by Quasar.

5. THE INTERFACE OF THE MONITORING SYSTEM

every variable to be loaded into the server by the **Mon-FPGA**. Therefore, each variable detailed in this file corresponds to the already described "datapoints" (**DPs**) in WinCC. As can be verified in Appendix C, the number of variables can easily grow in the development of such complex systems, like the Interlock System. With this being said, the Cacophony tool creates Ctrl⁹ scripts to aid in the creation and server connection of these **DPs**¹⁰, speeding the project's development. The scripts referred before are called "**createDpts.ctl**" and "**configParser.ctl**". The process of creating a **DP** is strictly sequential, by first having to create the **DPT** and then its associated **DPs**, with all of the necessary specifications. The latter can only be created if the former already exists. This justifies the need for two distinct scripts, one for the creation of the **DPTs** and the other for the creation of all the **DPs** with their values.

For instance, see the **NTC** temperatures transmitted through the T2I-12 channels. The **DPT** would be the type of the card, which is **T2I**. Associated to this **DPT** would be a set of **DPs** corresponding to that specific T2I-12 card. Clearly, the temperature of the **NTCs** connected to each channel of that T2I-12 card, would represent a sub-set of **DPs**. But this card is also associated to each channel binary interlock signals, which would correspond to another sub-set of **DPs** for that card of that **DPT**.

As such, the "createDpts.ctl" is designed for the **DPTs** instantiation and the "configParser.ctl" for the **DPs** creation under each dedicated type. The former builds the **DPTs** in a pre-defined structure. The latter is the bridge between the server configuration file and the respective **DPs**. It **takes on the "config.xml" file** and **creates the **DPs**** according to the variables described in it.

The generation process of these scripts is based on a python script from the Cacophony repository [65], which uses another **XML** server file, called "Design.xml", that results from the Quasar server creation⁸. This file is similar to the "config.xml" file, but it comprises generic information for each type of variable to be converted into WinCC **DPT**, such as the data type and the addressSpaceWrite of each card. Basically, it works as a basic frame for each different card. By using the attributes from "Design.xml", the Cacophony creates structured schemes in the scripts for those specific **DPTs** and the **DPs** that should come with it. The generation of such scripts can even be customized using optional flags of the Quasar git commands. The most used are the **DPT** prefix, to help identify, for example, the detector to which such "devices" belong, and the server name designation, which should correspond to the name specified in the previous server creation under Quasar (**ITk** defined it as "mon_opcua_server" and to facilitate their project's migration it remained the same name). These flags are then incorporated in the scripts and **DP** creation, allowing an easy and effective server connection to them afterwards.

The "createDpts.ctl" contains a set of functions to create **each card type variable**, from the "config.xml", **as a **DPT****. It creates a **DPT** for the **Mon-FPGA**, **ILK-FPGA**, **T2I**, **GSS** and **OUT** cards and these **DPTs** group all the cards of the same type. This is where the previous flag prefix is introduced, by concatenating the **DPT** with it (for instance, the **Mon-FPGA** from the **HIS** could be referred to as "HGT-DMonFPGA", if the prefix is "HGTD" and the "MonFPGA" is the name specified in the "Design.xml" file). These functions not only create the **DPT**, but also create the "datapoint element" (**DPE**) for every type. The **DPEs** are the declared **DPs**, basically, with empty values. Then they need to be instantiated through the **DP** creation. With the attribution of **DPEs**, the WinCC system will recognized their existence, although they add no value to it. Every **DP** needs a **DPE** to be represented within the system and that is when it actually represents a value for the WinCC system. So, when the "createDpts.ctl" is first executed, it **creates the **DPTs**** and a sort of **scheme for the available **DPs**, the **DPEs****. Appendix D.1 exemplifies the use of this script when applied to create the **DPT** of the **T2I** card and the **DPEs** of its

⁹Ctrl is the scripting language of the WinCC software, based on ANSI-C.

¹⁰Besides the project configuration as the server client, each **DP** needs to be configured to pass through the communication channels and exchange values with the server variables.

channels for temperature readings. It also shows that for every **T2I** and OUT card sub-type, this script creates **DPEs** for every possible number of channels that these card types can hold (16 and 32 channels, respectively). However, for sub-types as the T2I-12, OUT-20 and OUT-24 (which do not have the maximum number of channels), only 12, 20 and 24 channels actually establish a connection with the server, respectively. This connection configuration is executed by the other script.

The "configParser.ctl" is built around a sequence of function executions to **create each individual DP**, sequentially according to the **XML** hierarchy, and **establish a connection** with the equivalent **OPC UA** server variables. Parser is a designation for syntax analysis and it is aimed to the "config.xml" file, which is analysed and used by this code to create the necessary and equivalent **DPEs** for the server connection. The **DPEs** creation process follows the "config.xml" variable hierarchy presented in Section 4.6.2.1 (Figure 5.1). There is a central function called "parseConfig", that accepts the "config.xml" directory as its input. It first strips the configuration file to analyse the first variable on the hierarchy, the "MonitorController" (the highest **DPT** in the "config.xml" hierarchy). Then, upon calling its dedicated configuration function, it creates its own **DPEs**, under the parsing content. This process is repeated until the leaf-variables¹¹ of each branch are found. The "MonitorController" invokes the "Crate" (the second highest **DPT** in the hierarchy) dedicated function, that then calls each card function for the same purpose. At each step of **DP** creation, the parsing of the "config.xml" is used to handle the correct creation of the **DPEs** of the cards being monitored. The dedicated card type functions follow the same name format: "configure" + **DPT** name, like "configureMonitorController" or "configureCrate". Figure 5.1 illustrates the sequence of events and the functions used in the **DP** creation process.

Figure 5.2 represents both Quasar purposes, for the **OPC UA** server and for the **DP** creation scripts and their dependencies. The "mon_opcua_server" is configured by its design files, of which the "config.xml" and the "Design.xml" are the most used. The Cacophony needs the "Design.xml" file to create WinCC Ctrl scripts for the project's **DPTs** and **DPEs**. The "createDpts.ctl" has an already built scheme from the Cacophony execution and the "configParser.ctl" analyses the "config.xml" for the **DPEs** creation.

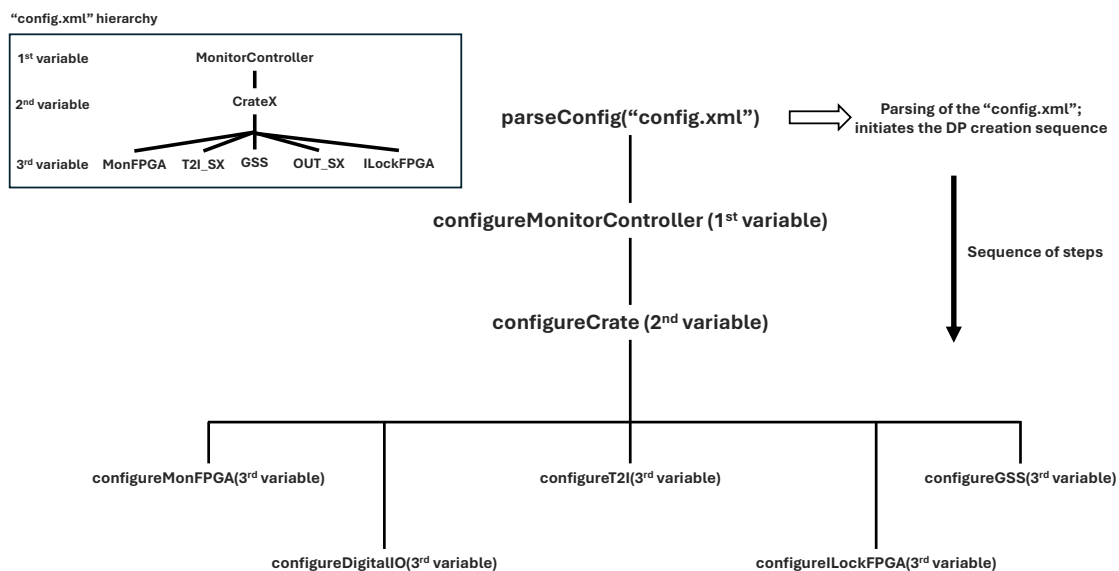


Figure 5.1: Illustration of the **DP** creation process inside the "configParser.ctl". Each process iteration corresponds to a card type dedicated function to create all and every **DPEs** for those types. It depends on the parsing of the "config.xml" file from the **OPC UA** design files. Each variable result from the parsing of this file. On the upper left side is a generic illustration of the "config.xml" hierarchy of variables.

¹¹"leaf-variable" in a tree structure script, as the referred **XML** files, is the lower variable of the tree structure and possesses no child-variables.

5. THE INTERFACE OF THE MONITORING SYSTEM

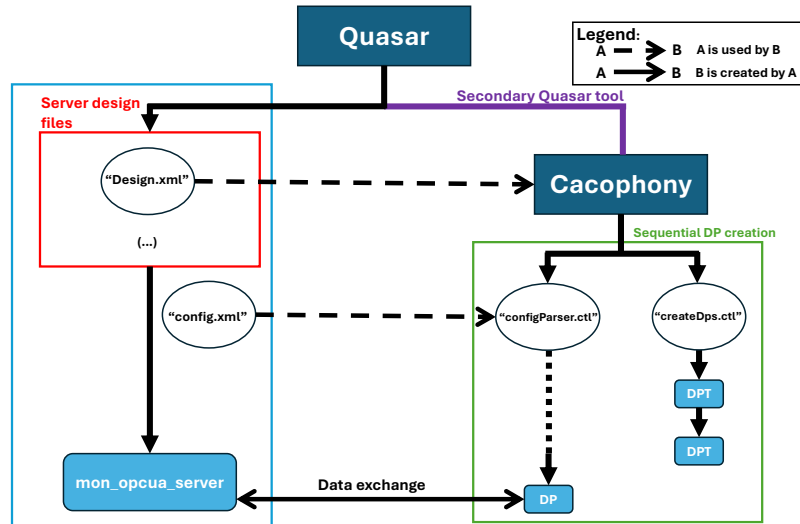


Figure 5.2: Illustration of the Quasar functionality for the Monitoring System. Quasar creates an **OPC UA** server according to the **CERN**'s framework and the Cacophony creates the WinCC scripts to create the necessary **DPs** for the server data. The figure represents the dependencies between both purposes. The "Design.xml" and "config.xml", both from the server design files, are used by the Cacophony tool to create the scripts. One of them, the "configParser.ctf" uses the "config.xml" file for the WinCC **DPs** creation, thus establishing a connection to the server variables.

The **DPs** creation should be executed in the first stages of the WinCC project's development and it only requires to be executed once, unless they suffer modifications. Due to its exclusive purpose, it is necessary to build WinCC panels for their execution. These panels are **graphical interfaces**. They should execute the scripts and should be accessible only for developers and not for end users⁴; they will be provided with the final interface. The Quasar repository already delivers instructions on how to build such panels. Therefore, the panels for the **DPs** creation in projects under the Quasar framework, such as the **ITk** and **HGTD**, usually follow that same structure. With this said, a single panel with two main buttons (a very common event-control interface in **SCADA** systems) was reused. Each button is dedicated to one of the two scripts. Figure 5.3 displays a panel example for this purpose. **The first button executes the "createDpts.ctf"** when pressed, causing the **DPTs** creation and **DPEs** association. When its execution is finished, **the second button**, when pressed, **executes the "configParser.ctf" script** for the final **DP** set up, using the "config.xml" file. For this, a text box allows the introduction of the "config.xml" file directory, since it is a file external to the project. In this way, the script uses the directory written in the text box as an input for its main function, the "parseConfig".

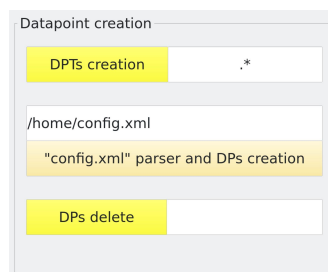


Figure 5.3: WinCC panel for the **DPT** and **DP** creation. Each button is dedicated to one script execution, the "createDpts.ctf" and the "configParser.ctf". Additionally it has a button to remove all **DP**.

The button dedicated to the "createDpts.ctf" was named "DPTs creation" and has a text box which specifies that all **DPT** from the predefined scheme will be created. It is introduced as ".*" ¹². Appendix

¹²In programming, when dealing with pattern matching, "*" usually means all of the available patterns.

D.2 details the codes used for both button pressing events¹³ for the "createDpts.ctl" and "configParser.ctl" scripts. Appendix D.1 studies the code of the "createDpts.ctl" script.

In addition to these two buttons, it is considered good practice to introduce another button for the removing of all or of specific **DPs** and avoid having to remove them one by one, for instance, when the necessary **DPs** suffer modifications. It was designated as "DPs delete". As the creation process, the deleting of the **DPs** is strictly sequential, with the first having to remove each **DP** and then the **DPTs**. Appendix D.2.3 details on the script implemented when this button is pressed.

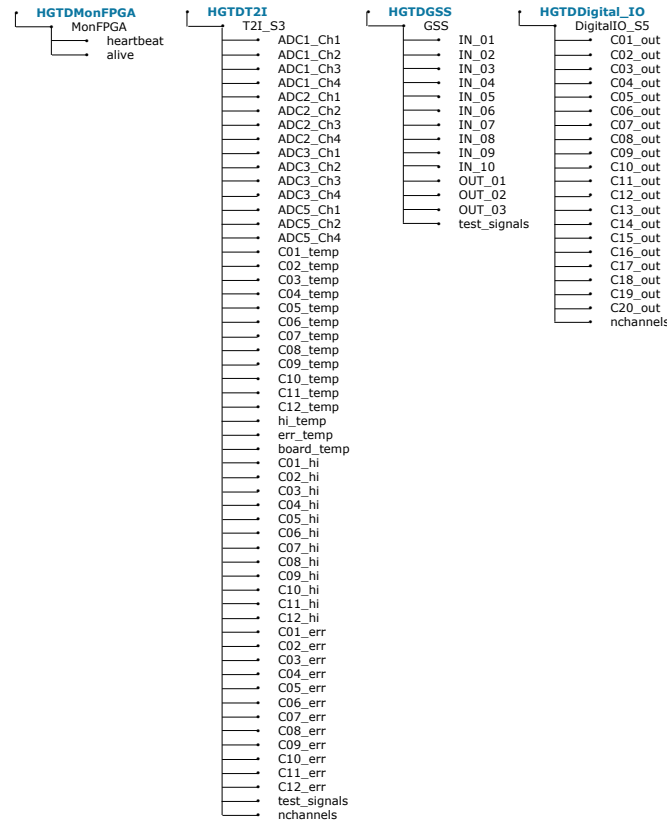


Figure 5.4: Monitoring project's main "datapoints" tree scheme for the **Mon-FPGA**, **T2I**, **GSS** and **OUT** **DPTs**. The blue category represents each **DPT** mentioned.

As shown in Figure 4.4, the material supplied for the **HIS** local development and for this thesis work included a **LISSY** crate: one **Mon-FPGA**, one **T2I-12**, one **GSS**, one **OUT-20** and one **ILK-FPGA** card. With this said, Figure 5.4 displays the most important server variables and the WinCC **DPs** created, under each individual **DPT** (at blue). The **Mon-FPGA** has the "heartbeat" and "alive" **DPs**¹⁴. The "alive" will be subsequently used as an alarm to monitor the card's and the server behaviour. The **T2I-12** has three different sets of **DPs**. One set for the four **ADCs** it has installed. The first three **ADCs** are dedicated to the converted **NTC** voltage and each one can input four signals, resulting in three **ADCs** for 12 channels. The **ADC** designated as **ADC5** is exclusive for the card's on-board sensors, such as the thresholds temperatures (40 °C: "hi_temp"; and -60 °C: "err_temp") and the **T2I** card's temperature ("board_temp"). This was introduced in Section 4.6.2.1. Additionally, it has two sets of dedicated **DPs** for each channel "ERROR" and "Broken Cable" signals, as formatted in the "config.xml" file (Table 4.1: CXX_hi and CXX_err, respectively). At the end are the test signals' 16-bit bus, transmitted by the

¹³Depending on the event-control interface, there can be various methods of execution, such as "Clicked" (the one used), "DoubleClicked" or "MousseOver", for to execute when the mouse simply hovers the button.

¹⁴These were introduced in Section 4.6.2.1 and will be deepened in Section 5.2.4.1.

5. THE INTERFACE OF THE MONITORING SYSTEM

Mon-FPGA (Section 3.1.2.2). The number of channels of the card is attributed at last.

For the **GSS** card **DPT**, its input and output channels are the only **DPs** represented, excluding its test signal 10-bit bus, which operates similarly to the test signal bus for the **T2I** card (Section 3.1.2.2). The OUT-20 card is represented by the "DigitalIO" **DPT** and it has associated 20 output channels, each one with only one value attribute, the bit value from the **ILK-FPGA** commands. "0" means "OK" state, and "1" means "INTERLOCK" state for that channel¹⁵. The **ILK-FPGA** does not possess any monitoring variable at this stage of development.

5.2.4 The LISSY panels

With the **DPs** created, the next step would be the monitoring panels that would display the updated **DPs** values. The more important addition from the **ITk** project migration was certainly this advanced monitoring panels that have already been tested and used in several dedicated systems, although they are still in development. These panels allow to monitor every signal entering and leaving the crate. Besides this, they are quickly integrated into new Interlock layouts. Considering this, it can be stated that the overall Interlock System, from the hardware layout to the software supervisor system can be easily adapted to new setups, which is one of the greatest advantages of the **HIS**.

The modification of the crate's layout, by removing or adding new cards in new slots, only requires the adjustment of the files that describe such layout. The **Mon-FPGA** firmware already allows a *plug and play*⁴ mechanism and the server only requires the update of the new layout by modifying the "config.xml" file. The "Design.xml" does not require a change because it describes the core of each type of card. So, the addition of new cards should reuse the same structure of the older ones, independently of its sub-type (T2I-12 or T2I-16 and OUT-20, OUT-24 or OUT-32). As it was detailed in the sections above, the project only needs the modified "config.xml" to create the correct sets of **DPs**, and the panels should automatically re-adjust to cover such changes, by adding or removing the equivalent layout's slots and cards.

The main panel should mirror the crate layout being monitored, only including the **I/O** cards (**T2I**, **GSS** and OUT cards), while specifying the **T2I** and OUT sub-type for reference. Each active slot (with a card installed in it) should have displayed all the channels being read. Recall that the script creates a fixed number of channels for these two cards, corresponding to their maximum number of channels. However, not all created channels are connected to the server. This means that, although there are 16 and 32 channels recognized by the project for any **T2I** and OUT card, only the ones receiving readings are actually updated and shown on the panels. The cards should be grouped by their type and placed by slot number order, except the **GSS** card, that, conventionally, should be placed between the **T2I** and the OUT cards. The **T2I** and **GSS** slots should have accessible test signals panels to enable the testing of their individual channels by the **DCS** user reading this interface. This testing should order the **Mon-FPGA** to set the specific test signals on the testing channels of those cards, causing a chain reaction and, consequently, the interlock¹⁰ of those channels. This was explained in Section 3.1.2.2. Apart from the card's status, the panel should display the server status and the **Mon-FPGA** communication and data exchange process status with the server, using the "heartbeat" and "alive" variables.

The panel adaptability, for different Interlock System's layouts, is achieved because the monitoring panel is sliced into smaller parts. Each part comes together in a generic monitoring panel. Taking as reference any **I/O** card in the panel, each channel is represented as an individual reference panel ("T2I_ch_reference.pnl"¹⁶, "GSS_INch_reference.pnl", "GSS_OUTch_reference.pnl" and

¹⁵Same bit association for the **T2I** channels interlock signals, "ERROR" and "Broken Cable".

¹⁶The WinCC panel's file extension is ".pnl".

"OUT_ch_reference.pnl" for the **T2I**, **GSS** inputs and outputs and OUT channel panels, respectively). These **channel's reference panels** should display the channel's number and status. Then, each card has a generic slot frame to place these individual channel's reference panels ("T2I_reference.pnl", "GSS_reference.pnl" and "OUT_reference.pnl", for the **T2I**, **GSS** and OUT card's dedicated reference panels). These **card's reference panels** are specific of the **I/O** card in question and they have their length adjusted according card's channel number. Following this mounting scheme, a **generic crate panel** ("LISSY.pnl"), also adjusts its dimensions to place every card's reference panels, from left to right, **T2I**, **GSS** and the OUT card, displaying the entire corresponding **I/O** layout of the **LISSY** crate being monitored. All the panels mentioned above are referred to as the reference panels and can be reused multiple times. Each crate should have, nonetheless, individual generic panels, designated as **implementation or top panel** ("LISSY_Crate1.pnl"), which calls the generic crate panel to display the slots and data specific of that crate. The difference between the **generic crate panel** and the **implementation panel** is that the former serve as a generic scheme for every crate layout and the latter is specific of a crate layout. There is only one generic panel, but there can be several implementation panels. So the **HIS** project, which plans on having four **LISSY** crates, should have four implementation panels, each one dedicated to each crate and with the conventional name "LISSY_CrateX", with "X" being the crate number. Each implementation panel, calls all the other references panels to build the desired interface. Basically, the panel's granularity is down to the card's channels and builds itself up into the entire crate. This method reduces the maintenance effort required to operate such panels, since different card's sub-types and different crate layouts do not necessary require different panels. The reference panels are used between all the cards of the same type. For the building process, the data exchange with the server is initiated in the generic crate reference panel, which exchanges the necessary information to know what cards are being monitored, what sub-type they are and the slot position they are installed in the backplane. This causes the panel referencing chain between the **I/O** card's reference panels and their individual channel's reference panels. Figures 5.5 and 5.6 exemplify this chain of referencing panels until the implementation panel is achieved.

The panel's size adjustment for the card's reference panels and the generic crate panel to carry the necessary channels and slots, respectively, is done with a "spacer", which is an event-control interface, the same as a button, but allows to automatically adjust the dimensions of the panel according to pre-defined parameters. This allows to stretch the panel's frame to fit the necessary reference panels, when knowing the number of channels of the card in question and the number of cards in the crate. To apply the "spacer", a desired area should be defined and the stretching parameters should be introduced and applied at the panel's initialization. The channel and card iteration in the building of the panels is supported by some Ctrl scripts developed by the **ITk** team. These carry functions which have *ITk foot prints*, such as its name as "ITK", but this fact does not affect the panel's behaviour when adapted to the **HIS**. These functions were maintained, due to the numerous dependencies between them. It also allows to keep future collaborations between the **HIS** and the **ITk** Interlock System teams, by having both projects as standardized as possible. Most of the functions purpose is to iterate on the **DPs** created for each card and collect the updated values to display them.

The most used functions come from the "LISSY.ctl" script, such as the "ATLITK-LISSY_statusCircleValueChange" and "ATLITKLISSY_tempBoxValueChange", which display the update values of the connected **DPs** for the interlock signals colours (red for "INTERLOCK" state and green for "OK" state) and the estimated values of temperature, respectively. Their implementation is presented in Appendix D.3 as an example of the connection of the **T2I** channel's interlock signals and temperature values.

5. THE INTERFACE OF THE MONITORING SYSTEM

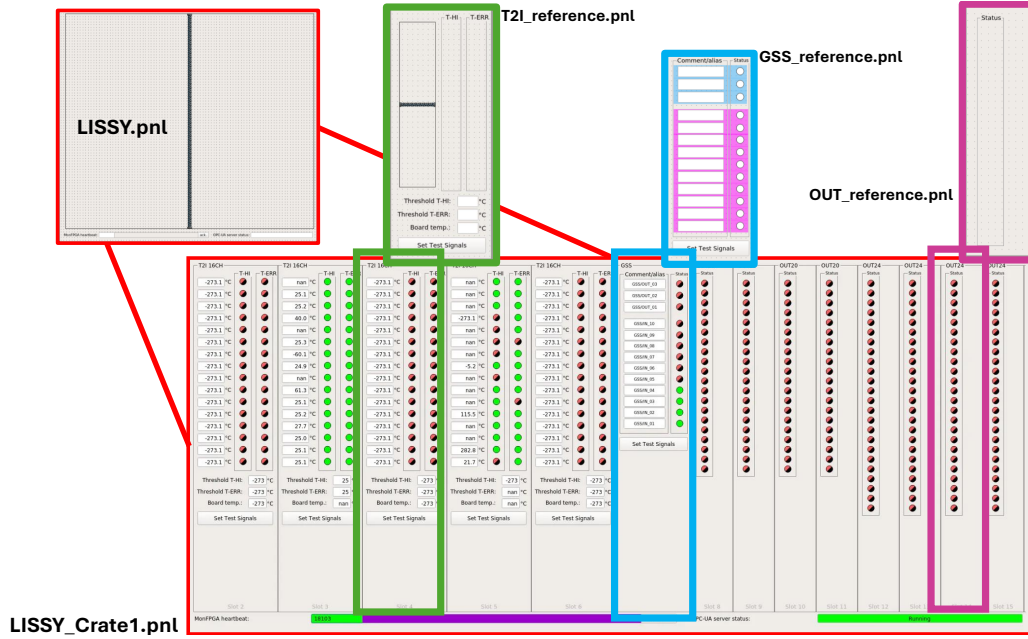


Figure 5.5: Example of the implementation panel's deconstruction and reference panels procedure. The red frame refers to the generic crate reference panel ("LISSY.pnl"), which is used by the implementation panel ("LISSY_Crate1.pnl") for the presented layout. Each individual card is built from the specific card type's reference panels ("_reference.pnl"). The colours are specific of these card type's reference panels (green for T2I, blue for GSS and pink for OUT cards). For instance, each T2I slot uses the "T2I_reference.pnl". The LISSY layout represented was not actually implemented, thus the values displayed do not represent a real implementation and serve only for demonstrative purposes.

Additionally, the implementation panel displays in its lower side the **Mon-FPGA** "heartbeat" value, a section for the "alive" DP (this will be addressed in Section 5.2.4.1) and a section that shows the status of the server connection. If the server goes down, it displays a red colour to trigger the user attention. If not so, it displays a green with "Running" written on it. Figure 5.7 shows the implementation panel for a LISSY layout interface for the local system¹⁷, having one T2I-12 on slot 3, one GSS and one OUT-20 on slot 5. Figure 5.7 also shows the T2I and GSS card's test signals panels, accessible through an "open-on-click" button with "Set Test Signals" written on it. This launches a panel to test each channel individually, which is built upon the same method as the card's individual panels. These panels are built from **testing channel's reference panels** and it gathers them on a **generic "spacer" panel** that adjust its dimensions according to the number of channels. The difference from the channels reference panels is that it has an additional check box to communicate to the **Mon-FPGA** that the corresponding channels should be tested. By checking the box, the test signal to that channel is activated, and the system should react to it, by forcing a false interlock signal in it, causing the respective OUT channel to turn off.

It is also possible to observe in Figure 5.7, that the panel's displayed temperatures have a resolution of 0.1 °C. The HGTD planned NTCs have an accuracy of 1% applied to the R_{25} and the β values, which are used in equation 4.1. This equation reflects a non-linear relation between the sensor's resistance and temperature. So, these variable's accuracies will affect differently the temperature measurements at different temperatures. However, according to the manufacturer's documentation, these parameter accuracies reflect an *accuracy of ± 0.5 °C or better* on the temperature measurement [44]. So, for this thesis, the NTC's accuracy will be fixed at the worst case-scenario, with ± 0.5 °C, which has the same order of magnitude as the panel's resolution. The 0.1 °C resolution seemed appropriate for this implementation, although the NTC's accuracy is only 0.5 °C, because rounding these estimated values to 1 °C (a resolution of 1 °C instead) would result in a loss of valuable information, given the sensor's accuracy

¹⁷The layout can be seen in Figure 4.4.

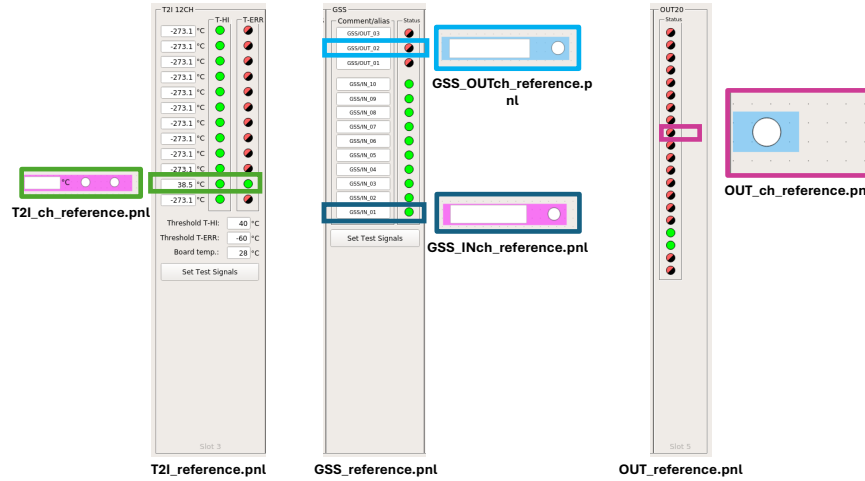


Figure 5.6: Each individual card slot is built around channel's reference panels. Each type of card has dedicated channel's reference panels. The **T2I** channel's reference panels display the respective **NTC** temperature and both interlock signals ("ERROR" and "Broken Cable"). The **GSS** has reference panels for the inputs and outputs and the OUT channels only display the interlock signal.

variability.

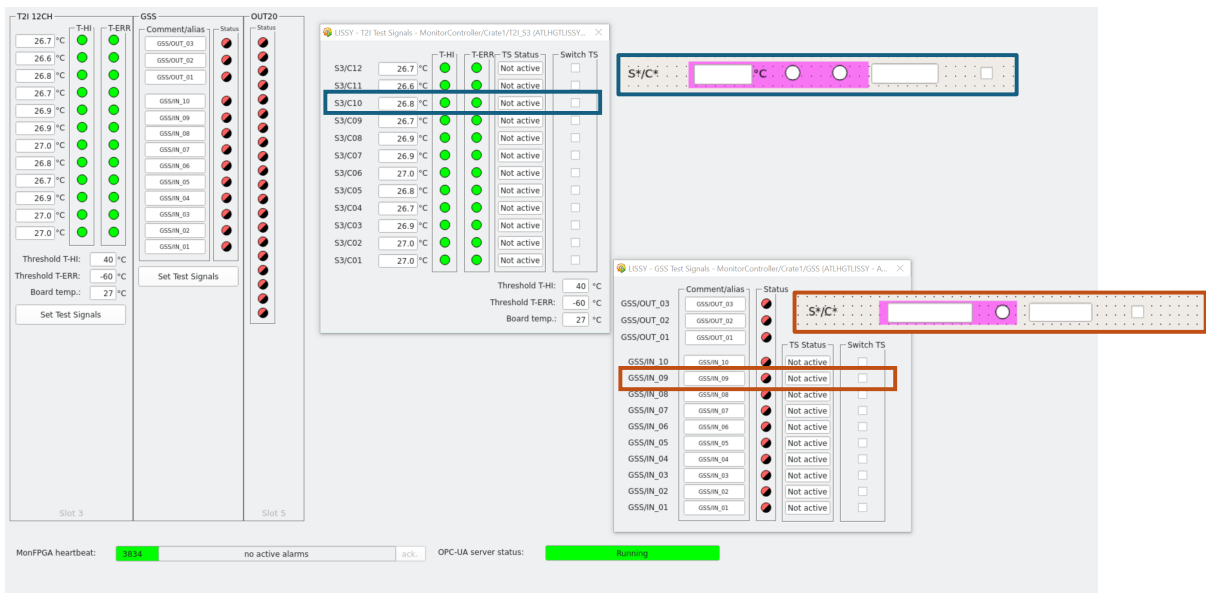


Figure 5.7: Example of a **LISSY** crate implementation panel with the local system layout (1 **T2I**-12, 1 **GSS** and 1 **OUT** card) and the **I/O** card's test panels. The **GSS** interlock signals are all at "INTERLOCK" state, thus the OUT channels are all turned off, although every **T2I** channel is at normal temperature and no "Broken Cable". The test panels are built around individual channel's reference panels and allow to select a channel to be tested using the "Switch TS" column ("TS" means Test Signal).

The panels, as the scripts explained above, can all be found in the **HIS** gitlab repository [11]. All panels shown in this section are displayed with the WinCC tool, the "QuickTest", which allows to test the selected panel as if it was accessed by the end user. The project's adaptability to different crate layouts was tested and did not present errors. As an example, Figure 5.5 displays an implementation panel ("LISSY_Crate1.pnl") layout with multiple **T2I** and **OUT** cards in the same crate, and in the Demonstrator implementation it was added a second **T2I**-16, besides the already used **T2I**-12. None of these situations represented a risk for the project's flexibility. To use this project as a multi-crate project, it is necessary to duplicate the implementation panels, although with their panel name modified to comply with the **OPC UA** server naming conventions for each crate data ("LISSY_CrateX", with "X" being the

5. THE INTERFACE OF THE MONITORING SYSTEM

crate number).

Panels for the T2I, GSS and OUT cards were implemented to display cards of only one of those types installed in the crate. They are based in the crate implementation panel's structure. They do not add new information for each type of card. Yet, they should be used for future implementations to display more detailed status of each type of card.

5.2.4.1 The Mon-FPGA "alive" alarm

As stated many times throughout this thesis, the Mon-FPGA carries two main variables, or WinCC DP's - the "heartbeat" and the "alive" variables. The "alive" is a function of the "heartbeat"; both variables are calculated within the OPC UA server configuration, as explained in Section 4.6.2.1. Shortly summarising, the "heartbeat" counts the communication cycles between the Mon-FPGA and the PYNQ board, while the "alive" verifies if those cycles are within a normal activity. Both are used for monitoring and alarm purposes of the Mon-FPGA behaviour through the DCS panel. The "heartbeat" estimates the **amount of data update cycles** (when the Mon-FPGA updates the PYNQ board memories, after PYNQ requests it to) **in a server update cycle** (when the actual server gets updated by the new values, stored in the PYNQ memory)¹⁴. These cycles have normal, although very tenuous, limits of operation, meaning that if the "heartbeat" gets outside these limits, it does not necessarily mean something is dangerously at risk. Network communication has very common and normal fluctuations, mostly when the server/client data exchange is still in development phases and it runs in local and crowded networks, as it is the HIS in this thesis work. The expected "heartbeat" count is around 25 cycles (that is, there should be 25 data update cycles per server update cycle). The **data update cycles are controlled by PYNQ's internal clocks**. In contrast, the **server update cycles** result from the Quasar framework tool and **are controlled by the PYNQ OS scheduler**, and because they are asynchronous, their execution time can vary, as opposed to a clocked process. This uncertainty in the duration of **the server update cycles is the primary factor** contributing to the variation in the "heartbeat" counts [72].

The "alive" variable is obtained from the normalization of the difference of successive "heartbeat" values, what allows to ascertain the values' consistency. The successive differences are executed to calculate the new "heartbeat" count. This is because the **"heartbeat" values are summed up** (the "heartbeat" is an accumulator variable). So, the only way to obtain the new value, is **to subtract the last one from the current one**. The "alive" is also obtained from the normalization of the new "heartbeat" count according to the expected number of cycles, 25 [72].

With this being said, the "alive" variable should be analysed by taking as normal operation the value of 1, instead of 25. When it is **lower than 1** (< 25), means that **less data update cycles are being achieved within a server update cycle**. This is possibly attributed to the server update cycles finishing overly ahead of the schedule, forcing less data to update the PYNQ's memory. If the value is **higher than 1** (> 25), means **more data update cycles are being accomplished within a server update cycle**. This is a consequence of the server schedule system being late, compared to the normal operation, forcing more data to be updated in the PYNQ memory. Considering this, the "alive" variable can easily be used to study and supervise the Mon-FPGA/PYNQ data exchange and be used as an WinCC alarm to allow the DCS user to accomplish such task [72].

So, the "heartbeat" and the "alive" DP's are used at the bottom side of the implementation panel (Figure 5.7). The "heartbeat" current count is shown and an alarm is triggered when "heartbeat" variations fall outside the normal count cycles. This alarm will stay active unless the user clicks on the "ack" button to acknowledge the alarm. Yet, this can only be achieved when the alarm no longer represents a risk. This behaviour is accomplished by having two phases. The **"Came"**, which is **activated when the alarm is**

triggered and it **stays** that way until the value returns to the normal operation; and the **"Went"**, which **is triggered when the value returns to the normal** interval of operation. Unless the user acknowledges the alarm when no longer represents a risk - when at "Went" - this phase will stay active. Thus, the "ack" button can only be used when the alarm is at the "Went" phase.

To configure an alarm, it is necessary to configure the **DP** on which the alarm is used, in this case the **Mon-FPGA "alive" DP**. To associate a **DP** with an alarm configuration it is necessary to add a new parameter to the given **DP**. One of the most important WinCC tools is the PARA, which allows to access and configure every **DP** and **DPT** in the project, whether they are internal or normal. So, by accessing the "alive" **DP** in this panel, it is possible to add a new configuration of the type "_alert_hdl" ("Alert Handling"). This provides a customized range interface to introduce the limits on where the alarm should actuate. As previously mentioned, the limits should be defined around 1. Since, the "heartbeat" can fluctuate a lot without necessarily mean that something is at risk, it is good practice to also define a range of normal operation and not use the exact value of 1, which will seldom present. The established ranges are as follows [72]:

- (1) $0 \geq \text{"alive"} \Rightarrow$ Server or **Mon-FPGA** stopped
- (2) $0.9 > \text{"alive"} \Rightarrow$ **OPC UA** server ahead
- (3) $0.9 < \text{"alive"} < 1.1 \Rightarrow$ Normal operation range
- (4) $\text{"alive"} > 1.1 \Rightarrow$ **OPC UA** server late

Since the alarm in WinCC has two phases, the "Came" and the "Went", its configuration requires to define messages for each phase and out-of-limit operation. For instance, "Server ahead" for the "Came" of the (2) limit and "Server late" for the (4) limit. The (1) "Came" can be established as "Server/Mon-FPGA stopped". For the three cases, the "Went" messages can be "Returned to Normal". This way, the user can recognize the alarm and acknowledge them through the "ack" button. The normal range does not require a message. Then, the alarm should be activated by the developer on the same panel. It was verified multiple times that the "alive" can go suddenly to the value 0 (zero), but it can recover after some cycles. This was considered as an indicator of some read error bug which should be worked on [72].

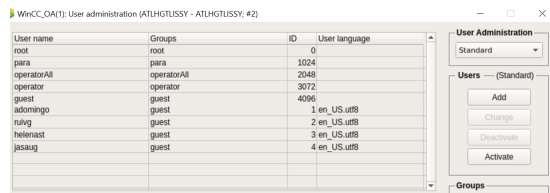
5.2.4.2 The web end user interface

As explained [before](#), although it was not possible to setup a web end user access for the project at **CERN**, one was implemented for the local **HIS** at the laboratory. So, this task was not executed for the Demonstrator implementation. It allowed to access specific panels, the ones dedicated to the end user (for instance [5.7](#)), unlike the **DP** creation panels, which belong to the developers.

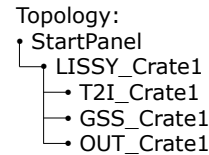
It required adding a new Ctrl manager (WCCActrl), from the Processing and Control group, in the Console interface and specifying a software Ctrl script, from the `scripts/` directory of the project folder, called "webclient_http.ctl". This script enables end user access to specific panels through a web browser. It should use the computer's or machine's **IP** address where the project is running, and a dedicated access network port¹⁸ for end users (8079). This functionality is only possible if the project has its topology correctly configured. This specifies which panels can be accessed, their arrangement and the relations with each other. For instance, one main panel to show the general status of the crate being monitored and then its child-panels (hierarchical scheme) should be specific to each card type, such as the **T2I**, the **GSS** and OUT card, displaying more detailed information of the cards or just to filter them by their type,

¹⁸Communication entry point for a web page. Each port usually has distinct functionalities and dedicated user's type.

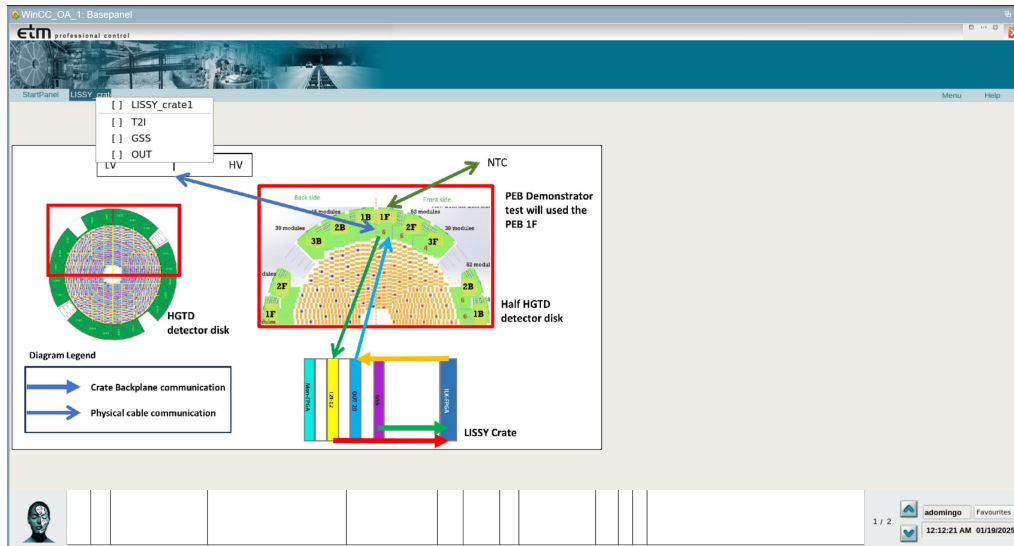
5. THE INTERFACE OF THE MONITORING SYSTEM



(a) User Administrator's panel where each user can be configured.



(b) The configured topology: "LISSY_Crate1" is the child-panel of "StartPanel" and the "T2I_Crate1", "GSS_Crate1" and "OUT_Crate1" are the child-panels of "LISSY_Crate1".



(c) The web interface of the end user accessed through the browser. At the bottom of the interface is the alarm section, with no alarm triggered and the user is the "adomingo". The panel's topology is shown in the upper side of the interface.

Figure 5.8: Web end user interface configuration

for performing a specific status analysis. This is possible through the software Topology panel on the GEDI's interface, by selecting which panels should be added and their hierarchical configuration.

The user access configuration for the web browser login process can be achieved through the project's System Management and Permission's tab and then in the User Administrator's panel, which enables adding, removing and modifying existing users and their administrative groups (guest, developer or root, for instance). With this, it was possible to add users for the local HIS developers. The main difference from the JCOP "fwAccessControl" component (referred in Section 5.2.2), is that the user administration is not done by the developers, but rather by dedicated CERN departments. Figure 5.8 displays a set of figures related to the web end user interface configuration.

Figure 5.8 (a) displays the User Administrator's panel with custom user configuration for the local HIS development ("adomingo", "ruivg", "helenast" and "jasaug"), all as "guest" users, meaning they have restrictive access, mainly just to access the monitoring interface. Figure 5.8 (b) displays the configured topology, with a "StartPanel" and its child-panel, the implementation panel ("LISSY_Crate1"). The latter has as child-panels the interface for each type of card, individually. The format "_Crate1" is followed by these to avoid conflicts in complex Interlock Systems involving more than one LISSY crate. Each type of card's panel displays the same information as the "LISSY_Crate1", but without the other types of cards. These three specific panels were not originally from the ITk project. They were added to test the end user web access interface by having more than one implementation panel, the "LISSY_Crate1". They were shortly introduced as additional work at the end of Section 5.2.4. Figure 5.8 (c) displays the end user interface, specifically, the "StartPanel" with simple information about the system and its functionality in the Demonstrator implementation. In the upper left side of the interface is the topology

of the panels, with each accessible panel. The interface was chosen between original WinCC templates, but it is possible to import customized functional interfaces.

5.2.4.3 Database configuration

Any type of supervisor system requires a database to archive the monitored data. This allows the user to study the data history and its past behaviour in case of necessity, such as under error or abnormal operation. It either aids to understand the current data or to prevent or predict future data updates. Given the advanced development of the electronics and monitored systems of the Interlock System, this issue was not a focal and crucial point of this thesis at this stage of the HIS development. Nonetheless, the project's and DPs' configuration to support data storage was set up, although the database establishment had not been settled.

The database connections on WinCC are dealt by the managers of the Communication and Archiving group. Being from the same group of managers, these support the "Event Manager" by handling the configuration of the data to be stored, linking the database backend¹⁹ to the project's DPs. The data being stored can include not only values but also alerts from the system's behaviour, facilitating point error detection, by comparing the alerts with the data at the time it was triggered.

The configuration that allows this data to be stored is executed in a similar way to the "alive" DP alarm configuration. It is necessary to configure every source of data (DP) individually by adding a new archive configuration to it. As verified several times, the number of DPs to be monitored can easily grow in number, just by inserting another card into the crate, for instance. However, these variables can be easily covered by cyclical loop processes, since they end up being repetitive and have the same conventional identifications. So, to ease the process of configuring each DP, it was developed an automatic mechanism which iterates over these repetitive variables, similar to the creation of the DP. This process is carried by a panel designed for the task.

As it was mentioned in Section 5.2.3, every I/O card has the maximum number of channels created on the project, although not every one is used. The selection of the DP that should be stored is a task of great importance: one has to evaluate if it is desirable having a lot of variables' values being stored, with data without interest being monitored (even just for a test run), which leads to do more maintenance to avoid overloading the database, or choosing a more conservative data selection, having less data to study over time but with less maintenance required. The storage of DPs that represent no value addition to historical data studies, such as the additional channels of the sub-type card variants of the T2I and the OUT cards that are not even connected to the server, represent a disadvantage to the database development. This issue was dealt by having distinct storage processes available, **one for a defined number of channels** for each of these two cards and **one that includes every channel DP** for them. Since the HIS is expected to have the same card sub-types per crate, an archiving process for only one sub-type of the cards was seen as enough for the current stage of the HIS.

Figure 5.9 displays the three panels developed for the automatic archiving mechanism of the DPs of interest. Figure 5.9 (a) is the main panel and the one that enables the configuration of the DPs. In the first section, it is possible to identify which sub-type of T2I and OUT cards are being used, by defining their number of channels. This allows to reduce the number of DPs with no value and the overload of the database with unnecessary data, as explained above. The "Archive datapoints" button applies a Ctrl⁹ script which configures every channel until the defined number is reached.

If the crate uses more than one sub-type, the button below ("Archive ALL datapoints") is more appropriated, since it can archive every channel's DPs from both card's types, even if they are not updated

¹⁹A database backend is where the user can access to study the data stored there.

5. THE INTERFACE OF THE MONITORING SYSTEM

with values. This section is also meant for the layouts which only possess T2I-16 and OUT-32 cards. On the section below, the "Remove ALL Archive Configs" can delete the archive configuration of all card's **DPs**. Even if they are not configured with an archive functionality, the project neglects them, with no effect on the project's behaviour. It has a similar purpose to the "Dps delete" button from the panel in Figure 5.3.

Since these projects are frequently migrated from system to system, it is crucial to set the development tools as easy and understandable as possible. With this said, it was also developed help panels to support future collaborations by detailing the mentioned panel description. By clicking on the "?" button, on the lower right side of the panel, one accesses a help panel which describes the functionality of the original. This panel is shown in Figure 5.9 (b). To back up this panel, by clicking on the green button, a more visually support panel is opened, that oversimplifies the above description (Figure 5.9 (c)).

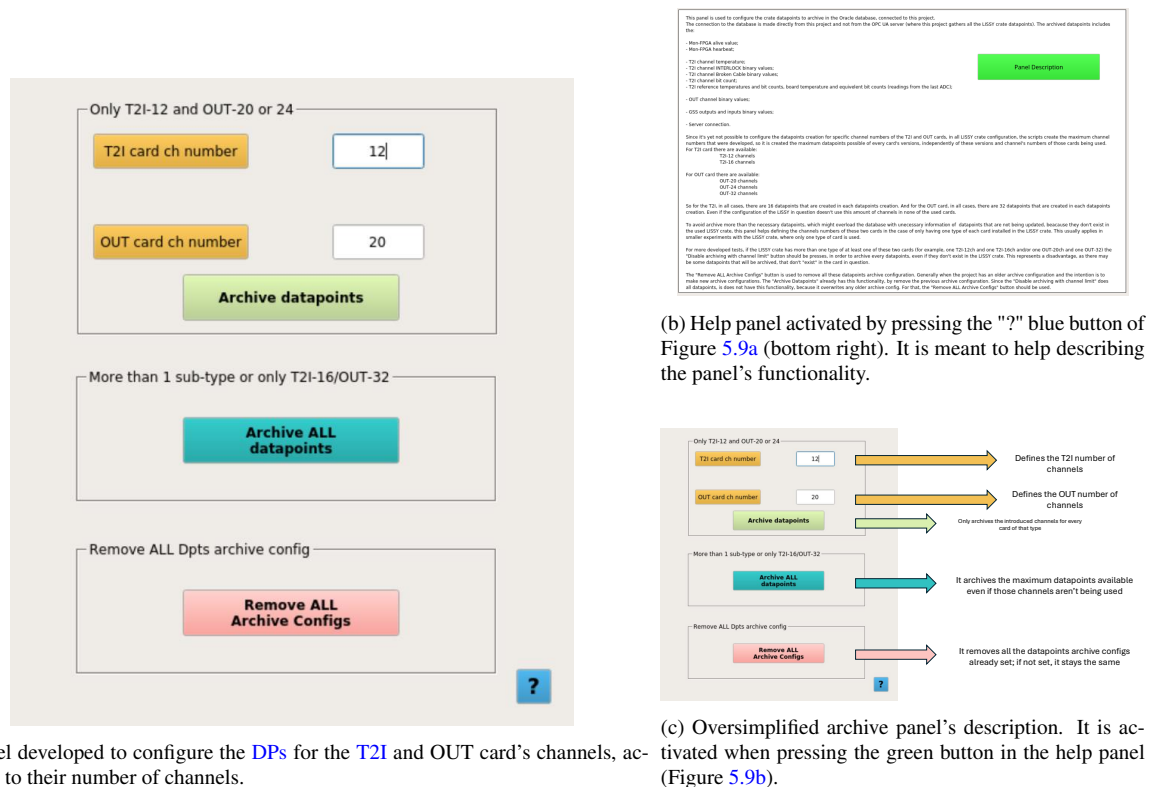


Figure 5.9: Archive panel deconstruction.

The archive configuration is automatically executed through the use of a framework function, the "fwArchive_set". It uses as arguments the **DP** "address name"⁵, the database backend, the "smoothing²⁰ mode" of archiving (applied smoothing or not) and the smoothing flags if applied ("smoothing type"). The smoothing can be done by **value variation** or **after a time window** or **both**. Depending on the nature of **DP**, the smoothing flags can be different. For instance, the **T2I channel's temperatures** are stored with smoothing in the "value or time window": either if there is a temperature difference larger than 0.1 °C or if there is an hour without any change. The 0.1 °C value filtering is the resolution displayed on the implementation panel. For the **interlock binary signals** of the **T2I**, **GSS** and **OUT** channels there was no desire to store the same value after 1 hour, even if it did not change. So, they were defined only with smoothing in the value as whenever it changes (through an old value-new value comparison), from "1" or "0" or vice-versa. The smoothing of the **T2I ADC converted digital word** for each channel also

²⁰In the jargon of WinCC OA documentation, "smoothing" does not refer the electronics' smoothing, this is, to an averaging or low-pass filtering of the data. Instead it is a mean of filtering and of reducing the data rate storage [69].

tracks changes in the value. The **T2I** and **GSS test signal bit bus** also applies this smoothing, as well as, the **Mon-FPGA** related **DPs**: the **"heartbeat"**, **"alive"** values and the **server status**.

These **DPs** smoothing should be analysed with the database connection established, since their values change frequently, without representing any variations of interest. This could only be done with various tests and iterations under an active archiving. If the database overloads easily, possibly it is necessary to apply a more restrict smoothing process. Under this thesis, these tests were not executed since the database backend was not established, having only the **DPs** configuration set up for a future database connection.

This configuration was executed in the WinCC 3.19 version used in the **CERN's Alma9 OS** machines, aiming the Demonstrator implementation. The database connection requires the authorization of dedicated **HGTD** departments and it was not done. The scripts executed by the panel's buttons follow the format of the Code 5.1.

Code 5.1: Archiving mechanism implemented on Figure 5.9 (a) panel button to configure a number of channels from the **T2I** cards. Only displays the loop on the **T2I "ERROR"** (high temperatures) interlock signals.

```

1  int t2i_chNumber = TF_T2I_chNumber;
2  int out_chNumber = TF_OUT_chNumber;
3  (...)
4  dyn_string dtps_t2i_hi = dpNames("ATLHGTLISSY:MonitorController/Crate1/T2I_S*.C*_hi");
5  for (int n =1; n <= t2i_chNumber; n++) {
6  //      fwArchive_set("address name", database backend,
7  //                  smoothing mode, smoothing type, value range, time range)
8      fwArchive_set(dtps_t2i_hi[n], "NGA) Oracle - EVENT",
9                  DPATTR_ARCH_PROC_SIMPLESM, DPATTR_COMPARE_OLD_NEW, 0, 0);
10     DebugN("Archiving set for: "+ dtps_t2i_hi[n]);
11 }

```

The "address name" format for these **DPs** is defined as "ATLHGTLISSY:MonitorController/Crate1/T2I_S*.C*_hi"¹². The **T2I** card has the format of "T2I_SX", with "X" being the slot number and the channels with the value of the "ERROR" signal are designated as "CXX_hi", where "XX" is the channel's number. Section 4.6.2.1 details this convention. On lines 1 and 2 are defined integers to represent the channels number for both **T2I** and OUT cards using the panel of Figure 5.9 (a). The TF_T2I_chNumber and TF_OUT_chNumber are the names of the text boxes on this panel used to introduce the channel's number. In this way, the script reads the introduced text and converts it to integer numbers, to be used by the script.

On line 4, dyn_string represents an array of strings, in this case with the **T2I "ERROR"** signals **DPs** "address names". These are obtained with the WinCC function "dpNames", that automatically gathers all the **DPs** that follow the specified format. In this case the **DPs** with the name "CXX_hi" from every **T2I** card with the format "T2I_SX".

By using the array of strings and the limit of channels (t2i_chNumber) that the script should cover, it is possible to iterate over every variable on that array to apply the "fwArchive_set" function shown in Code 5.1. It is good practice to leave a comment message on the project's log viewer, to ease on the debugging (line 10). This process is iterated over the **DPs** of the **T2I** channel's temperatures, "Broken Cable" and **ADC** digital words, **GSS** inputs and outputs and OUT channels. The **Mon-FPGA** associated **DPs** are configured individually. On line 9 the smoothing value and time ranges are not applied given the selected smoothing type (DPATTR_COMPARE_OLD_NEW: old value-new value comparison). The deleting of the archive configurations is executed with the "fwArchive_deleteMultiple" and "fwArchive_delete"

5. THE INTERFACE OF THE MONITORING SYSTEM

functions to delete multiple and a single **DPs** configuration, respectively. These functions use an array of strings with the "address name" of the **DPs**, provided by the "dpNames" function (similar to line 4).

6 The preliminary testing of the system

Despite the advanced development status of the I/O cards designed by the ITk teams, the necessary tests on the local laboratory, in Lisbon, were performed to ensure the proper operation of the system before the final implementation on-site at CERN, for the Demonstrator PEB. This also allowed to test and validate the monitoring interface (its deep overview is in Chapter 5) in view of future applications.

The tests were dedicated to the I/O cards channels, by simulating in and out-of-limit situations, on the T2I and GSS cards, which work as the system's inputs. The OUT card mirrored the system reaction to these input signals. It was built a circuit to visually demonstrate the system's reaction to the interlock signals at the OUT channels. This also allowed to test not only the correct operation of the conditioning circuits of each card, but also the system chain of reaction, implemented by the interlock matrix of the ILK-FPGA. In parallel, the Mon-FPGA should provide a second confirmation for the system's behaviour through the user's interface.

The test of each type of card was split among the local HIS developers. This thesis focuses on the T2I-12 board in-depth testing, although it also mentions the streamlined tests applied to the other I/O boards. This chapter will study the main components of the T2I board in order to understand how to approach their electronics' verification and validation. It will also describes the tests procedures, results and conclusions, validating the board before the Demonstrator implementation. The required temperature threshold accuracy is 1 °C [42].

The Mon-FPGA requirement of testing the card's channels through dedicated test signals is also addressed in this chapter. Although all I/O cards were validated for the Demonstrator implementation, the Mon-FPGA test signals were not reliable after the conducted tests and were not used in this stage [41].

6.1 The T2I board

The T2I with 12 channels is one of the interlock signal's entries in the HIS. It transmits, to the crate backplane, the NTCs' voltages connected to the card's channels. The backplane transmission allows to communicate to the Mon-FPGA and to the ILK-FPGA the NTCs' status. The T2I-12 board implements a set of conditioning circuits for each individual channel and signal routing circuitry to two sections of the board. One section is dedicated to the transmission of the voltage reading to the Mon-FPGA and the other section to the comparison of the voltage reading with the temperature range limits for normal operation of the NTCs. The latter section transmits the output for both the Mon-FPGA and the ILK-FPGA.

Figure 6.1 illustrates the designed main electronics of one input channel implemented in the T2I board. As it can be verified, it breaks into two sections. Following the signal flow, at the brown frame, the input conditioning circuit, the voltage of the NTC is inputted in the card's channel, which is routed to one of the ADC's ports (purple frame) and to the OP-AMP's section, as the input of two OP-AMPs (red frame). The ADC schematic is presented in Appendix F.1 and the voltage regulators LT3042 are presented in Appendix F.2 along with their voltage supply configuration. In this thesis it is only necessary to know that the voltage regulators provide a fixed reference voltage of $(2.500 \pm 0.002) \text{ V}^1$ for the ADC, R_REF and input circuitry.

¹The voltage regulator accuracy is considered as the manufacturer [47] output voltage offset (Appendix F.2).

6. THE PRELIMINARY TESTING OF THE SYSTEM

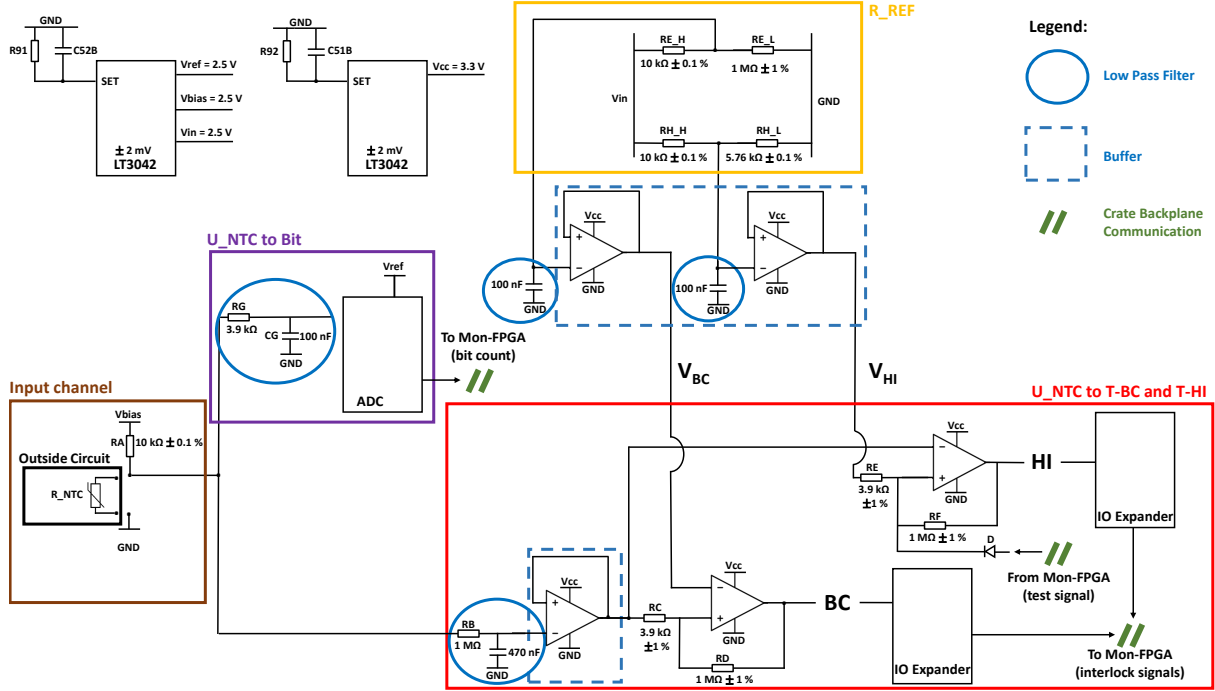


Figure 6.1: Illustration of the electronics of one input channel of the **T2I** board. **Brown frame** - Input channel: circuit that includes the **NTC**, installed outside of the board (**black frame**), connected through one of the card channels. It is directly connected to the **GND** and to the series resistors, **RA**. The input circuit is supplied by V_{bias} . Its terminals are connected to the rest of the circuitry. The voltage is read by two distinct circuits in parallel. **Purple frame** - U_{NTC} to Bit circuit: the U_{NTC} voltage is read by an **ADC**. The conversion to digital word is then outputted through an **I²C** bus with other channels readings and sent to the **Mon-FPGA** (double green slashes). **Red frame** - U_{NTC} to T-BC and T-HI: the input voltage is read by a pair of **OP-AMPs**, which verify if the voltage is within a normal operational range for the sensors. The output of each **OP-AMP** is connected to different **I/O** expanders (which gathered other channels readings) and send an **I²C** bus to the **ILK-FPGA** and **Mon-FPGA**. **Yellow frame** - **R_REF**: provides the reference voltages to the **OP-AMP** to define the limits of the voltage comparison. It is separated from the **red frame** circuitry with buffers. The V_{bias} , V_{ref} , V_{in} and V_{CC} are provided by two LT3042EMSE [47] voltage regulators (Appendix F.2). This illustration gathered different schematic blocks from the **T2I** board. The Appendix F.3 shows the **OP-AMPs** schematic block for 4 channels.

6.1.1 The input conditioning circuit

The input conditioning circuit of the **T2I** board, or just the input channel, conveys one **NTC** signal. Its is shown in Figure's 6.1 **brown frame**.

Each **T2I** input channel has implemented a voltage divider. This circuit has some resistors in series. In this case, the **NTC**, represented as a resistor² R_{NTC} (R_{NTC} in the illustrations), is in series with the fixed resistor designated as **RA**. The latter has a resistance equivalent to the selected **NTC** at 25 °C, of 10 k Ω . The **NTC** is the only component mounted outside of the **T2I**, since it is meant to be installed on the detector's disks. The voltage U_{NTC} at the **NTC** terminals leads to the **NTC** resistance (R_{NTC}):

$$U_{NTC} = V_{bias} \frac{R_{NTC}}{R_{NTC} + RA} \quad (6.1)$$

$$R_{NTC} = U_{NTC} \frac{RA}{V_{bias} - U_{NTC}} \quad (6.2)$$

where V_{bias} is the supply voltage of the circuit, of 2.5 V. The non-linear relation between U_{NTC} and R_{NTC} is used to get the temperature calculations (equation 4.1), making this circuitry³ reliable, effective and cheap.

²As already referred, the sensors are equivalent to resistors which depend on temperature variations.

³A possible option would be to drive the **NTC** with a precise current source, as is used in the LT3042 [47] in order to relate U_{NTC} and R_{NTC} , but it would be more cumbersome.

6.1.2 The U_{NTC} to Bit circuit

The temperature calculations are executed in the server using equation 4.1, which requires the NTC resistance (Section 4.6.2.1). This is obtained using equation 6.2 as already mentioned. However, the server has to received such measurement to execute the calculations, and it comes from the ADC.

As it can be verified in Figure 6.1 purple frame, the voltage of the input channel is read by an ADC, which, as was explained in Section 3.1.2.3, converts the analog voltage reading into digital word, according to a reference voltage, V_{ref} . The number of bits of the word, or the resolution of the ADC, was 15 bits. The conversion from the analog voltage to digital word (6.3) and vice versa (6.4) is given by following equations:

$$ADC = \frac{U_{NTC}}{V_{ref}} \times (2^N - 1) \quad (6.3)$$

$$U_{NTC} = \frac{ADC}{2^N - 1} \times V_{ref} \quad (6.4)$$

where ADC is the converted digital word and N is the number of bits used in the conversion.

From equation 6.2, the NTC resistance can be obtained from the U_{NTC} voltage, which can be estimated using equation 6.4. This relation gives:

$$R_{NTC} = \left(\frac{ADC}{2^N - 1} \right) V_{ref} \times \frac{RA}{V_{bias} - \left(\frac{ADC}{2^N - 1} \right) V_{ref}} \quad (6.5)$$

V_{bias} and V_{ref} are provided by the same voltage reference component, LT3042EMSE [47], both of 2.5 V. So, by equating V_{ref} and V_{bias} in equation 6.5, it is possible to reduce the calculation of the NTC resistance as a function of the normalized ADC converted digital word, u_{NTC} , according to the digital Full Scale Range (FSR)⁴:

$$R_{NTC} = u_{NTC} \frac{RA}{1 - u_{NTC}} \quad (6.6)$$

where u_{ntc} is given by $\frac{ADC}{2^N - 1}$. This allows to estimate the NTC resistance directly from the converted digital word, skipping the equivalent voltage estimation (equation 6.4). This process was taken from the "steinhartHart" function used in the design file of the OPC UA server (the function's body is analysed in Appendix C). The NTC voltage is filtered by a passive low-pass filter, before being read by the ADC, what allows to remove unwanted high frequency noise from the voltage readings. The selected ADC is the ADS1119 [8], from Texas Instruments, which samples at a maximum of 1 kHz. According to the Nyquist–Shannon sampling theorem⁵, the ADC can only convert signals with a bandwidth of 500 Hz to avoid aliasing. To exclude receiving frequencies higher than that, a low pass filter is used as the input. According to the T2I schematics, the filter is composed by $R_G = 3.9 \text{ k}\Omega$ and $C_G = 100 \text{ nF}$, giving a cutoff frequency⁶ of $\approx 408 \text{ Hz}$. In this way, the ADC will always have a sampling frequency higher than twice f_{cutoff} . The use of these filters also allows to reduce noise on the receiving signal. The voltage converted in digital word is then transmitted to the Mon-FPGA through the crate backplane (Figure 6.1).

⁴The digital FSR is the maximum number that can be represented using a given number of bits. Since there are only two symbols, "0" and "1", the maximum value that can be represented by N number of bits is $(2^N - 1)$.

⁵To avoid the aliasing effect on the receiving signal, the ADC sampling frequency needs to be at least twice the frequency of the signal to be converted.

⁶Frequency at which the output signal has an amplitude which is $1/\sqrt{2}$ of the signal's maximum amplitude. For this type of filter $f_{cutoff} = \frac{1}{2\pi RC}$.

6. THE PRELIMINARY TESTING OF THE SYSTEM

6.1.3 The U_NTC to T-BC and T-HI circuit

The OP-AMP's electronic block (red frame of Figure 6.1) of the T2I card has the purpose of comparing the voltage from the NTC with the reference values defined by the R_REF circuitry (yellow frame). The former is composed by the BC and HI OP-AMPs and the latter can be split into two circuits, which intent to mirror the input circuitry (Section 6.1.1) at the defined temperature limits for the NTCs. Let V_{HI} and V_{BC} be the voltages for the upper and lower temperature limits, respectively, as well as $T_{HI} = 40\text{ }^{\circ}\text{C}$ and $T_{BC} = -60\text{ }^{\circ}\text{C}$ their temperatures. For the selected NTCs [45], the resistance of the component at T_{HI} it is $5.781\text{ k}\Omega$ and by manipulating equation 4.1 to estimate the theoretical⁷ resistance at T_{BC} it is $(0.99 \pm 0.05)\text{ M}\Omega$. This value's uncertainty, as well all the other estimated throughout this thesis, was deduced by the propagation of uncertainties⁸ of the relevant function. It is applied by $\Delta f(x_1, x_2, \dots, x_n) = \sqrt{\sum_{i=1}^n \left(\frac{\partial f}{\partial x_i} \Delta x_i \right)^2}$ on equation 4.1 rewritten as a function $R(\beta, R_{25})$ (6.7) and the partial derivatives are given by 6.8 and 6.9, respectively, which results in the uncertainty of equation 6.10:

$$R = R_{25} e^{\beta \left(\frac{1}{T} - \frac{1}{298.15} \right)} \quad (6.7)$$

$$\frac{\partial R}{\partial \beta} = R \left(\frac{1}{T} - \frac{1}{298.15} \right) \quad (6.8) \quad \frac{\partial R}{\partial R_{25}} = \frac{R}{R_{25}} \quad (6.9)$$

$$\Delta R = R \sqrt{\left(\frac{1}{T} - \frac{1}{298.15} \right)^2 (\Delta \beta)^2 + \left(\frac{1}{R_{25}} \right)^2 (\Delta R_{25})^2} \quad (6.10)$$

However, the T_{HI} established reference voltage (V_{HI}) is given by the series of $RH_H = (10.00 \pm 0.01)\text{ k}\Omega$ and $RH_L = (5.760 \pm 0.006)\text{ k}\Omega$, from R_REF. For the lower limit it is given by $RE_H = (10.00 \pm 0.01)\text{ k}\Omega$ and $RE_L = (1.00 \pm 0.01)\text{ M}\Omega$. Both RH_H and RL_H mirror the RA resistor from the input channel circuit, while RH_L and RE_L mirror the NTC (Section 6.1.1) at the temperature limits.

Directly through equation 4.1, the RH_L resistor gives a reference temperature of $(40.0 \pm 0.3)\text{ }^{\circ}\text{C}$, while RE_L gives the lower temperature of $(-60.1 \pm 0.6)\text{ }^{\circ}\text{C}$. These ranges cover the predefined limits and the required accuracy of $1\text{ }^{\circ}\text{C}$ for the HIS temperature threshold [42]. This validates the resistors choices for the current requirements. The temperatures uncertainties were deduced in Appendix E.1, using $\beta = (3435 \pm 35)\text{ K}$, $R_{25} = (10.0 \pm 0.1)\text{ k}\Omega$ [45].

These reference voltages are then used by the Operational Amplifiers (OP-AMPs). These are wide-band and rail-to-rail⁹ used with the output in saturation. The component selected is the AD8604 ARZ [27] and is generic in all applications in Figure 6.1. The OP-AMPs are widely used in electronics given their flexible implementation range. They are usually characterized by two inputs, the non-inverting (V_+) and the inverting (V_-), and one output (V_{out} or BC and HI in Figure 6.1) and have two power supplies, usually one positive (V_{CC} ¹⁰) and the other negative or GND. The output depends on the inputs and its value is limited by the power supplies of the OP-AMP. So, besides powering the component, they also define the range of possible output values. When V_{out} approximates the power supply voltages, the OP-AMP is considered to operate in the aforementioned saturation, where its output is close to one of the supply values. Otherwise, the OP-AMP operates in the linear mode, with the output given by:

$$V_{out} = A_{OL} \cdot V_d = A_{OL} \cdot (V_+ - V_-) \quad (6.11)$$

⁷The selected NTCs do not actually reach such temperatures, as they are limited to the range $-40\text{ }^{\circ}\text{C}$ to $125\text{ }^{\circ}\text{C}$.

⁸It allows to get a weighted total uncertainty according to how each variable's uncertainty contributes.

⁹The rail-to-rail OP-AMPs are designed to output voltages close to their supply voltages.

¹⁰Provided by the LT3042 voltage regulator, discussed in detail in the Appendix F.2.

A_{OL} is the open-loop gain of the OP-AMP, usually in the order of 10^5 and $V_{+/-}$ are almost equal¹¹.

However, for the OP-AMP to compare its input voltages, it has to operate in the saturation mode. This is achieved by having a reference voltage on one of its inputs¹² and using no feedback or positive feedback. For this case, equation 6.11 is used to predict which output value the component is going to provide. By fixing one of the inputs, the other is the one which is going to be compared. If $V_+ > V_-$, $V_d > 0$, which easily escalates V_{out} by multiplying it by A_{OL} , exceeding the positive supply voltage and resulting in a positive saturation and so $V_{out} = V_{CC}$. On the other hand, if $V_+ < V_-$, $V_d < 0$ and $V_{out} = \text{GND}$ (negative saturation), in the present case. That is the reason why the linear mode results from negative feedback, which does not allow the output to increase over a defined range.

With this said, the reference voltages from the R_REF block are introduced in each of the OP-AMPs. Each output voltage is equivalent to the binary states of the interlock signals: bit "1" ("INTERLOCK") is 3.3 V and "0" ("OK") is 0 V. The OP-AMP which outputs the BC signal defines the lower temperature limit and the broken cable situation and has V_{BC} as its voltage reference at V_- . While the OP-AMP that outputs the HI signal defines the upper temperature limit of 40 °C and has its reference voltage, V_{HI} , introduced in the inverting input, V_+ . Both OP-AMPs' outputs (BC and HI) have an associated uncertainty, which according to the manufacturer [27] is the deviation between the typical value and the supply voltage. The typical application in device's datasheet uses a supply of 3.0 V with a typical output value of 2.95 V (meaning ± 0.05 V) and 0.02 V for the low voltage output, when using GND. So, by extrapolating, it is possible to achieve 0.06 V as the high output voltage uncertainty, with $V_{CC} = 3.3$ V.

Nevertheless, in real-case scenarios, noise can cause sudden and unwanted changes in the input signal, making it switch between saturation stages, when in proximity to the threshold or reference voltage. This can induce errors in the system's operation, by triggering false interlock signals. So, to avoid such scenarios, and despite being already filtered by a low pass filter (resistor RB), each OP-AMP has implemented a hysteresis range in their respective thresholds. The hysteresis creates *no transition zones* for small noise fluctuations, by establishing a range for the reference voltages, instead of an exact threshold value (this being the T_{HI} and T_{BC} , which no longer have effect under hysteresis). As the name implies, there are no transitions inside this range. The *no transition zones*' limits depend of the direction of the transition among saturation regions. So, it is **only possible to break out of this range through specific transition' directions**.

This is achieved by implementing a positive feedback on each OP-AMP, executed by RD and RF on the BC and HI OP-AMPs (red frame), respectively, on V_+ . Also, the voltage from the NTC and the R_REF circuitry are filtered by passive low-pass filters (blue circles in Figure 6.1) and by buffers (blue dash square in Figure 6.1), which allows to isolate the output from the input and to conduct the following analysis of uncertainty, with no interference between blocks.

For every estimated R_{eq} (equivalent hysteresis resistance) and T , the associated uncertainty is deduced in appendices E.3.2 and E.1, respectively. Also, the values obtained from now on were deduced using measurement of the input (V_{bias}), R_REF circuitry (V_{in}) and ADC's (V_{ref}) reference voltages (all being supplied by the same component, the LT3042) by a digital precision multimeter, the HP 34401 A [5]. It was done at open-loop directly on the channels, giving (2.5024 ± 0.0001) V¹³, consistently over every channel. As it can be verified, it is at the uncertainty's limit of 2 mV¹⁴. Given this, it was appropriate to use the measured value over the manufacturer's one, since the goal is to evaluate this

¹¹Property of OP-AMPs: the inputs drive nearly 0 current, given the component's high input impedance, which allows to neglect them and considered almost equal input voltages.

¹²These are the temperature reference voltages mentioned before.

¹³The measurement's uncertainty is given by 0.0035% [of reading] + 0.0005% [scale], while measuring with the 10 V scale.

¹⁴Appendix F.2 deepens this subject.

6. THE PRELIMINARY TESTING OF THE SYSTEM

specific **T2I** board, allowing to obtain a more dedicated analysis, although every other component used the manufacturer's values, due to not being possible to measure them.

For the BC **OP-AMP**, the current at the V_+ input is neglected¹¹, meaning the current at RC is approximately the same at RD, which by using Ohm's law, gives:

$$i_{RC} = i_{RD} \Leftrightarrow \frac{U_{NTC} - V_+}{RC} = \frac{V_+ - BC}{RD} \Leftrightarrow V_+ = \frac{U_{NTC} RD + BC \cdot RC}{RD + RC} \quad (6.12)$$

So, starting from the **OP-AMP**'s inputs equality of $V_+ = V_-$ (V_{BC} is at V_-):

$$\frac{U_{NTC} RD + BC \cdot RC}{RD + RC} = V_{BC} \quad (6.13)$$

As already stated, the hysteresis effect depends on the saturation region. So, considering that the channel is at "OK" state, $BC = 0$ V (negatively saturated), the goal is to determine when the transition for the opposite state occurs, meaning when the **OP-AMP** saturates positively i.e. $V_+ > V_-$, which gives:

$$U_{NTC} \frac{RD}{RD + RC} > V_{BC} \Leftrightarrow U_{NTC} > V_{BC} \frac{RD + RC}{RD} \quad (6.14)$$

Using this condition¹⁵ to rather determine the resistance (through equation 6.2) and then the temperature on which the transition should occur, the result is $R_{eq} > (1.65 \pm 0.04)$ M Ω and for that $T < (-66.5 \pm 0.7)$ °C. Considering that the system is now at "INTERLOCK", $BC = V_{CC} = 3.3$ V (positively saturated), and will transition to "OK", meaning $V_+ < V_-$:

$$\frac{U_{NTC} RD + V_{CC} \cdot RC}{RD + RC} < V_{BC} \Leftrightarrow U_{NTC} < \frac{V_{BC}(RD + RC) - V_{CC} RC}{RD} \quad (6.15)$$

which corresponds to $R_{eq} < (0.88 \pm 0.01)$ M Ω and $T > (-58.5 \pm 0.6)$ °C. For the HI **OP-AMP**, the inputs' equality is similarly given by the conditions of equation 6.12:

$$\frac{V_{HI} RF + HI \cdot RE}{RF + RE} = U_{NTC} \quad (6.16)$$

since V_+ is rather a product of V_{HI} and the hysteresis resistors, RE and RF and $V_- = U_{NTC}$. This equation already gives a straightforward deduction. If the considered transition is from "OK" to "INTERLOCK" state, it transitions to $V_+ > V_-$ while $HI = 0$ V, so $(\frac{V_{HI} RF}{RF + RE}) > U_{NTC}$ ¹⁵, which equals to $R_{eq} < (5.73 \pm 0.01)$ k Ω and $T > (40.17 \pm 0.33)$ °C. Considering the opposite transition, $V_+ < V_-$ while $HI = V_{CC} = 3.3$ V, $(\frac{V_{HI} RF + V_{CC} RE}{RF + RE}) < U_{NTC}$ and $R_{eq} > (5.85 \pm 0.01)$ k Ω and $T < (39.54 \pm 0.33)$ °C.

Each state transition will be addressed as: **OP-AMP** output "bit" $\rightarrow$ "bit"¹⁶. So, to resume, it needs to reach at least (40.17 ± 0.33) °C to achieve the HI "0" $\rightarrow$ "1" transition. However, once in "INTERLOCK", to return to "OK" state (HI "1" $\rightarrow$ "0"), it needs to drop to (39.54 ± 0.33) °C. Otherwise, it will stay in "INTERLOCK" state. For the "Broken Cable" to be triggered by BC "0" $\rightarrow$ "1", the temperature should drop below (-66.5 ± 0.7) °C and BC "1" $\rightarrow$ "0" should only happen above (-58.5 ± 0.6) °C. Also, the R_{eq} and T ranges follow opposites directions. So, to increase the temperature until the threshold, the **NTC** resistance needs to decrease and vice versa.

These conditions create a range of values that the **NTC** needs to reach to occur each state transition,

¹⁵The uncertainty of U_{NTC} for the BC and HI transitions are deduced in Appendix E.2.1 and E.2.2, respectively.

¹⁶For instance, HI "1" $\rightarrow$ "0" or BC "0" $\rightarrow$ "1" are from high temperatures to "OK" state and from "OK" to "INTERLOCK" state transitions for the upper and lower limits, respectively.

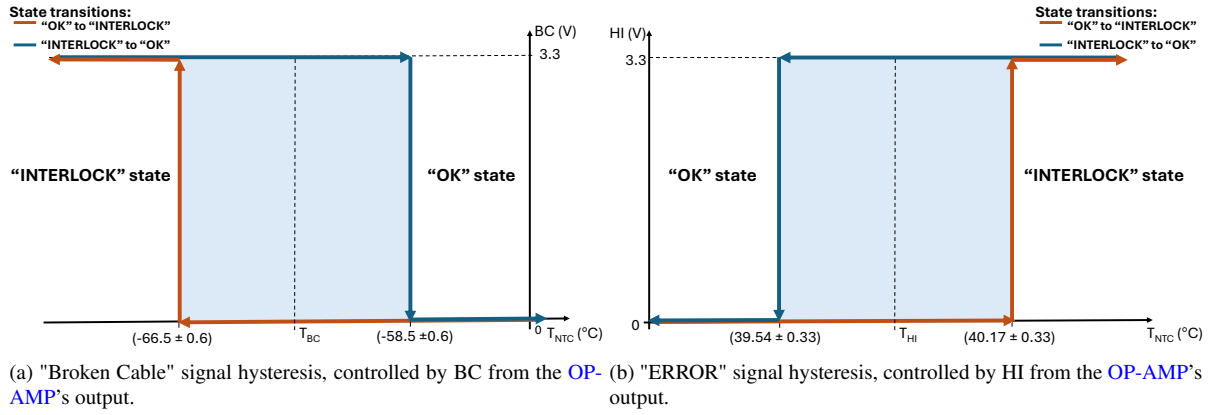


Figure 6.2: Temperature hysteresis graph for the T2I channel's input. The transition from "OK" to "INTERLOCK" state is defined by $T < (-66.5 \pm 0.7) ^\circ\text{C}$ and $T > (40.17 \pm 0.33) ^\circ\text{C}$ for the "Broken Cable" and "ERROR" signals. While the opposite transitions are defined by $T > (-58.5 \pm 0.6) ^\circ\text{C}$ and $T < (39.54 \pm 0.33) ^\circ\text{C}$, respectively. Realistically, the transitions are not instantaneous state changes, but rather occur gradually. The blue zone between transitions represents the *no transition zone*, on which no state transition occurs. The proportions on the graphs axis are not exact and serve merely for demonstrative purposes.

which implies that the transitions no longer occur at the same threshold. This allows to hold a specific state when the received signal suffers from noise fluctuations, safeguarding the system's operation. Illustrations of the hysteresis ranges are represented in Figure 6.2. The hysteresis also causes the "Broken Cable" transitions to occur outside of the limit if $1 ^\circ\text{C}$ of accuracy [42], while the "ERROR" transitions have $+0.17 ^\circ\text{C}$ and $-0.54 ^\circ\text{C}$ deviations from T_{HI} . Given the higher importance of the high temperature limit, it was imperative to ensure this requirement, while for the lower temperatures, which will not be achieved by the NTCs, exists to detect anomalies, not being strictly necessary to comply with them.

6.1.4 The T2I channel's tests procedure

The testing of the T2I card was focused on the individual hysteresis of each channel, by causing the state transitions. Given that a NTC can be simulated by an equivalent resistor, it was built specific circuits of series of potentiometers¹⁷ and resistors (combined, they form R_{eq}). By changing the equivalent resistance with the potentiometers, it was possible to adjust the temperature until a transition was detected. In the end, the temperature over the 12 channels of the card at each transition on both temperature limits was compared. The temperatures were obtained from the ADC readings (converted digital word) gathered from the server (equation 6.6) and from manual measurements of resistance and voltage, with the multimeter in average mode, which stores multiple values and calculates their average. To estimate the temperature, the resistance measurements are directly applied in equation 4.1 and the voltage is first used in equation 6.1 and then in equation 4.1.

Given the estimated R_{eq} for each transition, it was essential to choose the most granular and optimal range for the transition in question, allowing to deal with more precise resistance variations. However, another factor to consider was to choose a set of resistors and potentiometers that could cover the hysteresis *no transition zone* and enable the state transitions in both directions, from "0" $\rightarrow$ "1" and vice versa. This was due to the process of mounting the R_{eq} into the open channel, which causes sudden fluctuations. This is expected given the circuit adjustment to the new component. As stated before, the hysteresis defines a *no transition zone*, on which no transitions occur. When using a series that does not cover this zone, the fluctuations can cause the channel to momentarily switch between unwanted transitions, causing it to stay in the *no transition zone*. If R_{eq} does not have enough range to get out of the *no transition zone*, it will stay at that state. So, by using a series that covers this range and guarantees an outside safety margin, even with the initial oscillations, it would be possible to actually define which

¹⁷Resistors with a variable and controllable resistance.

6. THE PRELIMINARY TESTING OF THE SYSTEM

transition is simulated. The *no transition zone* of the BC hysteresis has $0.77\text{ M}\Omega$, so it was necessary to place four potentiometers and one resistor in series. It was used $R = 820\text{ k}\Omega$ and one potentiometer of $500\text{ k}\Omega$ [4], two of $200\text{ k}\Omega$ [1] and one of $100\text{ k}\Omega$ [3], which gives a variable resistance of $1\text{ M}\Omega$. For the HI transitions, it was used a fixed resistor, $R = 5.6\text{ k}\Omega$ with a potentiometer of $500\text{ }\Omega$ [2] (ranging from $5.6\text{ k}\Omega$ to $6.1\text{ k}\Omega$). All the used fixed resistors have a tolerance of 5% and measurements confirmed it.

The transition detection was accomplished with conditioning circuitry on the OUT and GSS channels. The open GSS channels caused the interlock at the OUT channels, even if the T2I channels were closed. For instance, the E_EXIT signal (GSS input 1) alone causes general interlock to the crate. So, to close the channels, i.e. placing them at "OK" state, as recommended by the ITk team [21], they required a voltage of 3.3 V (at least 2 V) at each input, what was easily achieved by mounting each channel on a breadboard powered by a voltage power supply.

For an easier transition detection, it was implemented LED circuitry at each OUT channel, which reacted accordingly to the signals. The channels' output voltage operates at negative logic, meaning 0 V at the channel's output reflects "INTERLOCK" state, which turns off the LEDs, whereas they remained powered at "OK". In order to limit the current at the LEDs, they were installed in series with $100\text{ }\Omega$ resistors. Each I/O card connector was specifically soldered¹⁸ to cables to allow the mounting of the circuits on breadboards.

Every T2I channel controlled at least 1 OUT channel¹⁹ and this channel mapping was configured by the ILK-FPGA's interlock matrix. The 12 T2I channels were not enough to cover all 20 OUT channels. So, until the 8th T2I channel, they were associated to two OUT channels sequentially. From the 9th to the 12th channel, the correspondence was one T2I to one OUT channel, sequentially. For example, the T2I channel 1 controlled the OUT channels 1 and 2, while the 9th controlled the 17th OUT.

The initial resistance for every transition was established by one of the potentiometer's ends, using it as reference for every channel. For the HI "0"→"1" transition, $R_{eq} = 6.1\text{ k}\Omega$ was the initial value. By decreasing its resistance (the temperature is increased) by steps, it would eventually achieve the desired threshold. Figures 6.3 and 6.4 exemplify this procedure as well as the opposite one. This was recreated at each channel. The HI "1"→"0" transition was initiated at $5.6\text{ k}\Omega$, the BC "0"→"1" and "1"→"0" at $820\text{ k}\Omega$ and $1.82\text{ M}\Omega$, respectively.

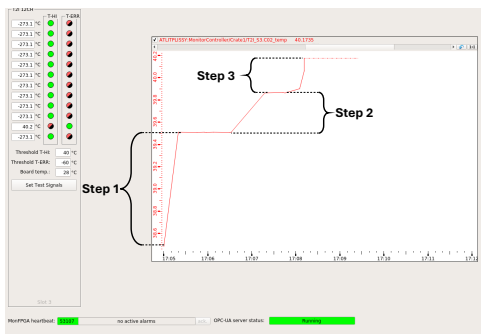


Figure 6.3: HI "0"→"1" resistance variation by steps, reflecting a temperature increase on channel 2 ("T2I_S3_C02_temp").

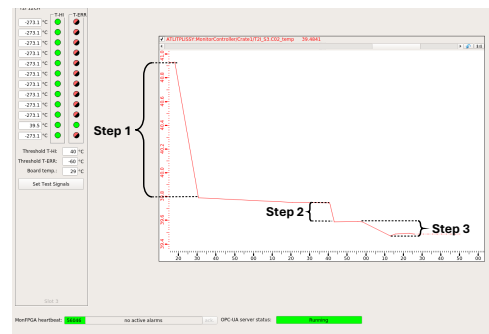


Figure 6.4: HI "1"→"0" resistance variation by steps, reflecting a temperature decrease on channel 2 ("T2I_S3_C02_temp").

So, when the threshold is achieved and observed, the voltage and resistance at the terminals of R_{eq} are measured. On the the monitoring side, the displayed channel's temperature, interlock signal and digital word are registered.

¹⁸Connectors, cables, and protective rubber coverings were used. The components were also used in the Demonstrator.

¹⁹Allowing to test not only the T2I channels but the system's reaction chain and the OUT channels' proper operation.

However, tests were conducted to evaluate the monitoring interface update reliability compared to the OUT LED's reaction, for each transition. For the T2I BC and HI "0"→"1" transitions, the LED circuitry reacted in 3 and 2 seconds, respectively, while the interface took 7 to 9 seconds. Opposite transitions were immediate for the LEDs, but took 7 seconds for the interface. For the GSS interlock signals, the LEDs reacted instantaneously, whereas the interface consistently required 6 seconds to update. Reactions were timed from the abrupt R_{eq} change triggering the transition, at each individual channel, starting the counting simultaneously for both methods. These time reactions can be caused by several factors, such as the human reaction on detecting the transition and communications delays through the local network. But for comparison, these factors do not interfere. The source of the LEDs' reaction delay at the "0"→"1" transitions is unknown and highly unlikely to be caused by the LEDs themselves or by the OUT card's circuitry, which indicates a possible ILK-FPGA firmware implemented delay.

Concluding, at this development stage, the monitoring interface's reaction time is unreliable, leading to the use of the LED circuitry to detect the transitions, which does not depend on server/client data exchange. The monitoring interface will only be use to note the temperatures and ADC readings after the transition is detected through the LEDs.

6.1.4.1 The ADC reading's derived temperatures

The first analysis compares the estimated temperature from the ADC readings (through equation 6.6 and 4.1) taken from each channel with the displayed values on the monitoring panel (Figure 5.7). The goal is to **evaluate the monitoring panel's temperatures legitimacy** and, at the same time, detect any anomaly on them or in the server calculation method²⁰. Figure 6.5 presents the temperature plots of the monitoring panel (WinCC) and the temperatures derived from the ADC readings (ADC), of each channel's detected threshold, for every transition. The WinCC panel's temperature resolution is 0.1 °C. However, for this analysis, the WinCC values' uncertainty is ± 0.05 (error bars), instead of the smallest displayed value on the monitoring panel (the resolution). This is to verify the temperature estimation rounding process, because the WinCC values are rounded at 0.5 °C. So, by comparing both values, it allows to **confirm the WinCC temperatures according to the ADC readings and this rounding process**. The temperatures derived from the ADC readings have a neglected uncertainty. They only serve to evaluate the actual behaviour of the channel. The ADC's values presented are supposed to mirror the temperature calculations happening in the OPC UA server (Section 4.6.2). The ADC readings are the only variable values, while $RA = 10 \text{ k}\Omega$, ADC's digital FSR of $(2^{15} - 1) = 32767$ (6.6), $A = \frac{1}{298.15} - \frac{1}{3435} \times \ln(10000)$, $B = \frac{1}{3435}$ and $C = 0$ (4.1).

For the HI "0"→"1" Figure 6.5 (a), the channel's 2 transition was detected later, with a WinCC value of 40.30 °C, which could be caused by an occasional slow human reaction or an undetected change in the test procedure, such as making a larger step change. However, the temperature derived from the ADC was at its rounding range lower limit, validating the WinCC value, but also possibly indicating an one-case scenario, as its was close to the other channels' tendency of $(40.20 \pm 0.05) \text{ }^\circ\text{C}$. Either way, it did not represent a risk of failure or anomaly, as every ADC value is consistently within the WinCC rounding range. The HI "1"→"0" Figure 6.5 (b) presents no anomaly, since every transition occurred within the rounding range of the WinCC values, and these occur at the theoretical value of $(39.54 \pm 0.33) \text{ }^\circ\text{C}$. Both transitions present **values within the required threshold accuracy** of 1 °C [42].

²⁰The monitoring panel temperatures are estimated with the ADC readings.

6. THE PRELIMINARY TESTING OF THE SYSTEM

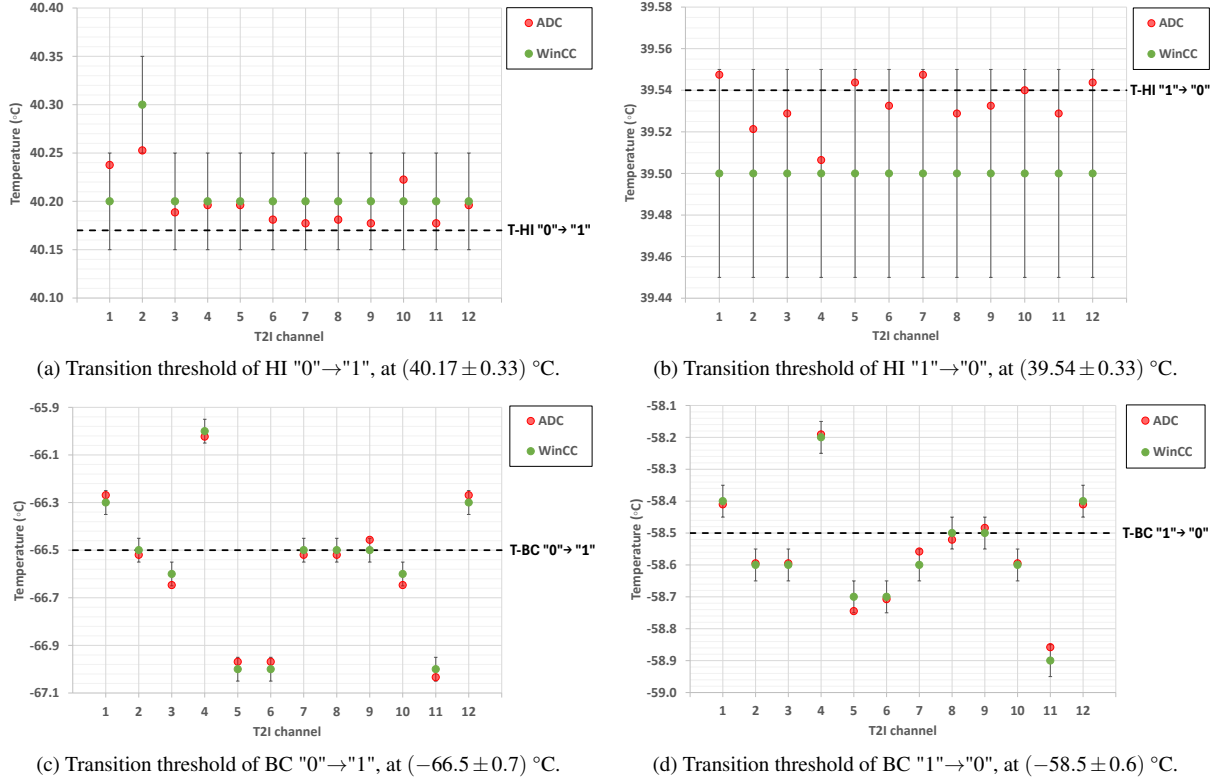


Figure 6.5: Monitoring panel's (WinCC) and ADC readings' (ADC) derived temperatures comparison. For every transition, the ADC derived temperatures were within the WinCC temperatures rounding range ($\pm 0.05 ^\circ\text{C}$), represented as the error bars. All transitions presented values within the respective theoretical value's uncertainties. These uncertainties are not represented as they exceed the temperature axis range.

It is worth reminding that the "0"→"1" transitions presented a **LED** reaction latency (the end of Section 6.1.4), which could caused the detected temperatures above the threshold on Figure 6.5 (a). This can be verified not only by the channel's 2 occurrence mentioned above, but as well as by the **ADC values tendency above the threshold value**, while "1"→"0" Figure 6.5 (b) presents lower deviations from the threshold, as it is detected earlier. Allied to this, the high temperature regime favours large temperature variations with minimal resistance changes²¹, intensifying this type of deviations.

The "**Broken Cable**" transitions, Figures 6.5 (c) and 6.5 (d), present a bigger detection value dispersion, since **variations of the calculation variables** from manufacturer values **have larger impact on these temperatures**²¹. This observation about the dispersion is also supported by the higher standard deviations: $\sigma_{\text{BC "0"→"1"}} = 0.3 ^\circ\text{C}$ and $\sigma_{\text{BC "1"→"0"}} = 0.2 ^\circ\text{C}$, while $\sigma_{\text{HI "0"→"1"}} = 0.03 ^\circ\text{C}$ and $\sigma_{\text{HI "1"→"0"}} = 0.01 ^\circ\text{C}$. However, every temperature derived from the ADC readings were within the rounding range of the WinCC values and both temperatures were within the theoretical uncertainties.

This analysis allows to **validate the WinCC panel's temperature**. The **ADC values will serve as a reference** for the channel's performance.

6.1.4.2 The resistance and voltage measurements' derived temperatures

This analysis focuses in the temperatures obtained from the **direct measurements of the resistance**, R_{eq} , estimated through equation 4.1, and the **voltage**, U_{NTC} , estimated through equations 6.2 and 4.1. The goal is to **evaluate the T2I channels performance over each transition threshold**. With the previous analysis' conclusion, the ADC values can be mostly used to analyse specific temperatures which might break any pattern consistency. Figure 6.6 shows the plots of the temperatures over the channels at

²¹This will be studied in Section 6.1.4.3.

every transition for the WinCC panel, the theoretical threshold values, both measurements and additionally the ADC readings' temperatures. Both WinCC resolution and theoretical values' uncertainties are represented as coloured areas (green for the resolution of ± 0.1 °C²² of the former and gray for the latter). The direct measurements uncertainties used the instrument manufacturer's weighting for *1 Year* $23^{\circ}\text{C} \pm 5^{\circ}\text{C}$ [5], which is weighted according to the measurement reading scale and the readings themselves. The uncertainty of the temperature estimation is shown in Appendix E.1.

Both temperature regimes required specific analyses. On the high temperatures (HI transitions), Figures 6.6 (a) and 6.6 (b), there is no noticeable value dispersion ($\sigma_{\text{HI} "0" \rightarrow "1"} = 0.03$ °C and $\sigma_{\text{HI} "1" \rightarrow "0"} = 0.02$ °C, on both measurements), while on low temperatures (BC), Figures 6.6 (c) and 6.6 (d), the values are less precise ($\sigma_{\text{BC} "0" \rightarrow "1"} = 0.3$ °C and $\sigma_{\text{BC} "1" \rightarrow "0"} = 0.2$ °C, on both measurements). Also, the R_{eq} temperature of channel's 4 BC "0" → "1" Figure 6.6 (c) is the only measurement apart of the theoretical uncertainty, despite its own uncertainty covering the deviation. This is once again explained by the higher error propagation at low temperatures²¹. The low temperature regime (Figures 6.6 (c) and 6.6 (d)) also impacted both measurements with larger deviations from the WinCC values, not fitting in the ± 0.1 °C resolution, although their uncertainties certainly cover the deviations and their values follow the WinCC values tendency.

For the HI transitions, the past event (Section 6.1.4.1) of channel 2 (HI "0" → "1") Figure 6.5 (a) was confirmed by the temperatures derived from the measurements of Figure 6.6 (a), as there was a similar difference when compared with the other channels. The event of the registered temperatures above the thresholds in Figure 6.5 (a) are as well confirmed by the Figure 6.6 (a) results. However, the measurements' uncertainties do not allow to guarantee such assumptions. The theoretical value's uncertainty of ± 0.33 °C, covered every estimated temperature (**below the accuracy of 1 °C** [42]). The panel's temperature resolution covered the measurement's deviations. Both measurement's uncertainties at the two transitions (Figures 6.6 (a) and 6.6 (b)) also were ± 0.3 °C, which could obviously, in return, cover every deviation from the WinCC values and errors from the theoretical values. This shows the **value's precision** and also their **accuracy at this temperature regime**, which is the crucial transition limit.

At these temperatures, Figures 6.6 (a) and 6.6 (b), every transition clearly registers a trend correlation of both measurements, as they mirror each other in most of the cases. They present no significant deviation from each other and the uncertainties cover that deviation. Although any deviation from each other could be explained by error propagation, since the U_{NTC} has additional calculation steps, while the direct measurement of R_{eq} already covers those error sources. Their negligible deviations give them, individually (by channel) high precision level.

Despite every measurement follows the ADC pattern, some cases show slight deviations from the ADC readings, but given the large uncertainty ranges of both measurements, is not possible to achieve a definitive conclusion with high confidence. Although they are worth mentioning, such as both channel's 11 HI transitions (Figures 6.6 (a) and 6.6 (b)) having higher measurement's derived temperature and channel's 10 BC "1" → "0" Figure 6.6 (d) which registered a higher U_{NTC} temperature.

All transition values are essentially **within the theoretical value range**. It is worth mentioning that the HI transitions (Figures 6.6 (a) and 6.6 (b)) were given more confidence on the estimated theoretical value, being associated with a two significant figures' uncertainty (± 0.33 °C). If the considered value were to be ± 0.3 °C, the information above the second figure was lost. In the case of the HI "1" → "0", (b) its value would be placed at 39.5 °C, which would contradict all the other values' tendencies with regards

²²The objective is not to verify the rounding process, so the ± 0.05 °C uncertainty will no longer be used.

6. THE PRELIMINARY TESTING OF THE SYSTEM

to the deviation from the expected threshold. In BC transitions (Figures 6.6 (c) and 6.6 (d)), the obtained values have larger deviations which would not be corrected by a more confident theoretical value.

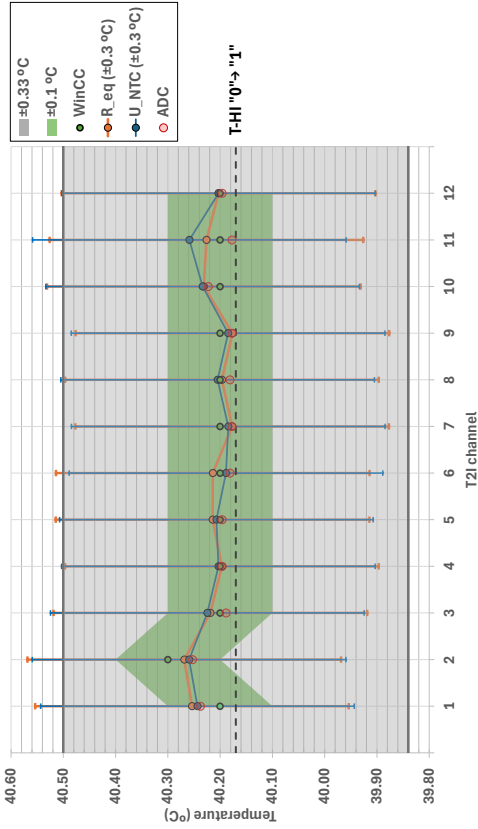
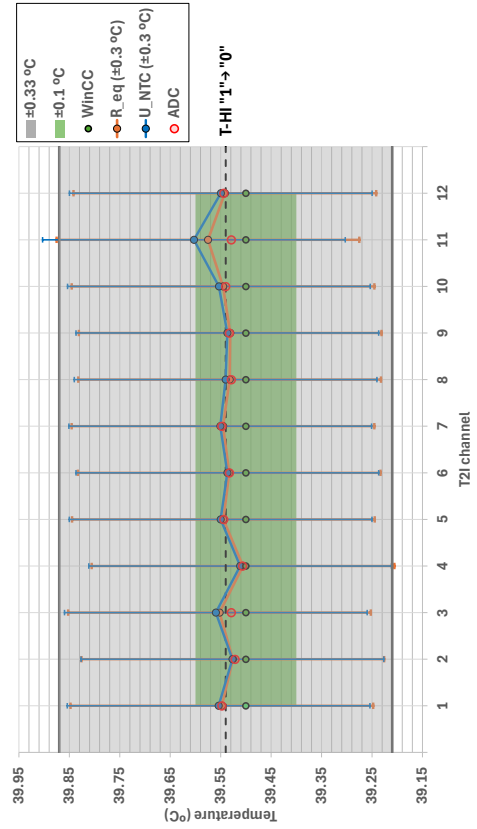
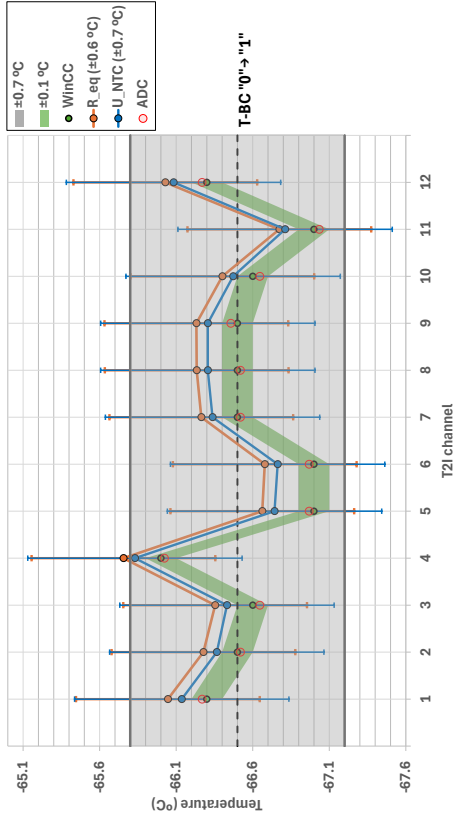
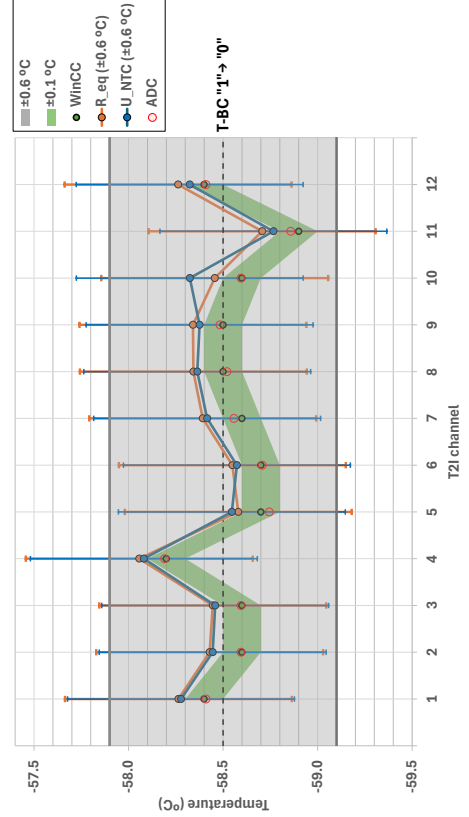
(a) Transition threshold of HI "0"→"1", at $(40.17 \pm 0.33) ^\circ\text{C}$.(b) Transition threshold of HI "1"→"0", at $(39.54 \pm 0.33) ^\circ\text{C}$.(c) Transition threshold of BC "0"→"1", at $(-66.5 \pm 0.7) ^\circ\text{C}$.(d) Transition threshold of BC "1"→"0", at $(-58.5 \pm 0.6) ^\circ\text{C}$.

Figure 6.6: Monitoring panel's (WinCC) and measurements' (R_{NTC} and U_{NTC}) derived temperatures comparison. The WinCC values of the HI transitions (upper side figures) covered every measurement's derived temperature, while the BC (lower side figures) measurement's values present slight deviations from the $\pm 0.1 ^\circ\text{C}$ of the WinCC values, although they are covered by value's uncertainty. The green area represents the monitoring panel values' resolution and the gray area the respective theoretical uncertainties.

6. THE PRELIMINARY TESTING OF THE SYSTEM

6.1.4.3 The temperature regimes

Given the clear different behaviour of the temperature values at different temperature regimes, it was essential to evaluate both regimes. **To analyse the error impact at different temperature regimes**, it was necessary to **study the temperature function**. Using equation 4.1 as a model for the temperature and resistance relation it is possible to obtain different behaviours on the resulting curve. In order to validate such model, it was conducted successive resistance measurements in progressively larger resistors, while verifying the WinCC temperatures at each resistor. For this demonstration, the channel 2 was randomly chosen and presented in Figure 6.7. The resistance measurements include the instrument's uncertainty.

Alternatively, each resistor's equivalent temperature was estimated with equation 4.1. Equation E.4 was used to calculate these temperatures' uncertainty, which never exceeded the order of 0.1 °C. The estimated temperatures' deviations from the WinCC values are in the order of 0.01 °C and 0.1 °C for larger resistances. The estimated temperature's uncertainty cover the WinCC values. This allows to **validate the model for the following analysis**. The equation 4.1 model's curve was not represented as it overlaps the WinCC temperature plot, given the large resistance and temperature ranges, as well as the estimated temperature uncertainties, which are not visible.

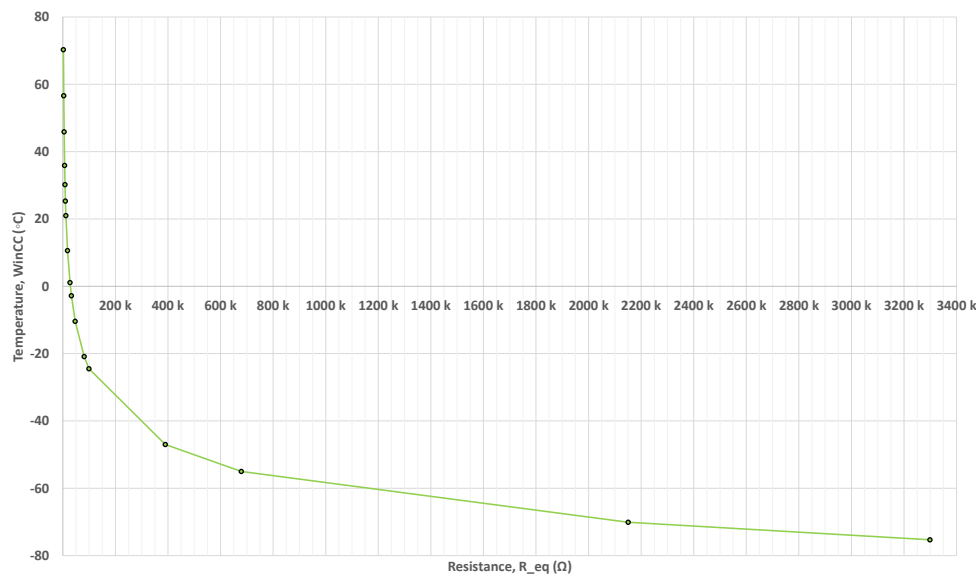


Figure 6.7: Monitoring panel's temperature (WinCC) as a function of the measured resistances (R_{eq}) for the T2I channel 2. The resistors used range from 2.2 kΩ to 3.3 MΩ.

With this, it is possible to verify that the temperature has an exponential decrease with the increase of the resistance. At first hand, at lower resistances, the **temperature changes drastically**, while at higher resistances has a **reduced slope**. This means that at high temperatures (HI transitions¹⁶), a small resistance increment can mean significant temperature changes. Despite the large uncertainty of the measurement's values compared to the deviations, this effect can easily intensify the LED's latency (Section 6.1.4) impact at the HI "0"→"1", presenting results above the threshold (Figures 6.5 (a) and 6.6 (a)).

However, it was possible to achieve better results at higher temperatures. One possible cause could be the resistance step change. At higher temperatures, although it changes excessively, the hysteresis *no transition* range does not exceed 130 Ω, allowing to have only one and more granular potentiometer (500 Ω) and providing a more precise and small step change, while at low temperatures (BC transitions¹⁶) it reaches a 770 kΩ range. For this range it was not feasible to use such granular and small potentiometers

(the smaller potentiometer has 100 k Ω), allowing to easily overstep the established threshold. However, individual tests have shown that even the small resistance overstep at both BC transitions should have not caused such deviations and dispersion, given the curve stability at this regime. Just for reference, for the BC "0" $\rightarrow$ "1" Figure 6.6 (c), the larger deviation from the threshold is 0.8 $^{\circ}\text{C}$, at channel 4. At low temperatures, this can equate to around 102 k Ω , while the smallest potentiometer closely matches that value.

The most possible cause, not only to the low temperatures larger **measurement's values deviations from the ADC readings and the WinCC values** (larger than ± 0.1 $^{\circ}\text{C}$), but also to the **all values tendency** (including the WinCC and ADC readings) **dispersion and deviations from the theoretical values**, is the error from the manufacturer's component values used in the temperature estimations and the real values, such as the reference resistors for the hysteresis values (Section 6.1) and RA (Section 6.1.1). The larger estimated uncertainty for either measurement's values (Figures 6.6 (c) and 6.6 (d)), compared to the HI temperatures (Figures 6.6 (a) and 6.6 (b)), confirm this assumption.

To emphasize this statement, every temperature was deduced from calculations. Even the ADC readings' temperatures are simply voltage readings. Their temperature estimations generalize every channel's behaviour. Yet, every single channel has individual circuitry, which could imply discrepancies in the individual performance that remains uncompensated. This could explain the case of channel's 4 transition, seen in Figures 6.6 (c) and 6.6 (d), drifting significantly compared to channel 8 for instance, even if both channel's uncertainties cover any deviation. Nevertheless, among all estimated values, the ADC readings remain as the most reliable. Despite these occurrences, both BC transitions (Figures 6.5 (c) and (d), Figures 6.6 (c) and (d)) follow a consistent tendency due to the temperature's stability in this regime. Consequently, while **the values present larger deviations, they propagate consistently**. It should be also noted and rephrased that the temperatures derived from the measurement's derived temperatures uncertainties also covered every deviation justified by these assumptions.

Although that every **deviation is covered by the direct measurement's uncertainties** (R_{NTC} and U_{NTC}) and that the **threshold accuracy of 1 $^{\circ}\text{C}$ was confirmed**, at least in the high temperatures (HI transitions) it was considered necessary to attempt to justify the channel's behaviours. With this, it can be concluded that the T2I board is operating correctly and prepared for the Demonstrator implementation (Chapter 7)

6.2 The GSS and OUT boards

The GSS card was easily tested by forcing the interlock signal by cutting the inputs' voltage (3.3 V at the card's inputs mean "OK" state and 0 V induce "Interlock" state). According to the interlock matrix established by the ILK-FPGA, it correctly powered off the corresponding OUT LEDs. The OUT card was tested by other developers of the local HIS and were operating properly. The tests were mostly based on the current measurement at the output of each channel to verify the correct operation of the current limiting circuit, as introduced in Section 3.1.2.4.

6.3 The T2I test signals

The testing of the I/O cards is one of the Mon-FPGA requirements, through the transmission of test signals, requested in the monitoring panel. In Figure 6.1 it is possible to identify a diode, D [60], on the HI (high temperatures) OP-AMP, which drives into the V_+ input, where the reference voltage, V_{HI} , is introduced.

As explained in Section 3.1.2.2, both T2I and GSS test signals correspond to bit bus commands (16-bit and 10-bit, respectively) sent by the Mon-FPGA, when requested by the DCS. Each bit from the bus

6. THE PRELIMINARY TESTING OF THE SYSTEM

corresponds to a card's channel. When it is "1" (3.3 V), it should trigger an interlock signal by activating the test signal. When at "0" (0 V), it does not interfere with the channel's operation. This bit is then applied in the diode's anode, while at the cathode is the V_+ voltage, given by the left side of the equation 6.16, which varies between (0.9110 ± 0.0008) and (0.9238 ± 0.0009) V (E.2.2). So, by injecting 3.3 V at the diode's anode, the $V_{anode} > V_{cathode}$ ²³ causes the component to drive current.

The resulting voltage at the cathode is given by a characteristic voltage drop on the anode's voltage. According to the diode's manufacturer, at the forward region the typical voltage drop is around 0.65 V, giving a $V_{cathode} = 2.65$ V. This forces $V_+ = 2.65$ V. Since the maximum voltage possible at the U_{NTC} is (2.5024 ± 0.0001) when in open loop channel, it will never exceed the new V_+ and OP-AMP will saturate positively, regardless of the U_{NTC} . This causes HI = 3.3 V ("ERROR" signal) and "INTERLOCK" state at that channel. The only way for this channel to return to "OK" state, is to deactivate the test signal, sending the bit "0" into the diode's anode, stopping the current flow through it. As stated before, the test signals of each channel are activated through specific test panels on the implementation panel (Figure 5.7).

The test procedure was to activate a test signal of a channel at "OK" state, **verifying its successful activation** and the **chain reaction**, causing the **"ERROR" signal at that channel**, leading to the respective **OUT channel (LED) to shut down**.

The testing of the test signals verified two distinct situations. From **channel 1 to 8**, the test signal was activated accordingly, sending a "ERROR" signal to the respective OUT channel, shutting it down. However, the deactivations did not operate properly. Most of the times, without a pattern, after being deactivated, the channel being tested would still be at "INTERLOCK" state. A common way to "unblock" such situation was to activate another channel's test signal, which would liberate the past channel from the "INTERLOCK" state. Although the newly activated channel would usually face the same situation. Sometimes a random activation of multiple signals would release all channels from that situation, but mostly a general normalization would only occur after restarting the entire system.

This complication was discussed with the **ITk Mon-FPGA** team, which explained that this is a known behaviour and it needs correction, but for the moment, would have to operate this way.

The higher channels, **from 9 to 12**, would present a completely different situation, where the test signal was activated by the **Mon-FPGA**, but none of the channels reacted to it. This could mean that the 3.3 V is not properly set at the diode's anode during the activation. According to the **ITk** team, these channel's test signals are controlled by a different IO expander, which could explain the different behaviour. However, the team have not registered this problem with their **Mon-FPGA** card models, which are newer and use different and dedicated firmware. This could possibly mean a sort of incompatibility of the firmware, which is not yet fully developed to carry these channel's tests.

To verify such behaviour, the **ITk** team recommended the same testing on a T2I-16 card, solely provided for this test, at **CERN**, during the Demonstrator implementation. The reason was to verify if the channels from 9 to 16 (controlled by another expander) would react to the signal's activations. So, after the temperature controlled tests (Section 7.1) and before the final **LISSY** installation (Section 7.2), the provided T2I-16 was installed (still with the T2I-12 and OUT-20), while the monitoring chain and interlock matrix were updated to carry the card addition. The aforementioned tests were repeated on the T2I-12 and then on the T2I-16 with no successful results. Both situations were still observed: channels from 1 to 8 blocked after the deactivation and channels above 9 would rarely react to the signal tests. With this being said, there was not an available

²³Designated as forward bias condition, which allows the diode to flow current from the anode to the cathode.

solution to the problem at the time of this thesis and **the T2I test signals were given as unreliable** for the moment.

6.4 The GSS test signals

The **Mon-FPGA** test signals for the **GSS** inputs operate the same way, with an **I²C** bus, but are applied to OR gates, which outputs "1" when the test signal is "1". The tests **evaluated the chain reaction** of the test signal activation and **its return to "OK", after deactivating the test signal**.

The **GSS** test signals also faced complications. When activating the channel's test signals, the input would send a "ERROR" signal and the respective OUT reacted accordingly. However when deactivating, the whole system would malfunction. Both **T2I** and **GSS** channels would randomly switch between interlock states, causing the system to repetitively shut down the OUT channels. This would only be corrected by restarting of the system. The **ITk** team pointed out this situation as an unfinished firmware [72] and so these tests signals can not be used.

It is worth mentioning that at the moment, the **Mon-FPGA** firmware is still in development, thus some blocks do not yet function properly. Although, for the next phase, the Demonstrator implementation, as it is a more elementary system's installation, focused on the temperature monitoring and safety, both cards, the **T2I** and **GSS**, were considered reliable for it. However, the **Mon-FPGA** test signals of these cards were not required for this stage of **HIS** development.

7 The Demonstrator implementation

This chapter will deepen the [LISSY](#) crate installation at [CERN](#), for a realistic implementation, using a future [HGTD PEB](#) board, the Demonstrator. This implementation will focus and ensure the [PEB](#) temperature safety, as an addition for the Demonstrator operation, which will be used to test other [HGTD](#)'s elements apart from the [HIS](#). Before that and in addition to the previous tests, it was also required a preliminary real-time temperature testing, using temperature sensors in a temperature controlled environment, only possible to accomplish at the [CERN](#)'s laboratory. These tests will **confirm the required NTC and ADC** temperature conversion and estimation method **accuracy** of 1% and 0.5 °C, respectively [42].

Moreover, the ongoing preparations for the [HGTD](#)'s [Final Design Review \(FDR\)](#) [22] (the document which *reviews all the available data from prototypes to determine how well the design and the implementation of the design, meet the specifications*), initiated discussions in the [HGTD](#) group about a new proposed [HGTD LISSY](#) layout, which would modify the current strategy for the [HIS](#). Given this fact, it was necessary to review the past strategy to handle this issue. This process will also be addressed in this chapter.

7.1 The real-time temperature tests

The Demonstrator implementation at [CERN](#) required a previous preparation, which included the centralization of the WinCC monitoring project. The local tests used the CentOS7 system, while the [CERN](#)'s machines were running the Linux Alma9 OS, which has installed the WinCC 3.19 and not the local monitoring project version, the WinCC 3.16. As already mentioned in Section 5.2.2, the [JCOPI](#) components are strongly linked to the software version, what needs the installation of the appropriated versions [37]. Also, small adjustments were made in the panels presentation, such as the panel's dimensions, caused by the migration to the newer WinCC version. The database configuration, described in Section 5.2.4.3, was executed only in this updated [CERN](#)'s WinCC project. This project can be found in the centralized gitlab repository [11]. The [Mon-FPGA](#) and [PYNQ](#) configurations are also in a gitlab repository [12].

Since the previous successful temperature tests were done with temperature simulating, it was necessary to perform specific tests for real-time temperature variations, with [NTCs](#). Only after this step, the system could be set up in its dedicated rack (Section 7.2). However, to evaluate the system's operation, the [LISSY](#) was firstly installed in that rack. This step was useful to check the [Mon-FPGA](#) correct configuration and the [ILK-FPGA](#) and [PYNQ](#) board connections to the network, by establishing communication through the [CERN](#)'s machine and the WinCC project. It also allowed the familiarization with the setup, and aided in the preparations for the final implementation. The [PYNQ](#) connection to the network was dealt with in advance, by requesting network access, similarly to the process explained in Section 4.6.2.

Temperature's in and out-of-limit situations were also simulated through fixed resistors. For the high temperatures, it was chosen $120\Omega \pm 1\%$, which along with the uncertainties of the other temperature estimation variables gives a $\pm 3.2\text{ °C}$ uncertainty¹ (Appendix E.4). The WinCC panel temperatures were within this range. For the "OK" state, $10\text{ k}\Omega \pm 5\%$ resistors were used, with their uncertainties alone

¹Unlike in the previous tests, the fixed resistor uncertainty was 1%, and not due to the instrument's measurement accuracy, causing a larger error propagation.

7. THE DEMONSTRATOR IMPLEMENTATION

leading to a range of [23.74, 26.33] °C. The registered temperatures ranged from 24.9 °C to 25.2 °C. The "Broken Cable" was achieved with open channels, and no anomalies were registered. The GSS interlock signals were simulated with the previous explained method (Section 6.2). The OUT channels were driving LEDs as detailed in Section 6.1.4.

Upon this preliminary implementation, the LISSY was set up in a laboratory which held a climate chamber, where the real-time temperature tests were performed.

7.1.1 NTC board tester

To perform the real-time temperature tests it was necessary to assemble real temperature sensors, and to set a most realistic scenario it was used the selected HGTD's NTC from Semitec [45], which have variable accuracy, being the worst-case scenario ± 0.5 °C [44].

For NTCs mounting, 4 PCB boards were designed, each one with 4 NTCs and 1 Sub-D9 male connector for the 3 T2I female connectors. However, they were used with cables and were not directly connected to the T2I connectors. One of the NTC boards had also installed secondary temperature sensors. Let this board be called the *active board*, while the others are the *passive boards*. The sensors had an accuracy of ± 0.09 °C from -25 °C to 85 °C, which is better than the NTCs selected, allowing to evaluate their temperatures and accuracy through comparisons with more confidence. These sensors were installed near the NTCs in a thermal pad, to get temperature homogeneity among every sensor. This defined a better temperature for the test environment, not relying entirely on the NTCs and the environment thermostat belonging to the climate chamber. This was essential given the uncertainty of the climate chamber dimensions compared to the boards' sizes at the time, providing an extra guarantee for reference of the environment temperature. The sensors' temperatures were transmitted to a machine's USB port and processed by a Python script. This script would print and store the temperatures of each sensor and the date-time when they were taken. The boards were designed and developed by HGTD's LIP team and the script was partially written by them and by the local HIS developers.

7.1.2 The climate chamber testing

In order to evaluate the NTCs and HIS operation under real-time temperature variations, it was necessary to place them under a controlled temperature environment. For that, a climate chamber [83], installed in a CERN laboratory, was used. It had a volume of 16 L, which when compared to the NTC boards volume, with about $4 \text{ cm} \times 10 \text{ cm} \times 2 \text{ cm}^2$, represents a large ratio. This discrepancy was expected to affect the results. The NTCs had to be introduced through an isolated side entrance by cabling, including the active board USB cable. The testing setup is similar to the one described in Section 6.1.4. There was one machine to run the monitoring panel and another to read the active board sensors. The **active board was connected to the last T2I 4 channels (9 to 12)**, while the passive boards were connected to the other channels, providing additional data.

After having done several attempts to get accustomed with the climate chamber, which required a slow cool down to the room temperature before opening it, it was verified that the temperature programmed in the chamber's thermostat was rarely achieved by both the NTCs and the active board's sensors. Both sensor's would also take a long time to approximate to the established temperatures. When increasing the temperature, the board sensors' values were always below the established ones. This was possibly caused, as expected, by the large volume of the chamber and the lack of uniformity of temperature inside it. The acclimatization of the chamber was achieved by venting heated air from the back. The boards dependency on the connectors cables would usually force them into positions that were believed

²Width x length x height.

to avoid convection currents, given the board's reduced weight compared to the cables' tension, such as being flat horizontally on the chamber's ground, avoiding most of the temperature convection. The chamber's own sensor could also be placed in a highly temperature-reactive area, such as the chamber's top, where heated air tends to go, or in the proximity of the air vent, registering high temperatures faster. This would be a result from the chamber's temperature heterogeneity. However, this information was unknown.

The boards' positions seemed to be vital for the correct temperature measurement to the point that different board's NTCs registered relatively different temperatures. In order to guaranteed the best positioning possible, allowing to close the temperature gap and homogeneity between boards, various iterations were needed, by ramping a temperature on the chamber thermostat and verifying the stabilized NTCs' and sensors' temperatures. The optimal positioning was to place them vertically, with both sensors facing the back air vent. Despite all efforts, both board's sensors never reached the chamber's temperature while it was increasing, and still were present discrepancies between NTCs of different boards. In the meantime, the NTCs and the active board's sensors temperatures were close enough to ensure a gross temperature comparison and validation for the procedure above. So, the mounting of secondary sensors proved to be essential for temperature analysis, leaving the chamber's temperature out of the NTCs' accuracy analysis what would decrease the confidence in the measurements.

The planned tests **intended to visualize the temperature variation monitored by the Mon-FPGA and the system's reaction when in the interlock threshold**. The low temperatures were not analysed since the established threshold (T_{BC}) is dedicated to the open-channels situation and not to the NTCs. Several test runs were made for different purposes. The main test run in this section was done to register several temperatures from the climate chamber's sensor and both board's temperatures, the interlock signals and the LED's reaction, by ramping the chamber's thermostat temperature by 2 °C steps from 35 °C to 39 °C and then to 42 °C. The last step was larger to ensure that both temperature sensors would actually achieve the HI "0"→"1" threshold (Section 6.1.3), since they have a temperature delay from the chamber's temperature. Then, the ramp-down consists in steps of −2 °C from 39 °C to 35 °C. This allowed to observe each state transition at these temperatures. Each step included a stabilization time for the chamber's and both board's sensors temperatures. Given that the WinCC database was not created, the monitoring panel's temperatures were registered through screenshot date time taken before the next step ramping, which were then compared to the active board's sensors date time.

Figure 7.1 displays the continuous temperature data from the active board's sensors. The goal is to **evaluate the active board's sensors and the chamber's behaviour**. The curves of the four sensors are represented, although they overlap each other given their proximity. Their standard deviation per sample can vary from 0.005 °C to 0.03 °C, with an average of 0.02 °C. These are within the sensor's ±0.09 °C tolerance. However, the temperatures during T_1 , T_2 , T_3 , T_4 , T_5 , T_6 and T_7 will be the focus of the analysis. The sampling period was 2 seconds and the test run lasted about 2:15 hours.

As it can be verified, when increasing the temperature the sensors rarely achieved the thermostat temperature set (red straight lines), while at the descent they are able to follow the temperature steps. This is expected since, according to the 2nd thermodynamic law, to heat a body, it is necessary to provide external energy and this process usually involves inefficiencies due to heat losses. This does not occur naturally. So, it is more demanding to increase the temperature of the sensors, than to decrease it. Due to this property, which adds to the chamber's temperature decrease, the sensors will easily reach the thermostat temperature set, also explaining the rapid temperature decrease. They not only reach the thermostat and chamber's temperature, but they do it faster.

7. THE DEMONSTRATOR IMPLEMENTATION

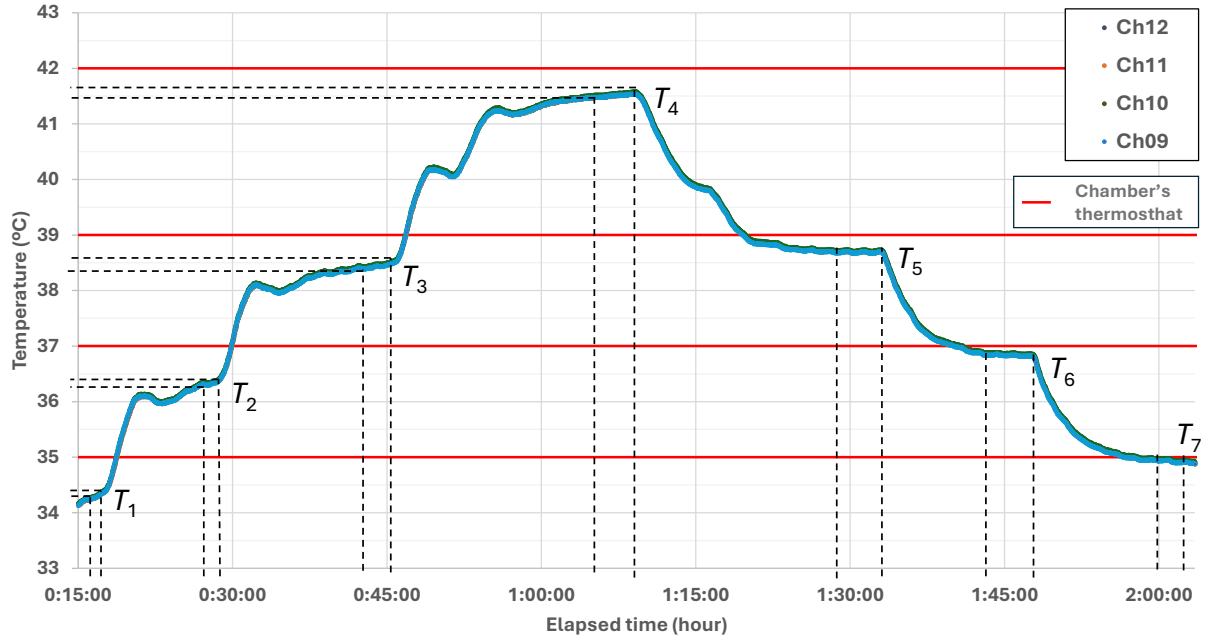


Figure 7.1: Active board's sensors temperature readings. The elapsed time presented starts 15 minutes after heating starts at the chamber. The Ch09, Ch10, Ch11 and Ch12 sensor's curves overlap each other. The red straight lines signalize the chamber's thermostat temperatures at a given T_N step. The climate's own temperature was always the same as the thermostat, with the exception of T_6 . The elapsed time is from the start of the sensor's reading, when defining the thermostat temperature of 35 °C. The reading period was 2 s.

The chamber's temperature sensor registered 36.9 °C at T_6 , which is below the thermostat temperature set. This explains why the active board sensor's temperatures are also lower. The sensor's lower value at T_5 results from the larger temperature ramp-down, causing it to deviate further from the chamber's value. The sensor's reading started with the chamber in cold, taking about 15 minutes to achieve a stable T_1 . It is also present an overshoot on the temperature increase steps. This was previously observed on the chamber's temperature, and it is possibly a consequence of the chamber's method of temperature increase. It overshoots the temperature at first, to reach the thermostat temperature faster and then to stabilize. It was already explained how a body or environment is able to loose heat more easily, than to increase it. So, by boosting the temperature excessively, it will naturally decrease and at the established reference, it is easier to stabilize it.

With this said, the sensor's temperature followed the chamber's temperature behaviour. The difference lies in the sensors' response, which due to a delay of approximately 0.5 °C, continues to register a slight temperature increase even after the chamber's overshoot, whereas the chamber itself stabilizes by letting the temperature drop.

Unlike Figure 7.1, which displays a temperature curve along time, Figure 7.2 intends to **display channel's values registered in specific instances of that test run and evaluate the results under the required accuracies**. Each discretized instance was taken in the aforementioned T_N steps. It is possible to state that every channel state transition occurred at the correct temperatures, as predicted in the Section 6.1.4 tests. The OUT channels' LEDs triggered each transition correctly. However, due to the unavoidable chamber's environment temperature heterogeneity, not all channels transitioned at the same exact instance. Despite this statement, the range of temperatures does not allow a clear discrimination of each channel's curves on the plot. So, only the T2I channel 9 values were represented in the plot, since the 3 did not add new information. It is important to recall that the "OK" to "INTERLOCK" transition should occur at (40.17 ± 0.05) °C, while the opposite one at (39.54 ± 0.33) °C. This is at high temperature regime (T_{HI}). The transition values are represented in the plot as illustrative data, only

7.1 The real-time temperature tests

indicating the temperature at each transition occur. The time on which transition occur was not acquired, although it obviously happened between steps $T_3 - T_4$ and $T_4 - T_5$, respectively. Even though there is no continuous monitoring of data, the sensors' and NTCs' values presented an approximated, almost similar behaviour.

As already proven, the ADC readings' temperatures are within the 0.05 °C rounding range of the panel's values (Section 6.1.4.1). This situation occurred once more throughout this test. So, the plot displays only the ADC monitoring estimated temperatures. This is done to **validate the required ADC's accuracy of 0.5 °C** [42] when estimating the temperatures, while also **evaluating the requirement of a NTC accuracy of 1%**²⁰ [42], on a -40 °C to 40 °C range, when compared to the active board's sensors. In addition, this evaluation accounts for the NTC's manufacturer accuracy of ± 0.5 °C [44], which is higher than 1%.

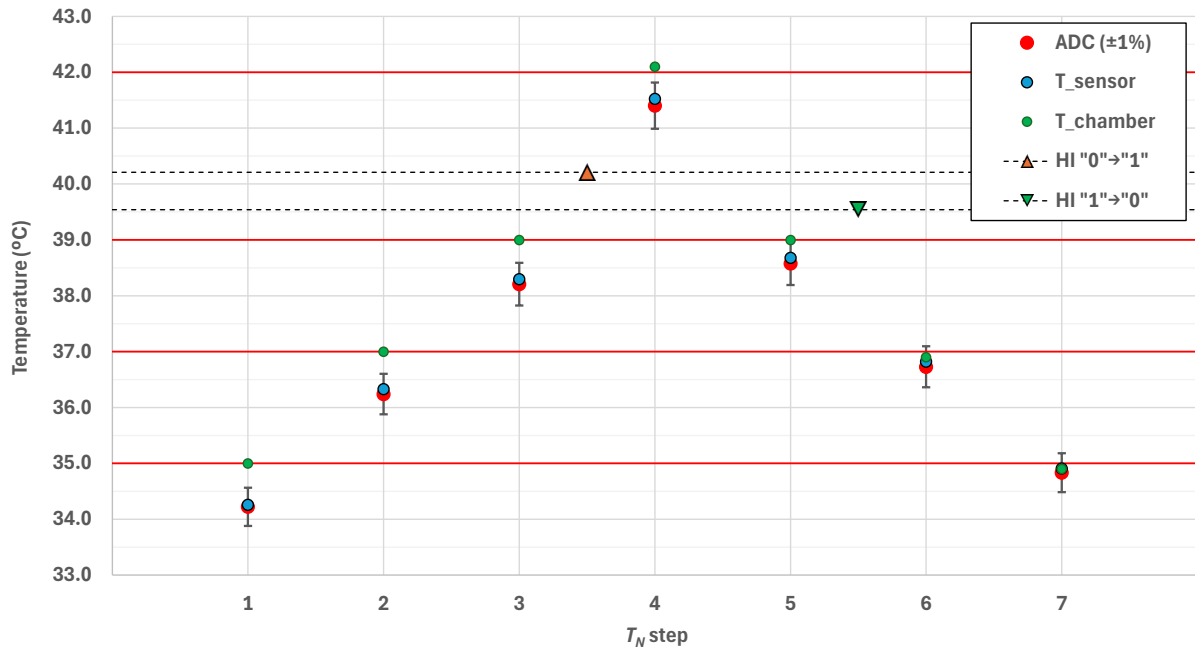


Figure 7.2: Ch09 test run of the NTC's (ADC) and the respective active board sensor's (T_{sensor}) temperatures at each T_N step. The state transitions are represented between the T steps it happened. They are represented as reference, so their x-axis values are illustrative. The error bars reflect the 1% uncertainty of the ADC values.

Since the NTCs temperature is estimated from the ADC's conversion²⁰, the accuracy of 1% in the ADC reading (the error bars in the plot) covers the readings of the active board's sensors, displayed in Figure 7.2. In the tested temperature range, from 34 °C to 42 °C, the 1% weight implies an accuracy of ± 0.34 °C to ± 0.42 °C. This also confirms the required ADC's temperature conversion and the NTC's manufacturer accuracy of 0.5 °C. The values still stayed within this ± 0.5 °C accuracy range, which reflects the high confidence on the WinCC temperatures (reflected by the ADC values). The sensor's deviations from the chamber's temperatures are within -0.7 °C and $+0.01$ °C (at T_7), while for the NTCs are from -0.8 °C to -0.1 °C. These results allow to conclude **the system correct operation** when facing real-time temperature variations and the **functionality of the monitoring interface**. Also, confirm that **the required NTCs accuracy** was reached and that the **proper ADC's conversion and temperature estimation** was done under the **required accuracies and manufacturers parameters**.

Also, before the final implementation, an overnight monitoring was performed with three NTC boards connected to all T2I channels, within a room temperature environment. This allowed to observe, for the first time, the prolonged and uninterrupted behaviour of the system under stress to all channels

7. THE DEMONSTRATOR IMPLEMENTATION

connected with real temperature sensors. The monitoring was done by accessing the monitoring panel several times through the night and ascertain the system's status. The system presented normal temperature values, between 23.5 °C and 23.8 °C and all OUT channels were at "OK" state, what confirmed the system's good performance.

7.2 The Demonstrator PEB

The success of the real-time temperature and the overnight stress tests provided the confidence to proceed to the final implementation, the Demonstrator PEB. As detailed in Section 3.1.2.3, the PEBs are the electronic bridge between the outside of the disk and the NTCs. Essentially, the sensor's data is transmitted to the LISSY crate by the PEB boards. The Demonstrator is the designation of a PEB 1F board (Figure 3.3). Its goal is to provide a preliminary implementation for subsequent and more complex systems. The layout it requires is relatively simple, without involving the MIC. The provided T2I-12, GSS and OUT-20 I/O cards fulfil the Demonstrator layout requirements, as it only demands 6 NTC connections to the T2I. This is achieved with 6 T2I channels. The GSS card is secondary on this implementation, as it is focused on temperature monitoring of the PEB.

The OUT channels should control one HV, one LV Stage 1 and one LV Stage 2 PS³. So, the interlock matrix, programmed by the ILK-FPGA developer, defined three preferred OUT channels for this purpose (channels 1, 4 and 7). The external connection from the OUT card connector to the respective PS is achieved through an adaptor box, which is an enclosure for electronics, mainly consisting of connections, protecting them and simplifying the setup. In this case, it distributes the three chosen OUT channels over the PS specific connectors. **One side** of the box carries a female Sub-D15 connector to **receive each OUT channel** from one of the two Sub-D15 connectors of the card. On the **other side**, it has **one Sub-D15 male connector, to receive the corresponding LV Stage 1 OUT channel** and **two Lemo female connectors to receive**, each one, **one OUT channel for the HV and the LV Stage 2**. The Lemo and LV Stage 1 connectors required GND connections, provided by the GND pins of the OUT card connector. The OUT card to box connection uses a multifilar cable with Sub-D15 males on both ends. Figures 7.3 and 7.4 show a diagram of the adapter box with its connections and the adapter box after soldering, respectively.

The PEB NTCs to T2I channels connection use standard cabling, with a Sub-D9 male connector on one of the ends for the T2I card connector. However, the PEB was still in development and so the installed setup was designed to allow a flexible and straightforward connection for the time when the implementation is ready to set up.

The final setup includes the installed LISSY crate with one T2I-12, one GSS and one OUT-20. The NTC boards were left connected to the T2I board connectors and the OUT channels were left connected to the adapter box, with no connection to the respective PS crates (also installed in racks, Figure 7.6). The GSS channels were left open. So, the OUT channels were all interlocked by the GSS signals, while the T2I were at normal temperature, ranging from 26.6 °C to 26.8 °C, which is a reasonable temperature at the rack. The monitoring interface remained running and accessible at the CERN's Alma9 machine, as well as the LISSY crate and PYNQ board. Figure 7.5 displays the final panel for the Demonstrator implementation.

³The type and sub-type of Power Supply were introduced in Section 2.4.

7.2 The Demonstrator PEB

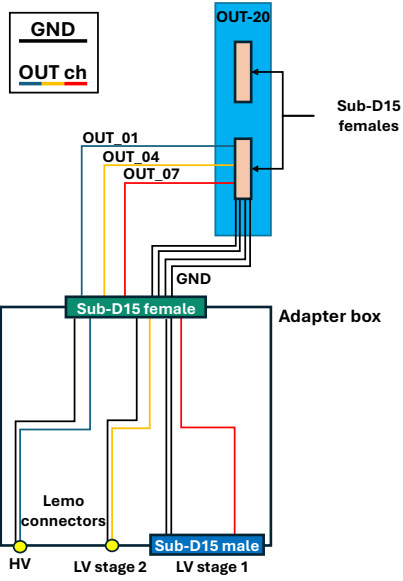


Figure 7.3: Adapter box illustration. The OUT channels and GND pins were protected by the same rubber cable.

The LISSY was installed in a dedicated rack, right in front of the PEB's clean room⁴. The racks had already installed routers for network connection, used by the PYNQ board and the ILK-FPGA, the PS crates and other crates. The strategic racks' location allow a straightforward PEB to crate communication. In Figure 7.6, which shows the final rack setup, it is possible to see the lateral cabling communication, allowing to exchange data between crates and to maintain the organization of the racks, since it houses crates for multiple purposes. The cabling to the Demonstrator is brought down to the horizontal lather (lower right corner) that helps to maintain, again, an organized setup. It conducts all the PEB cables that go inside the clean room, such as the T2I card cables, ensuring external communication. The NTCs that were left connected, as well as the adapter box, were kept between racks.

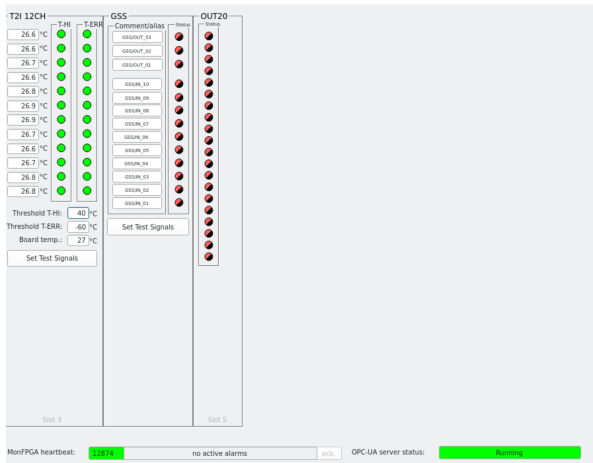


Figure 7.5: Panel implementing for the final layout. It remained active and accessible. The T2I channels have the NTC boards connected and the GSS channels were left open, causing interlock at the OUT channels.

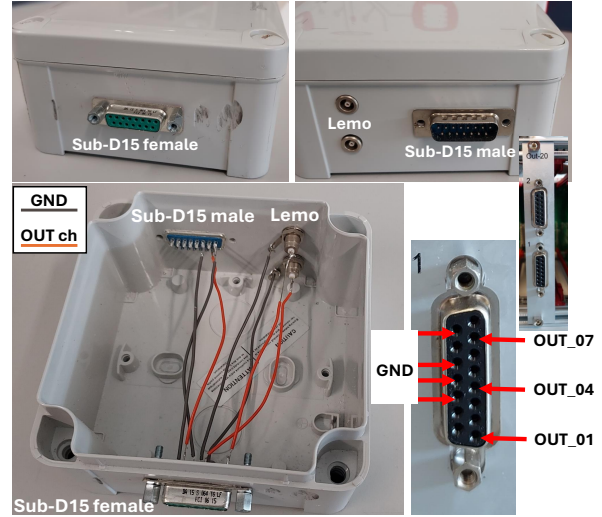


Figure 7.4: The adapter box, both sides of it (upper side) and the OUT connector pins (three channels and four GND.)

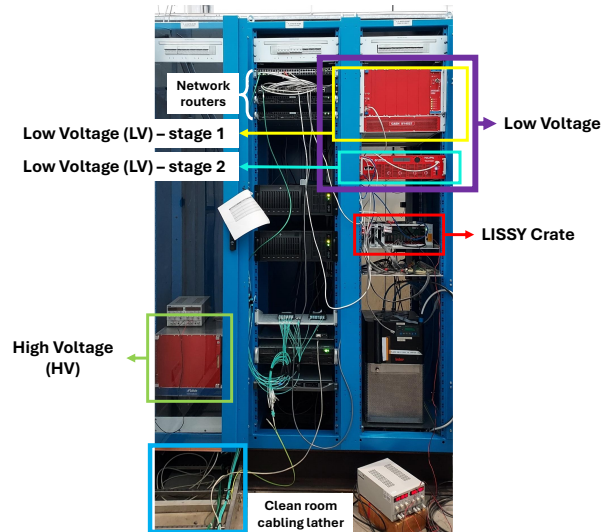


Figure 7.6: Installed rack setup. Connection between the LISSY and PS are made through the laterals sides of the racks.

⁴Laboratory highly protected from contamination and with a low airborne particle's concentration that could damage or interfere with the development the PEB board. Is regularly cleansed and used in research and industrial nanoscale work.

7. THE DEMONSTRATOR IMPLEMENTATION

7.3 The Interlock System strategy

When dealing with various sets of requirements, the **HIS** demands a strategy. This strategy should establish how the system will meet these requirements. For instance, how the monitoring of a certain data will be addressed or how the time delays between the **DCS** and **HIS** decision will be handled (Section 3.1.3). The strategy has to be documented and it is still in development [57], under the responsibility of the current local **HIS** developers. Although these requirements were already well established, they required an update due to new a proposal of requirements, which modified the **LISSY** layout.

The initial layout included a set of 4 **LISSY** crates, each one composed by one **Mon-FPGA**, one **ILK-FPGA**, one **GSS**, 14 **T2I-16** and 3 **OUT-20** cards. Each **LISSY** would be dedicated to a **HGTD** disk, leaving each **T2I** channel with a correspondence of one **NTC**, since one disk has 224 **NTCs**. **Each OUT card would be connected to Power Supplies of one sub-type**⁵ (3 sub-types) (1 **OUT** card - 1 **PS** sub-type). The **HIS** is expected to use in total 38 to 40 **OUT** channels associated to 1 **HV** (maximum of 10 per **LISSY**), 40 channels to **LV Stage 2** (10 per **LISSY**) and 10 channels to **LV Stage 1** (maximum of 3 per **LISSY**).

In the meantime, due to cabling constraints⁶, another **HGTD** developer proposed an alternative layout. This layout would demand a larger number of **T2I-16** cards per crate, from 14 to 16 cards. This way, with the same amount of **NTCs** per disk, not all **T2I** channels would be used. Given the fixed number of available **I/O** slots in a **LISSY** crate of 18⁷, the amount of **T2I** cards would leave two slots for one **GSS** card and one **OUT** card. This **one OUT card would then have to accommodate all three PS sub-types** (1 **OUT** card - 3 **PS** sub-types). The previous arrangement (1 **OUT** card - 1 **PS** sub-type) **provided a higher level of granularity regarding the PS sub-type specification**, because there would be one **OUT** card controlling one **PS** sub-type, instead of mixing all three sub-types in one **OUT** card. This granularity would be important given, among other reasons not deepened in this thesis, the **Mon-FPGA** strategy of monitoring the **PS** sub-type information through an **EEPROM** chip on each **OUT** card, which will be discussed in Section 7.3.1.

Apart from that, the new approach of 1 **OUT** card - 3 **PS** sub-types would need to **evaluate the voltage drop between the card and the connected PS**. The previous **OUT** card layout (1 **OUT** card - 1 **PS** sub-type) also allowed the use of **OUT-20** cards, which contains two Sub-D15 connectors, each one with 3 **GND** pins, making a ratio of 10/3 outputs per **GND** pin on each connector. The lower is the number of outputs per **GND** pin, the better the noise reduction on the outputs. A larger ratio, meaning less **GND** pins per outputs, could be insufficient to drive so many outputs, causing a larger and undesirable voltage drop on the outputs. While the new layout (1 **OUT** card - 3 **PS** sub-types) forces the use of an **OUT-24** or **OUT-32** card. The **OUT-24** and **OUT-32** cards have 8/1 and 32/5 outputs to **GND** ratio, respectively.

All **OUT** cards sub-types were developed by the **ILK-FPGA** team, with the intention of applying them to the **ITk** detector. This issue was never taken as a concern, given the small cabling length between the **OUT** channels and the **PS**⁸. However, the **HGTD** cabling is expected to be much longer, from 80 to 100 meters. This was considered initially an issue to be worked on, since it reuses the **ITk** **OUT** cards. At the time, it was preferable to use the **OUT-20** in order to avoid such issue [58, 72]. With the new proposed layout (1 **OUT** card - 3 **PS** sub-types), 20 channels were considered insufficient to cover so

⁵Each **PS** is controlled by one **OUT** channel, but in this layout each **PS** sub-types would be specific of an **OUT** card.

⁶The cable distribution between the **NTCs**, installed in the detector disks, and the **T2I** cards on the final **HGTD** layout was updated due to cabling constraints, which will not be addressed in this thesis.

⁷It has 20 slots in total, but the **Mon-FPGA** and **ILK-FPGA** occupy the slots 1 and 20, respectively.

⁸The larger the cable length, the larger the voltage drop between ends can be.

many connections. **Per LISSY are required at least 23 OUT channels** for every PS, forcing the use of at least the OUT-24 card. So, this subject required specific testing to evaluate the voltage drop between cabling ends. This task was not under the responsibility of the local HIS developers.

Other layout options were discussed, although they would increase the HIS hardware complexity and the cost of material. Compared to the new 1 OUT - 3 PS sub-types layout, they were less favourable and were quickly discarded or considered as last option, in the case that the new layout was not feasible.

7.3.1 The power supply sub-type monitoring

As detailed in Section 4.5 and illustrated in Figure 4.1, the Mon-FPGA reads the EEPROM chip of each connected I/O cards. This allows the Mon-FPGA to **recognize which card is reading and its sub-type**⁹ and then it **updates the DCS on it**. As it is stated in *The ITk Interlock Mon-FPGA module firmware* document, the *MON-FPGA shall read part of the information stored in the small I2C EEPROMs present in every IO module. The mandatory information is stored in the first byte, describing the module type. The second byte can be associated with the module sub-type (for example to distinguish between LV and HV for a module of type OUT)* [70, p. 6]. This means that, by associating individual OUT cards for each PS sub-type (previous layout, 1 OUT card - 1 PS sub-type) allows the reading of such information inside the OUT card's EEPROM chip by the Mon-FPGA. However, the proposed **new layout** (1 OUT card - 3 PS sub-types), **by combining all three sub-types into one OUT card, disables such capability**.

The new strategy intends to study such approach. For that, two viewpoints will be addressed. If it is considered as **mandatory to read the PS sub-type specification in the EEPROM chip**, then the Mon-FPGA firmware should undergo modifications, which would separate itself from the ITk versions and support. It is worth mentioning that one large advantage of such collaboration was to ease the work integration and support between both detector's teams. If **not mandatory**, the strategy should specify **an alternative approach to recognize the sub-types** and allow that information **update to the DCS**.

Firstly, the new 1 OUT card - 3 PS sub-types layout implies a **distribution of the PS sub-types per OUT channel**, since it combines all 23 PS into one OUT card¹⁰. This distribution was simplified by the previous layout (1 OUT card - 1 PS sub-type), because it was known that the signals from one OUT card were only specific to one PS sub-type. The exorbitant number of possible combinations of the OUT channels to 3 different PS sub-types, in the new layout, makes the EEPROM second byte reading unfeasible. It is emphasized that these combinations have to fit in a 8-bit (byte) sequence, in an optimized structure to ease the Mon-FPGA reading.

On the other hand, by avoiding the EEPROM chip's configuration with the channel's association to a PS sub-type, this specification should be executed in the OPC UA server configuration. As already deepened in Section 4.6.2, each variable from the server configuration file, the "config.xml", specifies a source of monitoring data, such as the I/O cards channels, transmitting the interlock signals and the OUT cards outputs. Each OUT channel is specified with a formatted name and value. This is exemplified in Table 4.1 and Appendix C. As the "config.xml" is modified for each different LISSY card setup, it would be possible and easy to add a new variable attribute to the OUT channels, containing information about the associated PS sub-type. This is feasible since it can easily be defined *a priori*, before setting up the monitoring chain, although the monitoring project would have to undergo modifications to display such information of the DCS. But since it is still under development, it did not represent an issue.

In conclusion, the new strategy it is still in development, although some issues have already been

⁹The card's sub-type are the card's variants, in the case, varying its channels. The PS sub-types are the High Voltage (HV), Low Voltage Stage 1 sub-type (LV Stage 1) and Low Voltage Stage 2 sub-type (LV Stage 2).

¹⁰The card will carry at least 10 HV, 10 LV Stage 2 and at least 3 LV Stage 1.

7. THE DEMONSTRATOR IMPLEMENTATION

addressed. The **granularity of the PS sub-type** with their distribution per OUT card was considered **negligible**. This allows the 1 OUT card - 3 PS sub-types layout. The Mon-FPGA reading of the PS sub-types through the OUT EEPROM chip was also discarded, leaving this **specification under the OPC UA server configuration**. In this way, the DCS will be able to identify the PS sub-type connected to every OUT channel and the Mon-FPGA firmware will still receive support by the ITk team. **The voltage drop** between the OUT-24 and OUT-32 cards and the PS using 80 to 100 meters cabling **still requires testing to evaluate its impact**. However, if the results enable the use of these card sub-types, the decision is to proceed with the proposed 16 T2I and 1 OUT card per LISSY layout as the definitive setup, as well as with the new Mon-FPGA strategy.

8 Conclusion and Future work

The focal objective of this thesis was the implementation of the Monitoring System of the [HGTD Interlock System](#), as a way to monitor the system's status. It would gather the interlock signals, the [NTCs](#)' temperatures and the [Power Supply](#) systems' conditions. This information would then be transmitted to a backend server, thus being available for inspection by a user through an interface connection. The system under study represented a preliminary setup, with only one [LISSY](#) crate, targeting an easy familiarization with it. It only required one card of each module type, each one having a dedicated function.

As made evident throughout this thesis, the Monitoring System is only running correctly when all three blocks are properly connected. For that, it was necessary to deepen the knowledge of already well developed and complex systems, in order to run them in a local [HIS](#). Then, the reduced setup can be efficiently adapted into a full [HGTD](#) setup, given its modular and replicative nature. The local data gathering is executed through one of the system's modules, the [Mon-FPGA](#) card. It is configured with [VHDL](#) firmware, implementing a complex interconnected block system, and upon receiving a backend request the card collects and returns the data. The backend requests imply that the [Mon-FPGA](#) recognizes what type of card is in each [LISSY](#) crate slot, which is only possible by reading the card's intrinsic information stored in memory chips. A proper [Mon-FPGA](#) implementation, one of the key elements of the Monitoring System, required an in-depth study of this firmware as a way to prepare for unforeseen issues, such as the case of a new [HIS](#) strategy. However, in this thesis no firmware modifications were required.

The server runs in a dedicated hardware, a development board (PYNQ-Z2) which is a replacement of the final hardware piece, the [EMP](#) board, still under development. The board configures the aforementioned [Mon-FPGA](#), as well as runs the server. The former operation needs to be executed before the latter. Both functions are based on different setups and scripts execution adapted from a similar detector's system, the [ITk](#). The board, which operates under a Linux system, needed an integration process and initial setup configurations to run both functionalities in the FCUL local network and in the [CERN](#) network.

With the server running and updating data, the monitoring of the local system was done through a user interface, developed in [WinCC OA](#), that displays the system's status. This WinCC project is based on a modular scheme, only containing reference panels, which are replicated to build each card installed in the crate. It takes the number of the card's channels and replicates them as many times as necessary. This project fulfilled the local [HIS](#) requirements. However, in order to hold more complex systems and fulfil all the [HGTD](#) requirements, it has to be developed, like it happens with the [Mon-FPGA](#) firmware. However, they were used in this thesis in a stable development stage. As every generic monitoring project requires, a database to enable historic data analysis was configured although its final stage was not reached over the course of the thesis. The database integration in the [CERN](#)'s framework required a specialized authorization, provided by [CERN](#) staff, which it did not come into place. However, all the database development that could be done locally in the [HIS](#) was described in this thesis and allows for a future easy migration.

The [HIS](#) and its Monitoring System was developed and refined in a local laboratory at FCUL, Lisbon, with the final goal being the [CERN](#) implementation in a realistic scenario. For that, to ensure that every part of the system was operating properly, more tests beyond the Monitoring System had to be done. This

8. CONCLUSION AND FUTURE WORK

thesis specified the testing of the available **T2I** card, which connects to the **HGTD**'s **NTCs**. This required an in-depth study of the electronic circuitry within this card, going from the receiving **NTC** signal, to an **ADC** voltage to digital conversion, to be sent to the backend server for temperature calculations, according to the sensor's temperature equation. Also, the signal comparison with thresholds through **OP-AMPs**' comparators for each temperature limit was studied. These **OP-AMPs** implement a hysteresis, creating a no-transition range, which mitigates noise-induced transitions between the states of normal operation and danger situation. The local tests were conducted with **NTC** simulation, with equivalent resistors and potentiometers using resistance changes as a way to mimic temperature variations. In addition, simpler tests were executed to the **GSS** and the **OUT** cards, albeit out of the frame of this thesis development.

At last, after the successful preliminary tests, the final **HIS** implementation at **CERN** required real-time temperature tests, only possible with **NTCs** connected on the **T2I** channels. The sensors used were the same as the ones predicted to be used in the **HGTD**. The tests were executed with the aid of a climate chamber that provided a necessary temperature controlled environment. There were several test runs: the one focused in this thesis was a step ramp-up and ramp-down consecutive run that went beyond the high temperature threshold of 40 °C, while under the hysteresis range. The goal was to cause state transitions between normal operation and danger situation, in both ways, on a single test run. The **NTCs** were mounted in boards designed to allow to place them inside the chamber, while being connected to the card's channels. One of them also had secondary temperature sensors that allowed to read the environment temperature. This availability was a welcome asset, given the chamber's dimension compared to the smaller developed **NTC** boards, which caused a major temperature discrepancy. During the test run, the health of the system was verified, by transitioning states at the expected temperatures. Both tests, the preliminary and the real-time temperature tests, allowed to study the Monitoring System's behaviour and to verify the required accuracies: for the **ADC** temperatures estimations of 0.5 °C, for the **NTCs**' 1% and for the thresholds' 1 °C [42]. After the favourable test results, the final implementation setup, the Demonstrator, was assembled, leaving the **HIS** ready for the streamlined connection to the final system, which was still in development at the time of the setup assembly.

This thesis' final implementation will allow an initial **HGTD** setup, to verify not only the **HIS** operation, but also other detector's elements. Due to the **HIS** modularity and Monitoring System easy-to-set project, the adaptation to more complex systems is not an issue, although the Monitoring System is far from concluded. For instance, there are **Mon-FPGA** firmware issues regarding the processing of test signals. These should allow to send false danger signals to the other cards of the **HIS**, through the WinCC interface, testing the system's reaction. For now, the development board that runs the server is well suited for the job, although the server will be run, in the future, by the planned **EMP**. The monitoring WinCC project is as well incomplete. The project was only tested properly with a single **LISSY** crate. Although several simulations were made to verify the project's behaviour under more complete system's layouts, given the limited material a full testing was not executed. The project's database will as well be further developed in the future, in addition to the work accomplished in this thesis, through progressively complex implementations.

As future work, given the local **HIS** developers' familiarization with the Demonstrator setup, the **HIS** integration in the Demonstrator is set to be concluded. The next upcoming project will be an expansion of the Demonstrator system. It will use an entire **HGTD** disk quadrant, connecting 28 **NTCs**, requiring more than one **T2I** card. As the development advances, the coordinate collaboration between the **ITk** Interlock System's teams is expected to continue, providing both detector's teams more in-depth feedback. This will accelerate the progress of both detector's integration in the **HL-LHC**.

References

- [1] *3386B-1-204LF Bourns*. Bourns. Available at Mouser Electronics. URL: <https://pt.mouser.com/ProductDetail/Bourns/3386B-1-204LF?qs=nE0RJHjSZ6XhdnZHIUJAQg%3D%3D> (visited on 02/04/2025) (cit. on p. 64).
- [2] *3386B-1-501LF*. Bourns. Available at Mouser Electronics. URL: <https://pt.mouser.com/ProductDetail/Bourns/3386B-1-501LF?qs=nE0RJHjSZ6WBf1F2E60EwA%3D%3D> (visited on 02/04/2025) (cit. on p. 64).
- [3] *3386P-DF6-104LF*. Bourns. Available at Mouser Electronics. URL: <https://pt.mouser.com/ProductDetail/Bourns/3386P-DF6-104LF?qs=G0bayqEFkQBe71%2FXK5EYGg%3D%3D> (visited on 02/04/2025) (cit. on p. 64).
- [4] *3386U-1-504ALF*. Bourns. Available at Mouser Electronics. URL: <https://pt.mouser.com/ProductDetail/Bourns/3386U-1-504ALF?qs=67cj60zW39g7DMT14s4QYA%3D%3D> (visited on 02/04/2025) (cit. on p. 64).
- [5] *34401A Multimeter*. This product is now obsolete. Hewlett-Packard. 1996 (cit. on pp. 61, 67).
- [6] *About (OpenOCD User's Guide)*. URL: <https://openocd.org/doc/html/About.html> (visited on 11/03/2024) (cit. on p. 28).
- [7] *Accelerators*. CERN. URL: <https://www.home.cern/science/accelerators> (visited on 11/05/2024) (cit. on p. 5).
- [8] *ADS1119 Data Sheet, Product Information and Support*. Texas Instruments. URL: <https://www.ti.com/product/ADS1119> (visited on 12/19/2024) (cit. on pp. 59, 121).
- [9] *Architecture*. URL: https://www.winccoa.com/documentation/WinCCOA/3.19/en_US/GettingStarted/GettingStarted-06.html (visited on 01/02/2025) (cit. on pp. 37, 38).
- [10] *AT24CS01/AT24CS02*. Microchip Technology Inc. 2020. URL: <https://www.microchip.com/en-us/product/at24cs02%5C#document-table> (cit. on p. 15).
- [11] *Atlas-Hgtd / DAQLumiMonitoring / Interlock / Mon-FPGA / ATLHGTLISSY_PEB1F*. GitLab. Sept. 18, 2024. URL: https://gitlab.cern.ch/atlas-hgtd/daqlumimonitoring/Interlock/mon-fpga/atlhgtlissy_peg1f (visited on 11/13/2024) (cit. on pp. 39, 49, 75).
- [12] *Atlas-Hgtd / DAQLumiMonitoring / Interlock / Mon-FPGA / HGTD_MonFPGA*. GitLab. Sept. 18, 2024. URL: https://gitlab.cern.ch/atlas-hgtd/daqlumimonitoring/Interlock/mon-fpga/hgtd_monfpga (visited on 11/13/2024) (cit. on pp. 21, 32, 75).
- [13] *atlas-ntp-dcs/dcs_itkpixsr1st*. GitLab. URL: https://gitlab.cern.ch/atlas-ntp-dcs/dcs_itkpixsr1st (visited on 01/05/2024) (cit. on p. 39).
- [14] *AUP PYNQ-Z2*. AMD. URL: <https://www.amd.com/en/corporate/university-program/aup-boards/pynq-z2.html> (visited on 11/04/2024) (cit. on p. 28).
- [15] A. Barriuso Poy et al. *ATLAS DCS Integration Guidelines*. Doc. ID: ATLAS-DQ-ON-0013 v2.0. access: Internal. Aug. 8, 2006. URL: <https://edms.cern.ch/document/685451/2> (cit. on p. 39).

REFERENCES

- [16] M. Beretta et al. *DCS: Requirements Document for HL-LHC*. Doc. ID: ATU-GE-ES-0004 v.1. access: Internal. June 11, 2020. URL: <https://edms.cern.ch/document/2276493/1> (cit. on p. 19).
- [17] Maximilien Brice. *ATLAS Barrel Detector*. 2003. URL: <https://cds.cern.ch/record/619028> (visited on 11/06/2024) (cit. on p. 7).
- [18] *Calorimeter*. CERN. URL: <https://atlas.cern/Discover/Detector/Calorimeter> (visited on 11/06/2024) (cit. on p. 7).
- [19] C. W. Chen, S. Kersten, J. P. Martin, and N. Starinski. *The ITk Interlock Mon-FPGA Module Firmware*. Doc. ID: AT2-IE-ES-0008 v2.0. access: Internal. Oct. 24, 2021. URL: <https://edms.cern.ch/document/2399605/2> (cit. on pp. 24, 26).
- [20] C. W. Chen et al. *Specifications for the ITk Main Interlock Crate*. Doc. ID: AT2-I-ES-0023 v.1.1. access: Internal. Feb. 14, 2023. URL: <https://edms.cern.ch/document/2772913/1> (cit. on pp. 13, 14).
- [21] C. W. Chen et al. *The ITk Local Interlock and Safety System Crate*. Doc. ID: AT2-IE-ES-0007 v.3.6. access: Internal. Aug. 30, 2023. URL: <https://edms.cern.ch/document/2399603/3> (cit. on pp. 13, 14, 17–20, 64, 121).
- [22] H. Chen et al. *Report of the HGTD DCS/Environmental Monitoring Final Design Review (FDR)*. Doc. ID: TC-R-MR-0141 v.1. access: Internal. Jan. 25, 2025. URL: <https://edms.cern.ch/document/3230929/1> (cit. on p. 75).
- [23] International Electrotechnical Commission. *IEC 60297-3-108:2014*. Sept. 9, 2014. URL: <https://webstore.iec.ch/en/publication/1291> (cit. on p. 13).
- [24] *CROSS SECTION AND LUMINOSITY*. CERN. URL: <https://cds.cern.ch/record/2800578/files/Cross%20Section%20and%20Luminosity%20Physics%20Cheat%20Sheet.pdf> (visited on 11/06/2024) (cit. on p. 8).
- [25] *Cyclone 10 LP*. Intel. URL: <https://www.intel.com/content/www/us/en/products/details/fpga/cyclone/10/lp/docs.html> (visited on 11/03/2024) (cit. on p. 22).
- [26] *Data Point Concept, Process Image*. URL: https://www.winccoa.com/documentation/WinCCOA/3.19/en_US/GettingStarted/GettingStarted-12.html (visited on 01/03/2025) (cit. on p. 38).
- [27] *Data Sheet AD8601/AD8602/AD860*. Analog Devices. URL: https://www.analog.com/media/en/technical-documentation/data-sheets/ad8601_8602_8604.pdf (visited on 01/25/2025) (cit. on pp. 60, 61).
- [28] *Downloads - Unified Automation*. Unified Automation GmbH. URL: <https://www.unified-automation.com/downloads.html> (visited on 12/14/2024) (cit. on p. 32).
- [29] D. Ecker et al. “The Embedded Monitoring Processor for High Luminosity LHC”. In: *Proc. ICALEPCS’23* (Oct. 9–13, 2023). 19. JACoW Publishing, Geneva, Switzerland, Feb. 2024, pp. 1470–1474. DOI: 10.18429/JACoW-ICALEPCS2023-THPDP063. URL: <https://jacow.org/icalepcs2023/papers/thpdp063.pdf> (cit. on p. 27).
- [30] *EP1C12Q240C8N* Altera. Intel. Available at Mouser Electronics. URL: <https://eu.mouser.com/ProductDetail/Altera/EP1C12Q240C8N?qs=P2rvyRAZsYit9ew9UxvGFg%3D%3D> (visited on 11/03/2024) (cit. on p. 22).

REFERENCES

- [31] *Experiments*. CERN. URL: <https://www.home.cern/science/experiments> (visited on 11/09/2024) (cit. on p. 6).
- [32] Bruno Jesus Fernandes. “Implementação em VHDL do Módulo MICTP do rimeiro Nível do trigger da Experiência ATLAS”. Faculdade de Ciências, Universidade de Lisboa. Relatório do Estágio Profissionalizante para obtenção do Grau de Licenciado em Engenharia Física. 2008 (cit. on p. 23).
- [33] *FPGA Design Software - Quartus Prime*. Intel. URL: <https://www.intel.com/content/www/us/en/products/details/fpga/development-tools/quartus-prime/docs.html> (visited on 11/03/2024) (cit. on pp. 22, 31).
- [34] C. Gemme et al. *ITk Interlock System Strategy*. Doc. ID: AT2-IE-ES-0005. access: Internal. Nov. 3, 2021. URL: <https://edms.cern.ch/document/2387552/7> (cit. on p. 20).
- [35] *Glossary | ATLAS Open Data*. CERN. URL: <https://opendata.atlas.cern/docs/atlas/GLOSSARY> (visited on 11/06/2024) (cit. on pp. xv, 7).
- [36] Trenc Electronic GmbH. *TE0711 - Artix 7*. Trenc Electronic GmbH Online Shop (EN). URL: <https://shop.trenz-electronic.de/en/Products/Trenz-Electronic/TE07XX-Artix-7/TE0711-Artix-7/> (visited on 10/30/2024) (cit. on p. 22).
- [37] JCOP Working Group. *JCOP Framework | Components*. CERN. URL: <https://jcop.web.cern.ch/pages/components.html> (visited on 01/06/2025) (cit. on pp. 40, 41, 75).
- [38] JCOP Working Group. *The CERN Joint COntrols Project*. CERN. URL: <https://jcop.web.cern.ch/> (visited on 01/06/2025) (cit. on p. 40).
- [39] *High-Luminosity LHC*. CERN. URL: <https://home.cern/resources/faqs/high-luminosity-lhc> (visited on 11/07/2024) (cit. on p. 9).
- [40] IEEE. “IEEE Standard for VHDL Language Reference Manual”. In: *IEEE Std 1076-2019* (2019), pp. 1–673. DOI: [10.1109/IEEESTD.2019.8938196](https://doi.org/10.1109/IEEESTD.2019.8938196) (cit. on p. 22).
- [41] *ITkCommonInterlock/MonFPGA*. GitLab. July 9, 2024. URL: <https://gitlab.cern.ch/itkcommoninterlock/monfpga/-/tree/master> (visited on 10/05/2024) (cit. on pp. 21, 31, 57).
- [42] Gritsay K. et al. *HGTD DCS and Interlock: Requirements Document for HL-LHC*. Doc. ID: AT2-G-ES-0013 v.1. access: Internal. June 11, 2020. URL: <https://edms.cern.ch/document/2648566/1> (cit. on pp. 9, 11, 12, 16, 17, 57, 60, 63, 65, 67, 75, 79, 86).
- [43] *Key Achievements | CERN*. CERN. URL: <https://home.web.cern.ch/about/key-achievements> (visited on 11/05/2024) (cit. on p. 5).
- [44] *KT High Accuracy SMD NTC Thermistor*. Semitec USA Corp. URL: <https://atcsemitec.co.uk/brand/semitec/> (visited on 02/18/2025) (cit. on pp. 48, 76, 79).
- [45] *KT Thermistor*. Semitec USA Corp. URL: https://www.semitec-global.com/products/thermistor_kt/ (visited on 02/17/2025) (cit. on pp. 33, 60, 76).
- [46] *LHC Aerial View Postcard*. CERN. URL: <https://visit.cern/node/616> (visited on 11/05/2024) (cit. on p. 6).
- [47] *LT3042 Datasheet and Product Info*. Analog Devices. URL: <https://www.analog.com/en/products/lt3042.html> (visited on 12/29/2024) (cit. on pp. 57–59, 122).
- [48] S. Malyukov. “Interlock layout”. HGTD Interlock layout discussion. Sept. 5, 2024 (cit. on p. 17).

REFERENCES

- [49] Claudia Marcelloni. *Installation of the First of the Big Wheels of the ATLAS Muon Spectrometer, a Thin Gap Chamber (TGC) Wheel. Installation de La Première Des Grandes Roues Du ATLAS Spectromètre à Muons, Une Roue TGC*. 2006. URL: <https://cds.cern.ch/record/986163> (visited on 11/06/2024) (cit. on p. 7).
- [50] Claudia Marcelloni. *Moving One of the ATLAS End-Cap Calorimeters. Vue de La Caverne ATLAS (Côté A) Pendant Que Le Calorimètre Bouchon Est Déplacé*. 2007. URL: <https://cds.cern.ch/record/1017394> (visited on 11/06/2024) (cit. on p. 7).
- [51] *Max 10 FPGA*. Intel. URL: <https://www.intel.com/content/www/us/en/products/details/fpga/max/10/docs.html> (visited on 11/03/2024) (cit. on p. 22).
- [52] *MCP23017/MCP23S17*. Microchip Technology Inc. 2020. URL: <https://www.microchip.com/en-us/product/mcp23017%5C#document-table> (cit. on p. 15).
- [53] *Muon Spectrometer*. CERN. URL: <https://atlas.cern/Discover/Detector/Muon-Spectrometer> (visited on 11/09/2024) (cit. on p. 7).
- [54] *NextGenArchiver*. Doc. ID: CERN BE-ICS-FT-2397531 v.2.0.0. access: Internal. Dec. 1, 2023. URL: <https://edms.cern.ch/document/2397531/2.0.0> (cit. on p. 41).
- [55] *OPC FOUNDATION - Home Page*. OPC Foundation. URL: <https://opcfoundation.org/> (visited on 12/10/2024) (cit. on p. 27).
- [56] *Our People*. CERN. Oct. 17, 2024. URL: <https://home.cern/about/who-we-are/our-people> (visited on 11/05/2024) (cit. on p. 5).
- [57] A. Parreira, H. Santos, and Vieira R. “HGTD Interlock Strategy: Specifications Document for HL-LHC”. access: Unpublished. Feb. 19, 2025. URL: [None](#) (cit. on p. 82).
- [58] A. Parreira, H. Santos, and R. Vieira. “Layout of the HGTD Interlock”. HGTD Interlock layout discussion. Sept. 26, 2024 (cit. on p. 82).
- [59] Joao Pequeno. *Computer Generated Image of the ATLAS Inner Detector*. 2008. URL: <https://cds.cern.ch/record/1095926> (visited on 11/06/2024) (cit. on p. 7).
- [60] *PMLL4148L (High-speed Switching Diodes)*. Nexperia. URL: <https://www.nexperia.com/product/PMLL4148L> (visited on 02/10/2025) (cit. on p. 71).
- [61] *Project Jupyter Documentation — Jupyter Documentation 4.1.1 Alpha Documentation*. Jupyter. URL: <https://docs.jupyter.org/en/latest/> (visited on 12/29/2024) (cit. on p. 29).
- [62] *PuTTY: A Free SSH and Telnet Client*. PuTTY. URL: <https://www.chiark.greenend.org.uk/~sgtatham/putty/> (visited on 12/06/2024) (cit. on p. 30).
- [63] *PYNQ-Z2 Setup Guide — Python Productivity for Zynq (Pynq)*. Read the Docs. URL: https://pynq.readthedocs.io/en/v2.6.1/getting_started/pynq_z2_setup.html (visited on 12/05/2024) (cit. on p. 30).
- [64] *Quasar Framework - OPC-UA Server Generation*. URL: <https://knowledge-transfer.web.cern.ch/technologies/quasar-framework-opc-ua-server-generation> (visited on 12/10/2024) (cit. on p. 32).
- [65] *quasar-team/Cacophony*. quasar-team, Nov. 26, 2024. URL: <https://github.com/quasar-team/Cacophony> (visited on 01/04/2025) (cit. on pp. 41, 42).
- [66] *quasar-team/quasar*. Nov. 25, 2024. URL: <https://github.com/quasar-team/quasar> (visited on 12/11/2024) (cit. on p. 32).

- [67] Chiara Rizzi. *A High-Granularity Timing Detector for the ATLAS Phase-II upgrade*. Tech. rep. Geneva: CERN, 2021. DOI: [10.22323/1.390.0868](https://cds.cern.ch/record/2747270). URL: <https://cds.cern.ch/record/2747270> (cit. on p. 9).
- [68] *SIMATIC WinCC OA World LogoSIMATIC WinCC OA World | Online Documentation*. Siemens. URL: <https://www.winccoa.com/product-information/documentation.html> (visited on 01/02/2025) (cit. on p. 37).
- [69] *SIMATIC WinCC Open Architecture Version 3.18 Documentation | _smooth (Smoothing)*. Siemens. URL: https://www.winccoa.com/documentation/WinCCOA/3.18/en_US/Referenz_PARA/Referenz_PARA-16.html (visited on 03/23/2025) (cit. on p. 54).
- [70] C. Solans et al. *Firmware requirements of MON-FPGA*. Doc. ID: AT2-IE-ES-0010 v2.0. access: Internal. Jan. 4, 2021. URL: <https://edms.cern.ch/document/2447073/2> (cit. on pp. 16, 83).
- [71] *Supervisor: A Process Control System — Supervisor 4.2.5 Documentation*. URL: <http://supervisord.org/> (visited on 12/11/2024) (cit. on p. 32).
- [72] ITK developers team. *Personal communication*. Discussions with the developers team. n.d. (Cit. on pp. 22, 27, 50, 51, 73, 82).
- [73] *Technical Design Report: A High-Granularity Timing Detector for the ATLAS Phase-II Upgrade*. Tech. rep. Geneva: CERN, 2020. URL: <https://cds.cern.ch/record/2719855> (cit. on pp. 9, 10).
- [74] *The Accelerator Complex*. CERN. URL: <https://www.home.cern/science/accelerators/accelerator-complex> (visited on 11/05/2024) (cit. on p. 6).
- [75] *The History of CERN*. URL: <https://timeline.web.cern.ch/timeline-header/89> (visited on 11/05/2024) (cit. on p. 5).
- [76] *The Inner Detector*. CERN. URL: <https://atlas.cern/Discover/Detector/Inner-Detector> (visited on 11/09/2024) (cit. on p. 7).
- [77] *The Large Hadron Collider*. CERN. Oct. 17, 2024. URL: <https://home.web.cern.ch/science/accelerators/large-hadron-collider> (visited on 11/05/2024) (cit. on p. 5).
- [78] *The LHC Racks up Records*. CERN. Oct. 17, 2024. URL: <https://home.cern/news/news/accelerators/lhc-racks-records> (visited on 11/05/2024) (cit. on p. 6).
- [79] *Trigger and Data Acquisition System*. CERN. URL: <https://atlas.cern/Discover/Detector/Trigger-DAQ> (visited on 11/06/2024) (cit. on p. 6).
- [80] *USB Blaster V2 - Waveshare*. Waveshare Electronics. URL: https://www.waveshare.com/wiki/USB_Blaster_V2 (visited on 12/04/2024) (cit. on p. 28).
- [81] *Verilog vs. VHDL – Digilent Blog*. Feb. 10, 2016. URL: <https://digilent.com/blog/verilog-vs-vhdl/> (visited on 11/02/2024) (cit. on p. 22).
- [82] *Vivado Design Suite*. AMD. URL: <https://www.amd.com/en/products/software/adaptive-socs-and-fpgas/vivado.html> (visited on 10/30/2024) (cit. on pp. 22, 31).
- [83] *VT4 4002 EMC*. Vötsch. URL: <https://weiss-na.com/product/vt-4002-emc/> (visited on 02/16/2025) (cit. on p. 76).

REFERENCES

- [84] D. Wiarda et al. *Overview of Nuclear Data Uncertainty in Scale and Application to Light Water Reactor Uncertainty Analysis*. U.S.NRC. Dec. 2018. URL: <https://www.nrc.gov/docs/ML1900/ML19009A313.pdf> (visited on 11/07/2024) (cit. on p. 8).
- [85] WinSCP. WinSCP. URL: <https://winscp.net/> (visited on 12/06/2024) (cit. on p. 30).
- [86] *Zynq-7000 SoC Data Sheet: Overview (DS190)*. AMD. 2018. URL: <https://www.amd.com/en/products/adaptive-socs-and-fpgas/soc/zynq-7000.html> (visited on 11/03/2024) (cit. on p. 28).

Appendices

A OUT module logic

The purpose of this table is to demonstrate the outcome of the possible combinations of the interlock signals at the gate inputs that the OUT channel receives.

Table A.1: Logic inverting process behind the OUT signals. The logic is executed with an OR Gate. When receiving "ERROR" or "Broken Cable" signal from either the interlock matrix or the [Slot Int](#) signals, the logic gate drives 3.3 V at its output and drives 0 when the interlock signals are "OK". To invert this positive logic (0 V at the output corresponds to "0", "OK" state, and 3.3 V corresponds to "1", "INTERLOCK" state), it was implemented a current limiting circuit with a [P-MOS](#), which only drives 3.3 V when at the OR gate output is 0 V (when both gate inputs are at "OK" state). "Ch" is the abbreviation of channel. The OR Gate inputs column has both the binary signal and the corresponding signal state.

OR Gate inputs				OR Gate output	OUT ch output (V)	OUT ch state
Interlock Matrix		Slot Int				
0	"OK"	0	"OK"	0	3.3	"OK"
0	"OK"	1	"INTERLOCK"	1	0	"INTERLOCK"
1	"INTERLOCK"	0	"OK"	1	0	"INTERLOCK"
1	"INTERLOCK"	1	"INTERLOCK"	1	0	"INTERLOCK"

Figure A.1 displays a part of the OUT card schematic, focusing on the one output channel, with each section of the Table A.1 highlighted.

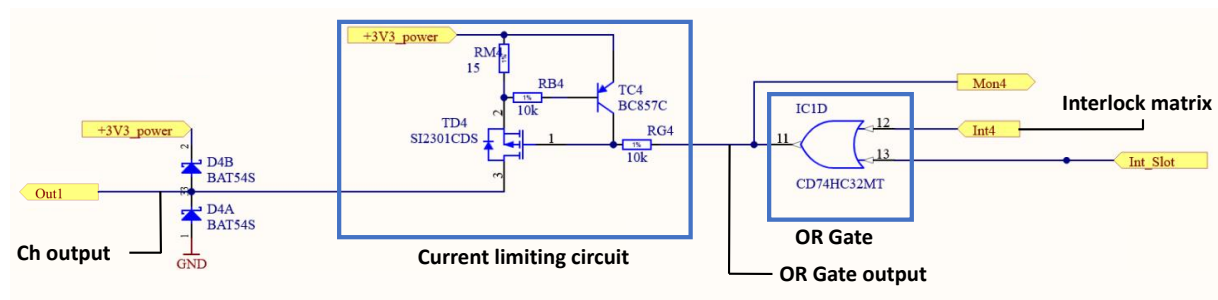


Figure A.1: OUT output channel schematic for the channel OUT_1. The Table A.1 sections are highlighted for an easy comparison. "Int_slot" corresponds to the [Slot Int](#) signal. This is a part of the schematic of the OUT card version (ITKInterlock_Out20chESD) which was used.

B Mon-FPGA firmware

B.1 Encoding 4b10b

Table B.1: Table with 4b10b encoding. If the 10-bit command packet does not correspond to any of the presented options and if the 5 **LSBs** do not coincide with the 5 **MSBs** negation (or complement), the modules which deal with the encoding ("decode_4b10b" and "encode_4b10b") flag an error. If not so, the 5 **MSBs** of the 10-bit command are encoded into a 4-bit nibble and a control bit, K, that distinguish among the command type.

10-bit encoded	5-bit encoded	4-bit decoded	K, control bit	Command
00000 00000	00000	0000	1	Lost signal
00110 11001	00110		0	x"0"
11111 00000	11111	0001	1	Idle character
01001 10110	01001		0	x"1"
11000 00111	11000	0010	1	Begin of packet
10100 01011	10100		0	x"2"
01101 10010	01101	0011	1	End of packet
10101 01010	10101		0	x"3"
00111 11000	00111	0100	1	Reset
01010 10101	01010		0	x"4"
01011 10100	01011	0101	0	x"5"
01110 10001	01110	0110	0	x"6"
01100 10011	01100	0111	0	x"7"
10010 01101	10010	1000	0	x"8"
10011 01100	10011	1001	0	x"9"
10110 01001	10110	1001	0	x"a"
10111 01000	10111	1011	0	x"b"
11010 00101	11010	1100	0	x"c"
11011 00100	11011	1101	0	x"d"
11100 00011	11100	1110	0	x"e"
11101 00010	11101	1111	0	x"f"
Any other	---		1	Error

B.2 Command packet sequence order

The following figure illustrates the 32-bit command packets sequence. On the figure's upper side are the different sections of each long word and their bit vector numbering, in a **VHDL** vector format. The highest bits (start at left) are the **MSBs**. The first 32-bit long word is the header of the packet and it is located at the address x"00" in the *cmdDPM*. The header comprises information on the crate number, used for error detection, comparing it to the value inputted value from the **Mon-FPGA** front-panel. The second byte has the slot number of the card to be read, for example. The "sequence_number" modification triggers the "Main_CTRL" **FSM** (Section 4.5.2). The packet type aids in the distinction of the command packets for an easier recognition by the system. The second long word, stored in the x"01" address, has flag data and the "payload_size" segment can be used to transport data. The third long word has the "cmd_code", which is used in the central **FSM** of the firmware, allowing to select the process requested. The "reg16" and "reg32" are used to store and transport data through the firmware blocks.

B. MON-FPGA FIRMWARE

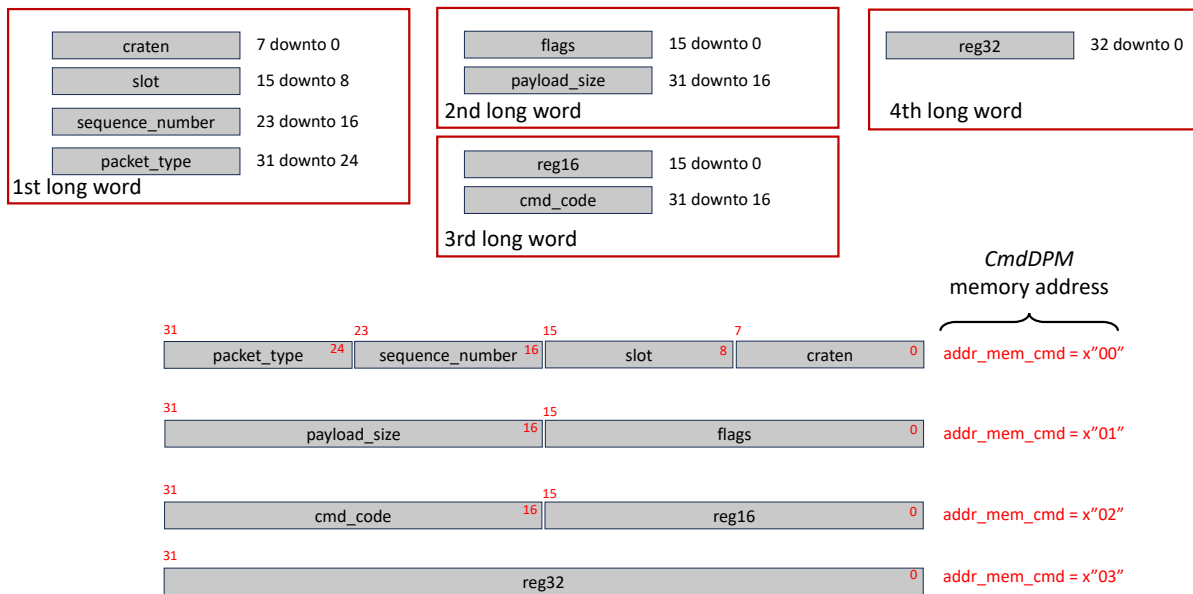


Figure B.1: Illustration of the command packet format, received from the backend server. On the figure's upper side are the segments of each long word and on the bottom are the complete long words. The numbers represent bit numbering inside the 32-bit sequence.

B.3 Command packet code

The "Main_CTRL" FSM reads the "cmd_code" from the command packet to know the processes it should execute. The following bit positions are according to the 16-bit sequence of the "cmd_code" (Figure B.1). The "updateI2Cscripts" is specific to "cmd_code" = x"4010" and this state outputs the flag "enable_update_i2cprogr". This is then used by the corresponding block to update the I^2C scripts, as it was explained in Section 4.5.2. The main Mon-FPGA requirement is controlled by the states "ReadDataBlocks" and "I2Cslots", that output the flags "enable_scan_slots_data" and "start_i2c_main", respectively. This is then sent to the corresponding modules (with the same name as the states) as explained in Section 4.5.3.

Table B.2: "Main_CTRL" process association to the "cmd_code" of the command packet and the outputted enable flags. The states' names correspond to the ones used in Figure 4.2 for easier interpretation. The last two states control the modules with the same name. The flags designation is the same used in the VHDL firmware code

FSM state	Hexa	Binary	Enable Flag
"updateI2Cscripts"	x"4010"	0100 0000 0001 0000	enable_update_i2cprogr
"ReadDataBlocks"	x"002"	0000 0000 0010	enable_scan_slots_data
"I2Cslots"	x"001"	0000 0000 0001	start_i2c_main

B.4 The top VHDL module

This section exemplifies the implementation of the top VHDL module for the Mon-FPGA firmware. It interconnects all the submodules through the mapping of their ports, where most of the inputs of a submodule, result from the outputs of another submodule. The Code B.1 is an example of the top module designated as "monitor_top" and it only shows the submodules detailed in Section 4.5 and their necessary port declarations and instantiations.

B.4 The top VHDL module

Code B.1: Portion of the top [VHDL](#) module which interconnects the submodules through direct port mapping. It only includes the submodules specified in [Section 4.5](#). cmd stands for command.

```
1  (...)
2  entity monitor_top is
3      ----- top module port mapping -----
4      (...)
5  end entity monitor_top;
6
7  architecture behavioral of monitor_top is
8      ----- submodules and port declaration -----
9
10     component from_link is                                -- FromLink module
11     port (
12         ----- from backend/front panel -----
13         craten  : in    std_logic_vector(7 downto 0);    -- crate number from front plane
14         link_in : in    std_logic;                       -- backend serial link
15
16         ----- to cmdDPM -----
17         bram_portb_0_addr : out std_logic_vector(31 downto 0); -- address
18         bram_portb_0_we   : out std_logic_vector(3 downto 0);  -- enable flag
19         bram_portb_0_din  : out std_logic_vector(31 downto 0)  -- cmd packet
20
21         (...);
22     );
23 end component;
24
25 component main_ctrl is                                    -- Main_CTRL module
26     (...)
27     port (
28         ----- from cmdDPM -----
29         dout_cmd_mem      : in    std_logic_vector(31 downto 0); -- cmd packet
30
31         ----- enable flags for processes -----
32         enable_update_i2cprog : out std_logic;    -- update I2C scripts
33         enable_scan_slots_data : out std_logic;   -- ReadDataBlocks
34         start_i2c_main        : out std_logic;   -- I2Cslots
35
36         ----- enable flags for backend transmission -----
37         enable_tolink_chain_i : in    std_logic;  -- enable flag from ReadDataBlocks
38         enable_tolink         : out std_logic;   -- enable flag to ToLink
39
40         (...);
41     );
42 end component main_ctrl;
```

B. MON-FPGA FIRMWARE

```
43 component i2c_slots is                                -- I2Cslots module
44     (...)
45     port (
46         ----- commands from Main_CTRL -----
47         start : in    std_logic;                      -- I2Cslots enable flag
48         ----- for ReadDataBlocks -----
49         ram_dout : out  std_logic_vector(8 downto 0);  -- to ReadDataBlocks
50
51     (...)
52 );
53 end component;
54
55 component read_data_blocks is                          -- ReadDataBlocks module
56     port (
57         ----- commands from Main_CTRL -----
58         start : in    std_logic;  -- cmd5 (ReadDataBlocks) enable flag
59
60         ----- from I2Cslots -----
61         muxd : in    std_logic_vector(8 downto 0);    -- I2C scripts
62
63         ----- to SlotPacketDPM -----
64
65         slot_packet_ram_addr_a : out  std_logic_vector(8 downto 0);  -- address
66         slot_packet_ram_data_a : out  std_logic_vector(31 downto 0);  -- 32-bit long word
67         slot_packet_ram_we_a : out  std_logic;  -- enable flag to write on memory
68
69         ----- to Main_CTRL -----
70         enable_tolink_chain : out  std_logic  -- enable flag to backend transmission
71
72     (...)
73 );
74 end component read_data_blocks;
75
76 component to_link is                                  -- ToLink module
77     port (
78         ----- from Main_CTRL -----
79         enable : in    std_logic;  -- enable flag to backend transmission
80
81         ----- to backend -----
82         serial_out : out  std_logic;  -- serial link to backend
83
84     (...)
85 );
86 end component to_link;
87 (...)
88
```

```

89  begin
90      ----- submodules instantiation and port interconnection -----
91  from_link_inst : component from_link
92      port map (
93          craten  => craten,
94          link_in => E_link_rx,
95
96          bram_portb_0_addr => bram_portb_0_addr,
97
98          bram_portb_0_we  => bram_portb_0_we,
99          bram_portb_0_din => bram_portb_0_din
100      );
101
102  main_ctrl_inst : component main_ctrl
103      port map (
104          dout_cmd_mem => dout_cmd_mem,
105
106          enable_update_i2cprog => enable_update_i2cprog,
107          enable_scan_slots_data => enable_scan_slots_data,
108          start_i2c_main => start_i2c_main,
109
110          enable_tolink_chain_i => enable_tolink_chain,
111          enable_tolink         => enable_tolink,
112
113          (...)
114      );
115
116  i2c_slots_inst : component i2c_slots
117      port map (
118          start => start_i2c_main,
119
120          ram_dout  => ram_dout,
121
122          (...)
123      );

```

B. MON-FPGA FIRMWARE

```
124 read_data_blocks_inst : component read_data_blocks
125     port map (
126         start                => enable_scan_slots_data,
127
128         addr                 => ram_ra,
129         muxd                 => ram_dout,
130
131         slot_packet_ram_addr_a => slot_packet_ram_addr_a,
132         slot_packet_ram_data_a => slot_packet_ram_data_a,
133         slot_packet_ram_we_a   => slot_packet_ram_we_a,
134
135         enable_tolink_chain => enable_tolink_chain
136
137         (...);
138     );
139 to_link_inst : component to_link
140     port map (
141         enable => enable_tolink,
142         bram_portb_1_dout => d32
143
144         serial_out => e_link_tx2,
145
146         (...);
147     );
148
149     (...);
150 end architecture behavioral;
```

As already stated in Section 4.3, the top module interconnects all the other submodules in one port mapping VHDL file. The "monitor_top" starts with its own port mapping on the entity, from line 2 to 5. This part was not considered relevant and was omitted. The module `architecture` starts at line 7 and it is where the other submodules are declared. From line 10 to 23 is the "FromLink" module (Section 4.5.1). It receives the serial backend link through port "link_in" at line 14 and the command packets to be stored in the *cmdDPM* memory, using the ports from line 17 to 19 (the address of where it should be stored in the memory, the flag to enable the writing process and the command packet itself, respectively).

The "Main_CTRL" module (Section 4.5.2) is from line 25 to 42. The command packet (previously stored in the memory) is declared in line 29. The FSM states processes are controlled by the enable flags from line 32 to 34. Section 4.5.2 details these states and their processes. After the card readout by the **Mon-FPGA**, the "ToLink" module (Section 4.5.4) sends it to the backend server. This transmission is enabled by the flags in lines 37 and 38. The process is as follows: the "ReadDataBlocks" (Section 4.5.3) executes the readout from the cards of the crate and stores them into the *SlotPacketDPM* memory, where the "ToLink" reads from. However, the reading only happens when the "ReadDataBlocks" sends to the "Main_CTRL" the designated "enable_tolink_chain" (line 37) flag. The "Main_CTRL", after receiving it, flags the "ToLink" module about the "clear to go" message by sending the "enable_tolink" (line 38) flag. The "ToLink" finally starts its own processes after receiving this flag.

The "I2Cslots" module (Section 4.5.3) is declared from line 43 to 53. At line 47 is the input flag which enables the starting of its process and at line 49 the I²C scripts from the *I2CDPM* are outputted. The "ReadDataBlocks" goes from line 55 to 74. Again, the enable flag is declared in line 58. The actual scripts (referred before) are transmitted through the port at line 61. This process is illustrated in Figure 4.1. The readout data is then stored in the *SlotPacketDPM* through lines 65 to 67 and the enable flag to start the backend transmission is in line 70.

Finally, the "ToLink" is shown in lines 76 to 86, where the enable flag from the "Main_CTRL" is declared in 79 and the serial link to the backend server is outputted in 82. With the submodules declared, the *architecture* specification starts at 89. This is where the port mapping between the submodules (submodules instantiation) is executed. The submodules ports are presented, of which the only ones that should be noted are the enable flags from the main control to the respective blocks. The I²C script generation and card readout are specified as "start_i2c_main" and "enable_scan_slots_data", used later by the "I2Cslots" and "ReadDataBlocks" modules to initiate their processes (lines 118 and 126, respectively). The backend transmission process flags are instantiated in lines 110 and 135 for the "enable_tolink_chain" and in lines 111 and 141 for the "enable_tolink" variables.

C The OPC UA server configuration

This section exemplifies the configuration file described in Section 4.6.2.1. It is used by the monitoring OPC UA server running on the PYNQ board and it is supposed to detail each variable that should be updated on the server.

In XML scripting, there is the root-variable, being the most generic and comprehensive variable on the file. Then the root can have multiple child-variables, each one capable of having child-variables and so on, successively, in a tree hierarchy. The attributes are variables' specifications, such as their name, ID or serial number. For the card channels there is also the value attribute, which assigns a value to the channel.

The Code C.1 is an example of the "config.xml" file referred before. It only uses one channel for each card type for demonstration, excluding specific variables. Section 4.6.2.1 has more details of the conventions applied to the file, as defined for this project. However, this section tends to clarify specific excerpts from the file.

Code C.1: Example of the XML configuration file ("config.xml") for the OPC UA server. It describes the LISSY crate layout of the cards and each monitored variable. Only the channel number 1 of each card and the specific type of variables are represented. The cards serial numbers are demonstrative.

```
1 <MonitorController name="MonitorController" controllerID="1">
2   <Crate name="Crate1" crate_id="1">
3
4     <MonFPGA name="MonFPGA" slot_id="1" serial="aa-bb-cc-dd-ee-ff">
5       <CalculatedVariable name="alive"
6         value="cumulativeDifferenceWithOverflow(
7           $parentObjectAddress(numLevelsUp=0).heartbeat,
8           65535)/25"/>
9     </MonFPGA>
10
11
12   <ILockFPGA name="ILockFPGA">
13   </ILockFPGA>
14
15   <T2I name="T2I_S3" slot_id="3" channels="12" serial="gg-hh-ii-jj-kk--ll">
16     <CalculatedVariable name="Cal_A"
17       value="1/298.15 - 1/3435 * log(10000)"/>
18     <CalculatedVariable name="Cal_B"
19       value="1/3435"/>
20     <CalculatedVariable name="Cal_C"
21       value="0"/>
```

C. THE OPC UA SERVER CONFIGURATION

```
22         <CalculatedVariable name="C01_temp"
23             value="steinhartHart(
24                 $parentObjectAddress(numLevelsUp=0).t2i_adc1_chan1,
25                 $parentObjectAddress(numLevelsUp=0).Cal_A,
26                 $parentObjectAddress(numLevelsUp=0).Cal_B,
27                 $parentObjectAddress(numLevelsUp=0).Cal_C)"/>
28         <CalculatedVariable name="hi_temp"
29             value="steinhartHart(
30                 $parentObjectAddress(numLevelsUp=0).t2i_adc5_chan1,
31                 $parentObjectAddress(numLevelsUp=0).Cal_A,
32                 $parentObjectAddress(numLevelsUp=0).Cal_B,
33                 $parentObjectAddress(numLevelsUp=0).Cal_C)"/>
34         <CalculatedVariable name="err_temp"
35             value="steinhartHart(
36                 $parentObjectAddress(numLevelsUp=0).t2i_adc5_chan2,
37                 $parentObjectAddress(numLevelsUp=0).Cal_A,
38                 $parentObjectAddress(numLevelsUp=0).Cal_B,
39                 $parentObjectAddress(numLevelsUp=0).Cal_C)"/>
40         <CalculatedVariable name="board_temp"
41             value="steinhartHart(
42                 $parentObjectAddress(numLevelsUp=0).t2i_adc5_chan4,
43                 $parentObjectAddress(numLevelsUp=0).Cal_A,
44                 $parentObjectAddress(numLevelsUp=0).Cal_B,
45                 $parentObjectAddress(numLevelsUp=0).Cal_C)"/>
46
47         <CalculatedVariable name="C01_hi"
48             value="getBit(
49                 $parentObjectAddress(numLevelsUp=0).THi, 0)"/>
50
51         <CalculatedVariable name="C01_err"
52             value="getBit(
53                 $parentObjectAddress(numLevelsUp=0).TErr, 0)"/>
54     </T2I>
55
56     <DigitalIO name="DigitalIO_S5" slot_id="5" channels="20" serial="mm-nn-oo-pp-qq-rr">
57         <CalculatedVariable name="C01_out"
58             value="getBit(
59                 $parentObjectAddress(numLevelsUp=0).out_a, 0)"/>
60     </DigitalIO>
61
62     <GSS name="GSS" slot_id="7" serial="ss-tt-uu-vv-ww-xx">
63         <CalculatedVariable name="IN_01"
64             value="getBit(
65                 $parentObjectAddress(numLevelsUp=0).input, 0)"/>
66
```

```

67         <CalculatedVariable name="OUT_01"
68             value="1 - getBit(
69                 $parentObjectAddress(numLevelsUp=0).output,
70                 0)"/>
71     </GSS>
72 </Crate>
73 </MonitorController>

```

On line 1 is the root of the file, the "MonitorController", which should comprise every variable to be monitored. The first variable in the XML tree hierarchy is the crate, which includes the cards installed in that crate. This allows to set multiple crate layouts in one single configuration file. The crate only contains its name and ID indicator, corresponding to the crate number, also specified in its name attribute. Inside the crate variable each card installed should be defined and described with its attributes and the associated child-variable, such as their channels.

On the Mon-FPGA variable, the only child-variable that belongs to it is the "alive" variable, which comes from a calculation of the difference between successive Mon-FPGA "heartbeats". The function that estimates this difference is named "cumulativeDifferenceWithOverflow" (line 6 of Code C.1), which is defined in the secondary server file, the "CalculatedVariable". It is dedicated to functions which are called on the main design file of the server. The connection between these two types of design files is executed with the XML variable declaration, <CalculatedVariable>, at every variable that results from the secondary file. Section 4.6.2.1 details this file dependency, such as the "alive" and "heartbeat" purposes. The function not only calculates the difference of an input value, but also verifies if the output suffered from numeric overflow. The following Code C.2 displays the body of the function.

Code C.2: Body of the "cumulativeDifferenceWithOverflow" function used to determine the difference between consecutive values and to verify if there is numeric overflow.

```

1  parser.DefineFun("cumulativeDifferenceWithOverflow", [(double x, double o){
2      static double oldVal = 0;
3      static double newVal = 0;
4      double oldDiff = newVal - oldVal;
5      oldVal = newVal;
6      newVal = x;
7      return (newVal + o <= oldDiff + oldVal) ? (o + newVal - oldVal) :
8          (newVal - oldVal);
9  });

```

The parser.DefineFun is a specific set of function definition commands according to the framework used for the server, built with Quasar. The first argument, is the name of the function and then there are the other arguments. The function uses 2 double variables, a numeric data type in C++ scripting. The x represents the current value on which the difference is going to be calculated and the o represents the digital FSR of x. Comparing to the arguments used in Code C.1 (lines 7 and 8), the x is the current "heartbeat" value of the Mon-FPGA and the o is the "heartbeat" counter FSR of 16 bits (65535). On Code C.2, two variables are initiated as static with 0, the newVal and oldVal (lines 2 and 3). The static statement allows to attribute a value to a variable only once, in this case at the beginning, used to assign an initial value. Then, the oldDiff gets the difference value between these two variables (line 4), representing a past difference, since it did not involve the current x value. The 2 variables are sub-

C. THE OPC UA SERVER CONFIGURATION

sequently shifted, assigning the past value x , previously stored at `newVal` to the `oldVal` (line 5), while `newVal` gets the current input value x (line 6). The return line executes an if condition to verify if the current difference (unlike `oldDiff`) between successive values will not overflow the maximum possible value, o . Resuming, in lines 4 and 5 it deals with past x values, on line 6 it is where the current value is introduced and in line 7 it is where the actual current difference is returned as the output of the if condition.

The overflow is expected because the counting of the "heartbeats" is always summing, so it should increase successively, eventually exceeding the maximum possible value, this being o . Basically, the difference of "heartbeats" has the purpose of estimating the new counting value and not comparing the new one with the previous value. Additionally, although the "heartbeat" is given by the successive sum of data update cycle¹, it is taken as only the new cycles and not the sum with the last cycles. With this said, the if loop condition can be explained as: if the current value (`newVal`) plus the range of the counter (o) is less or equal then the previous difference plus the past value (`oldVal`), it is considered an overflow situation, with the final output being the overflowed correction. So the current difference (at this point, the `newVal - oldVal` is not the same as `oldDiff`) plus the overflow value to actually return a non overflowed result. If the previous condition is not satisfied (it does not verify an overflow), the current difference is directly returned as the output of the function, without overflow correction. The difference of the successive "heartbeats" is then normalized by a preferred factor of 25 (Code C.1, line 8), because the system was designed to execute 25 "heartbeats" per server update.

For the T2I module specification (Code C.1, lines 15 to 54) it is used the "steinhartHart" function, which, as expected, applies the Steinhart-Hart equation to the received digital word from the T2I ADCs. The function also comes from the file "CalculatedVariable". This equation is explained in Section 4.6.2.1 and the digital conversion to the NTC resistance is in Section 6.1. The Code C.3 displays the function's body.

Code C.3: Body of the "steinhartHart" function used to estimate the NTC temperature, by using the digital word of the voltage converted by the ADC.

```
1 parser.DefineFun("steinhartHart", [(double adc_counts, double A, double B, double C = 0) {
2     constexpr double ADC_RANGE = 32767; // 15-bit
3     constexpr double R_A = 10000;
4
5     double u_ntc = (adc_counts / ADC_RANGE); ;
6     if (1.0 <= u_ntc) return -273.15; // Broken Cable/Open Channel
7     double R_ntc = u_ntc * R_A / (1 - u_ntc);
8
9     double sh = 1 / (A + B * log(R_ntc) + C * pow(log(R_ntc), 3));
10
11     return sh - 273.15; // degC conversion
12 }]);
```

The function takes as arguments the NTC voltage converted into a digital word by the ADC, `adc_counts`, and the Steinhart-Hart equation coefficients (Code C.1, lines 16 to 21. Lines 2 and 3 define fixed double values: the `ADC_RANGE` and `R_A`, for the digital FSR of the digital word reading (for 15 bits, 32767) and for the NTC series resistor of 10 kΩ. Line 5 applies the NTC voltage digital word

¹Explained in Section 4.6.2.1.

normalization according to the digital FSR and the resistance estimation, equation 6.6, is applied at line 7. Its output is used for the temperature calculation at line 9, applying the Steinhart-Hart equation and the equation coefficients defined. The $\log$ functions correspond to the natural logarithm, $\ln$. The temperature is then converted to Celsius in line 11. The if condition at line 6 tests the open channel or "Broken Cable" signal, meaning an infinite resistance at the T2I input channel. By applying this value at the Steinhart-Hart equation, it results in $1/\ln(x)$ with $x \rightarrow \infty$, which cannot be calculated. So, the if condition evaluates when the normalization of the digital word reaches 1.0 or higher, meaning the open channel situation. This happens because with an open channel the input circuit forces 2.5 V at the input of the ADC, which is the same as the reference voltage and equivalent to the digital FSR (ADC_RANGE). Figure C.1 illustrates the T2I input channel to the ADC circuit when the channel is open ("Broken Cable").

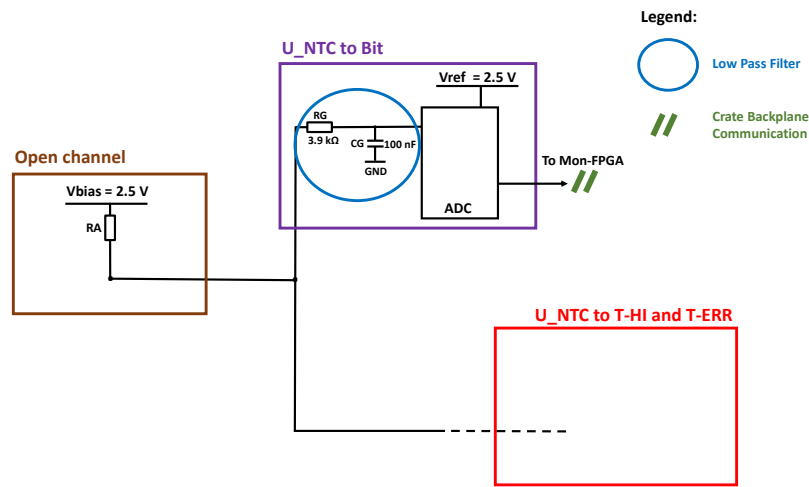


Figure C.1: Illustration of one T2I input channel and the path to the ADC when the channel is open. There is, basically, a direct connection from V_{bias} to the input of the ADC.

In the Code C.1, the lines from 22 to 27, 28 to 33, 34 to 39 and 40 a 45 represent T2I child-variables with the estimated temperature using the "steinhartHart" function, for the input channel 1, both limit OP-AMPs reference (Section 4.6.2.1) and for the on-board sensor reading, respectively. The t2i_adc1_chan1 designation on the temperature child-elements refers to the respective channel ADC and its input. For this case, the digital word comes from the ADC number 1 and its channel 1, when the on-board temperatures come from the ADC number 5 and from its channel 1,2 and 4 respectively.

From line 47 to 49 and 51 to 53 are represented the child-variables corresponding to the channel 1 digital words for both limits of operation, the "C01_hi" and "C01_err" for the "ERROR" and "Broken Cable" signals, respectively. These variables use the "getBit" function (Code C.4), from the file "CalculatedVariable".

Code C.4: Body of the "getBit" function used to collect a bit in a specific bit position from a decimal representation of an I^2C bus.

```

1 parser.DefineFun("getBit", [(double x, double y)
2     {return (double)(
3         (((unsigned long long)x) >> ((unsigned long long)y))
4         & 1);
5     }]);

```

C. THE OPC UA SERVER CONFIGURATION

This function takes 2 double arguments as decimal numbers. It returns the bit from the x decimal representation of a bit bus, at the y position. The unsigned long long command converts the decimal numbers of x and y into unsigned 64-bit integer (long represent a 32-bit long word, thus long long gives a 64-bit sequence) and the >> command does a right shift of y number of bits of the x bit bus from the MSB to the LSB (line 3). The & 1 (line 4) executes a bitwise AND operation between the shifted x bus and the bit 1, basically isolating the LSB of the bus, which corresponds to the position y of the original bus. Figure C.2 exemplifies a right bit shifting process and the resulted LSB from the bitwise AND operation.

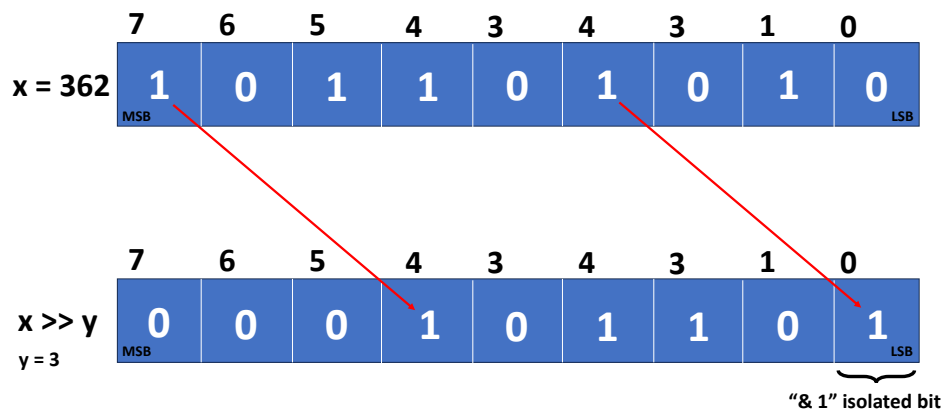


Figure C.2: Right shift on a 8-bit bus. The x bus corresponds to 362 in decimal number and y the decimal 3. The >> allows to shift y bit positions (3) to the right of the x bus and the bitwise operation, on Code C.4 line 4, isolates the new LSB.

So, as example, in Code C.1, from line 47 to 49, it is defined the variable C01_hi, which corresponds to the bit at the position 0 (it does not shift the bus and collects the LSB) in the I²C bus, given by \$parentObjectAddress(numLevelsUp=0).THi. The bit position at the bus identifies the channel number, the same way as channel 2 is located at the bit position 1 of the I²C bus (designated as THi), channel 3 at the bit position 2 and so on.

The remaining cards apply the "getBit" function the same way, to retrieve the respective channel bit position on the I/O expanders I²C bus. This is executed at lines 57 to 59, 63 to 65 and 68 to 70, for the OUT card and GSS card input and output channels, respectively. The GSS output has the singularity of returning the inverse bit value to the server. It does this by subtracting to 1 the gathered bit value from the bus (line 68).

D Design of the HIS interface build

D.1 The "CreateDpts.ctl" script

Code D.1 represents some segments of the "createDpts.ctl" file, specifically of the creation of the **T2I** card type and its **DPEs**, only for the temperature readings on each channel. The function `formatChannel` (lines 1 to 5) is used under the `createDptT2I` (lines 8 to 21), in a for loop to create sequentially formatted and numbered **DPEs** for each **T2I** channel (lines 14 to 20). It creates 2 digits padded channels as `CXX`, where `XX` is the channel number. The channels' **DPEs** form under the appending of the string `"HGTDT2I"` (line 10) with the channels padded number and the string `"_temp"` into an array of strings (`makeDynString`), at lines 15 and 16. As it can be verified, the for loop is fixed at 16 cycles (line 14), which is the maximum number of channels for any **T2I** version. Even if the used card has 12 channels, as it was the case in this **HIS** implementation, the project will always have 16 **DPEs**, which allow to fit every **T2I** sub-type card possible. However, only 12 of the created **DPEs** would establish a connection with the **OPC UA** server variables. This connection and value attribution is only achieved by the creation of the **DPTs** using the `"configParser.ctl"`. As for now, it only has the scheme for 16 channels, with no values. The same method is applied for every **OUT** card created in the project. It has 32 channels in every situation, although not every channel establishes connection and captures a value reading from the server. The **DPEs** are only a scheme for the channels that the **DPTs** can afford, not meaning that all **DPEs** actually establishes a connection. It is worth mentioning that the project's objective is to be flexible and allow different **LISSY** layouts. Then the **T2I DPT** dedicated function is called in a generic function (`createDpts`, from line 23 to 34), on line 27, creating the **DPTs** and, consequently, the **DPEs**. The functions dedicated to the **Mon-FPGA**, **ILK-FPGA**, **GSS** and **OUT** cards are designated as `createDptMonFPGA`, `createDptILockFPGA`, `createDptGSS` and `createDptDigitalIO`. There were also created functions to attribute formatted channels designations for the **GSS**, similarly as the `formatChannel`. They are called `formatInput` and `formatOutput` for the inputs and outputs channels of this card, respectively.

D. DESIGN OF THE HIS INTERFACE BUILD

Code D.1: Code segment of the "createDpts.ctl" script for the creation of the DPTs and the DPE declaration.

```
1  string formatChannel(int ch) {
2      string output;
3      sprintf(output, "C%02d", ch); // string format CXX
4      return output;
5  }
6  (...)
7
8  bool createDptT2I()
9  {
10     dyn_dyn_string xxdepes;
11     dynAppend(xxdepes, makeDynString("HGTDT2I", ""));
12     (...)
13
14     for (int i = 1; i <= 16; i++) {
15         dynAppend(xxdepes, makeDynString("",
16             formatChannel(i) + "_temp"));
17         (...)
18
19     (...)
20     }
21 }
22
23 int createDpts (string dptFilter=".*")
24 {
25     {int result = regexpIndex(dptFilter, "T2I");
26         if (result >= 0)
27         {if (!createDptT2I())
28             return 1;
29         }
30         else
31         {DebugN("DPT T2I not covered by provided dptFilter, skipping");
32         }
33     }
34 }
35 (...)
```

D.2 The panel of the creation of the DPs

This section details the functionality of each button in the panel dedicated to the creation of the project's DPTs and their DPs (Figure 5.3). Each button executes a dedicated Ctrl script, which are in charge of, in turn, executing the scripts overviewed in Section 5.2.3.

D.2.1 The "createDpts.ctl" button

The Code D.2 displays the Ctrl script implemented at the "Clicked" event execution method under the "DPTs creation" button of the panel of Figure 5.3.

Code D.2: Script for the "DPTs creation" button, used to create the DPTs of specific or of all available types.

```

1  #uses "createDpts.ctl"
2  main(mapping event)
3  {
4      Debug("=== Called createDpts(" + CREATE_DP_TYPES.text + ") ===");
5      createDpts(CREATE_DP_TYPES.text);
6  }
```

The line 1 explicitly tells that what will be written might use specific functions found in the "createDpts.ctl" script. The `main(mapping event)` line determines that the following function's execution is driven by a specific event, determined earlier to be the "Clicked" method. Line 4 generates a debugging message in the Log Viewer tool, displaying the specified types to be created, thereby assisting in error detection. The text in the text box, dedicated to specifying which DPT the system should create, is introduced by the textbox's name, in this case, 'CREATE_DP_TYPES,' along with the flag text for proper utilization of the written content and is used as the `createDpts` function from the "createDpts.ctl" script.

D.2.2 The "configParser.ctl" button

The Code D.3 displays the defined code for the second button of the panel of Figure 5.3, used for the DPs creation, according to the introduced "config.xml" file directory.

Code D.3: Script for the DPs creation button, that imports the directory of the "config.xml" file for that purpose.

```

1  #uses "configParser.ctl"
2  main(mapping event)
3  {
4      parseConfig(
5          CONFIG_DIR.text,
6          (...)
7      );
8  }
```

Once more, line 1 declares the use of specific functions from the "configParser.ctl" script and from line 4 to 7, its main function, the `parseConfig`, is initialized with its arguments, one being the "config.xml" directory. With it, the DP sequential creation process, detailed in Section 5.2.3 and Figure 5.1, is initiated. The file directory is introduced in the code by calling the text written in the "CONFIG_DIR" text box, right in front of the second button (/home/config.xml on the panel of Figure 5.3). This code is also controlled by the "Clicked" method.

D. DESIGN OF THE HIS INTERFACE BUILD

D.2.3 "DPs delete" button

Code D.4 displays an example of the T2I card's DPs and then its DPT removing.

Code D.4: Script for the DPs removing button, which executes the procedure sequentially, first removing the DPs and then the DPTs. Only displays the removing of the T2I DPs and DPT.

```
1  main(mapping event)
2  {
3      string dpName;
4
5      for (int i = 1; i<4; ++i){
6          //Delete T2I card DPs
7          for (int j = 2; j<16; ++j){
8              dpName = "MonitorController/Crate" + i + "/T2I_S" + j + ".";
9              if (dpExists(dpName)){
10                 DebugN("DP exists ", dpName, " deleting...");
11                 dpDelete(dpName);
12             }
13         }
14         (...)
15     }
16
17     //Delete DP Types
18     dpTypeDelete("ITkT2I");
19     (...)
20 }
```

The code loops around 4 cycles, to remove the DP of 4 crates (lines 5 to 15), which is the planned number of crates to be used by the HIS final implementation. In this case, it only deals with 1 crate layout and the remaining cycles are simply neglected, since those crate's DPs do not exist. The iteration number of the crate is used in the DP "address name" (5), designated as MonitorController/CrateX/T2I_SY. (line 8), given that MonitorController is the group of variables monitored by the Mon-FPGA, CrateX is the variable's crate name that joins the cards of the crate number X, T2I_SY the T2I card installed in the slot Y¹ and the dot operator at the end is a marker between the DP "leaf-variable", meaning it deletes every DP of that DPT. For each T2I card DPT, the code removes 16 channels's DP, allowing to remove the DPs of both card types (T2I-12 and T2I-16), regardless of what card is installed (lines 7 to 13). This loop first verifies if the DP with the "address name" of dpName actually exists (lines 9 to 12) and if so, it prints a message in the Log Viewer telling which DP was removed (line 10). The Ctrl function used to remove the specified DP is called dpDelete (line 11). The T2I DPT is then removed by the function dpTypeDelete (line 18). This process is executed for every channel and card type of the crate.

¹The naming convention is detailed in Section 4.6.2.1.

D.3 The DPs' values display functions

The event-control interface connection with the respective DP is achieved using a WinCC function called "dpConnect". It enables the connection of the panel's variables with the DPs values, only when it changes. By detecting a change in the value of the DP in question, it applies the above functions to display the values according to the function's specifications. The "ATLITKLISSY_statusCircleValueChange" defines the colour of the interlock signals in each I/O card channel in dedicated circles (red for "INTER-LOCK", green for "OK" state). The "ATLITKLISSY_tempBoxValueChange" displays the estimated temperature of the T2I channels, on the text box where it is implemented. Code D.5 exemplifies the use of the dpConnect function while using these functions to display the binary "ERROR" signal as green or red, according to the type of signal and to display the estimated temperature of the T2I channel, respectively. The dpConnect needs the DP "address name", which is built through \$dpT2ICard + "." + \$channel + "_temp" and \$dpT2ICard + "." + \$channel + "_temp", which concatenates the \$dpT2ICard (the T2I card designation as T2I_SX) and \$channel (the T2I channel format as CXX) with the string "_temp" or "_hi".

Code D.5: Example of use of the "dpConnect" function and the ITk "ATLITKLISSY_statusCircleValueChange" and "ATLITKLISSY_tempBoxValueChange" functions to attribute the DP value change of a specific parameter.

```

1 i = dpConnect("ATLITKLISSY_statusCircleValueChange",$dpT2ICard + "." + $channel + "_hi");
2
3 i = dpConnect("ATLITKLISSY_tempBoxValueChange", $dpT2ICard + "." + $channel + "_temp");

```


E Uncertainties

All the deduced uncertainties of derived variables (or parameters, or expressions) were obtained through the propagation of the uncertainties of the basic components influencing them. This chapter intends to describe their deduction. The propagation of uncertainties is given by the quadratic sum of the product of each variable's partial derivative and its uncertainty, as presented in Section 6.1.

E.1 Steinhart-Hart's temperature

Given the Steinhart-Hart's equation in 4.1, it is possible to deduce its uncertainty as a function of $\beta = (3435 \pm 35)$ K, $R_{25} = (10 \pm 0.1)$ k Ω and the resistance of the sensor or its equivalent resistance simulated by resistors, such as the reference RH_L, RE_L and R_{eq} from the T2I hysteresis tests. So, applying the partial derivatives with respect to these variables (equations E.1, E.2 and E.3, one obtains the temperature total uncertainty, ΔT , in equation E.4.

$$\frac{\partial T}{\partial \beta} = T^2 \cdot \frac{1}{\beta^2} (\ln(R) - \ln(R_{25})) \quad (\text{E.1})$$

$$\frac{\partial T}{\partial R_{25}} = -T^2 \cdot \frac{1}{\beta R_{25}} \quad (\text{E.2})$$

$$\frac{\partial T}{\partial R} = -T^2 \cdot \frac{1}{\beta R} \quad (\text{E.3})$$

$$\Delta T = T^2 \sqrt{\frac{1}{\beta^4} (\ln(R) - \ln(R_{25}))^2 (\Delta \beta)^2 + \frac{1}{\beta^2} \left(\frac{1}{R_{25}} (\Delta R_{25})^2 + \frac{1}{R^2} (\Delta R)^2 \right)} \quad (\text{E.4})$$

E. UNCERTAINTIES

E.2 Hysteresis voltage

The hysteresis range on each **OP-AMPs** is explained in Section 6.1. Its is obtained as a range of possible U_{NTC} values, which limits each state transition.

E.2.1 "Broken Cable" hysteresis

For the "Broken Cable" transitions, it is possible to deduce either transition with the equation 6.13. However, each transition ends up as U_{NTC} as a function of V_{BC} , BC , RD and RC :

$$U_{NTC} = \frac{V_{BC}(RD + RC) - BC \cdot RC}{RD} \quad (E.5)$$

The partial derivatives with respect to V_{BC} , BC , RD and RC are shown in equations E.6, E.7, E.8 and E.9, respectively. Despite when $BC = 0$ V when initially in "OK" state, this output has an associated uncertainty of 20 mV, while $BC = 3.3$ V has 60 mV. RC and RD have both a tolerance of 1%.

$$\frac{\partial U_{NTC}}{\partial V_{BC}} = \frac{RD + RC}{RD} \quad (E.6)$$

$$\frac{\partial U_{NTC}}{\partial BC} = -\frac{RC}{RD} \quad (E.7)$$

$$\frac{\partial U_{NTC}}{\partial RD} = (BC - V_{BC}) \frac{RC}{RD^2} \quad (E.8)$$

$$\frac{\partial U_{NTC}}{\partial RC} = \frac{V_{BC} - BC}{RD} \quad (E.9)$$

Which results in ΔU_{NTC} for each "Broken Cable" transition:

$$\Delta U_{NTC} = \sqrt{\left(\frac{RD + RC}{RD}\right)^2 (\Delta V_{BC})^2 + \left(\frac{RC}{RD}\right)^2 (\Delta BC)^2 + \left((BC - V_{BC}) \frac{RC}{RD^2}\right)^2 (\Delta RD)^2 + \left(\frac{V_{BC} - BC}{RD}\right)^2 (\Delta RC)^2}$$

E.2.2 "ERROR" hysteresis

The "ERROR" **OP-AMP**, or HI , also has, for both transitions, the same initial equation given by 6.16, which gives $U_{NTC}(V_{HI}, HI, RF, RE)$. The partial derivatives are expressed by E.10, E.11, E.12 and E.13, respectively.

$$\frac{\partial U_{NTC}}{\partial V_{HI}} = \frac{RF}{RF + RE} \quad (E.10)$$

$$\frac{\partial U_{NTC}}{\partial HI} = \frac{RE}{RF + RE} \quad (E.11)$$

$$\frac{\partial U_{NTC}}{\partial RF} = (V_{HI} - HI) \frac{RE}{(RF + RE)^2} \quad (E.12)$$

$$\frac{\partial U_{NTC}}{\partial RE} = (HI - V_{HI}) \frac{RF}{(RF + RE)^2} \quad (E.13)$$

Giving ΔU_{NTC} for each "ERROR" transition as:

$$\Delta U_{NTC} = \sqrt{\left(\frac{RF}{RF + RE}\right)^2 (\Delta V_{HI})^2 + \left(\frac{RE}{RF + RE}\right)^2 (\Delta HI)^2 + \left(\frac{RE(V_{HI} - HI)}{(RF + RE)^2}\right)^2 (\Delta RF)^2 + \left(\frac{RF(HI - V_{HI})}{(RF + RE)^2}\right)^2 (\Delta RE)^2}$$

E.3 Voltage divider

The voltage divider equation is widely used throughout this thesis to either determine the voltage at a given resistance or vice versa. For instance, to determine the reference voltage for each temperature limit, as a result of the R_REF circuitry V_{BC} and V_{HI} (which uses equation 6.1 directly). Or, on the other hand, to determine the resistance (R_{eq}) for each hysteresis transition limit or the the resistance equivalent of the measured voltage during the testing of the hysteresis (which uses the arrangement of equation 6.1 in order of R_{NTC}).

E.3.1 $U_{NTC}(R_{NTC}, RA, V_{bias})$

Taking as reference the voltage determination at the terminals of R_{NTC} , in the T2I channel input circuit $U_{NTC}(R_{NTC}, RA, V_{bias})$ in the equation 6.1, the partial derivatives are E.14, E.16 and E.16, respectively, and the uncertainty is given by E.17.

$$\frac{\partial U_{NTC}}{\partial R_{NTC}} = -V_{bias} \frac{R_{NTC}}{(R_{NTC} + RA)^2} = -\frac{U_{NTC}}{R_{NTC} + RA} \quad (E.14)$$

$$\frac{\partial U_{NTC}}{\partial RA} = -V_{bias} \frac{RA}{(R_{NTC} + RA)^2} \quad (E.15) \quad \frac{\partial U_{NTC}}{\partial V_{bias}} = \frac{R_{NTC}}{R_{NTC} + RA} \quad (E.16)$$

$$\Delta U_{NTC} = \sqrt{\left(\frac{U_{NTC}}{R_{NTC} + RA}\right)^2 (\Delta R_{NTC})^2 + \left(\frac{V_{bias} RA}{(R_{NTC} + RA)^2}\right)^2 (\Delta RA)^2 + \left(\frac{R_{NTC}}{R_{NTC} + RA}\right)^2 (\Delta V_{bias})^2} \quad (E.17)$$

E.3.2 $R_{NTC}(U_{NTC}, RA, V_{bias})$

Using the voltage divider equation 6.1 to obtain $R_{NTC}(U_{NTC}, RA, V_{bias})$ i.e. rather as a function U_{NTC} , RA and V_{bias} , as shown by equation 6.2, it is possible to deduce the uncertainty by the partial derivatives shown in E.18, E.19 and E.20, respectively, and the resulting uncertainty on equation E.21.

$$\frac{\partial R_{NTC}}{\partial U_{NTC}} = V_{bias} \frac{RA}{(V_{bias} - U_{NTC})^2} \quad (E.18)$$

$$\frac{\partial R_{NTC}}{\partial RA} = \frac{U_{NTC}}{V_{bias} - U_{NTC}} \quad (E.19) \quad \frac{\partial R_{NTC}}{\partial V_{bias}} = -U_{NTC} \frac{RA}{(V_{bias} - U_{NTC})^2} \quad (E.20)$$

$$\Delta R_{NTC} = \sqrt{\left(\frac{V_{bias} RA}{(V_{bias} - U_{NTC})^2}\right)^2 (\Delta U_{NTC})^2 + \left(\frac{U_{NTC}}{V_{bias} - U_{NTC}}\right)^2 (\Delta RA)^2 + \left(\frac{U_{NTC} RA}{(V_{bias} - U_{NTC})^2}\right)^2 (\Delta V_{bias})^2} \quad (E.21)$$

F Electronic components and cards

F.1 The ADS1119 ADC

Besides being defined by the digital [FSR](#), an [ADC](#) can also be characterized by its voltage [FSR](#), as the voltage range that the [ADC](#) can deal with its inputs. For the chosen [ADC](#), the voltage [FSR](#) is given by $V_{ref} = REFP - REFN$, where REFP and REFN are the positive and negative reference voltages of the component. From the schematic of [Figure F.1](#) it is possible to verify that REFP=2.5 V and REFN=0 (GND).

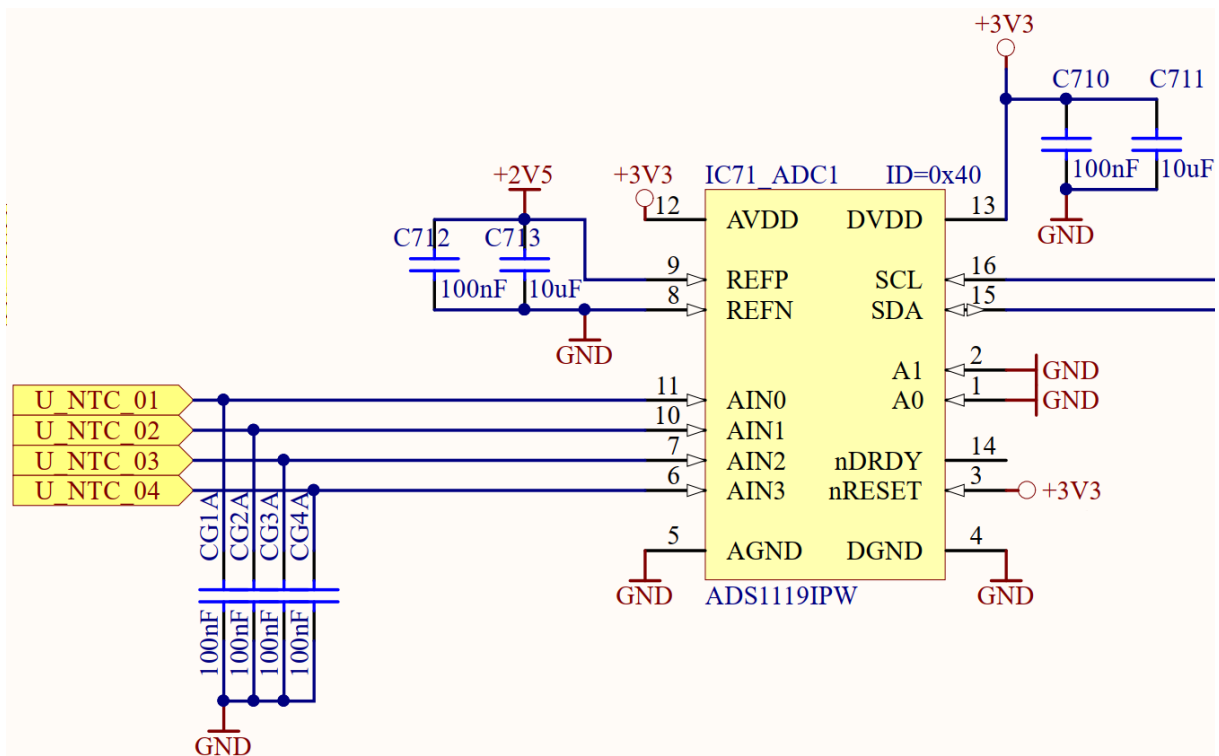


Figure F.1: Schematic of on-board [T2I](#) ADS1119 [ADC](#)[\[8\]](#) with the input channels 1 to 4 as "U_NTC_0X", X being the channel number. The [ADC](#) is powered with 3.3 V and has a reference voltage of 2.5 V (REFP–REFN). The [SCK](#) and [SDA](#) lines are the [I²C](#) output of the signal conversions, communicated to the [Mon-FPGA](#) through the backplane. The "CGNA" capacitors belong to four passive low pass filters, at the input of each port. This is part of the schematic of the [T2I](#) card available (ITKInterlock_T2I_12ch_0421) [\[21\]](#).

F.2 The LT3042 voltage regulator

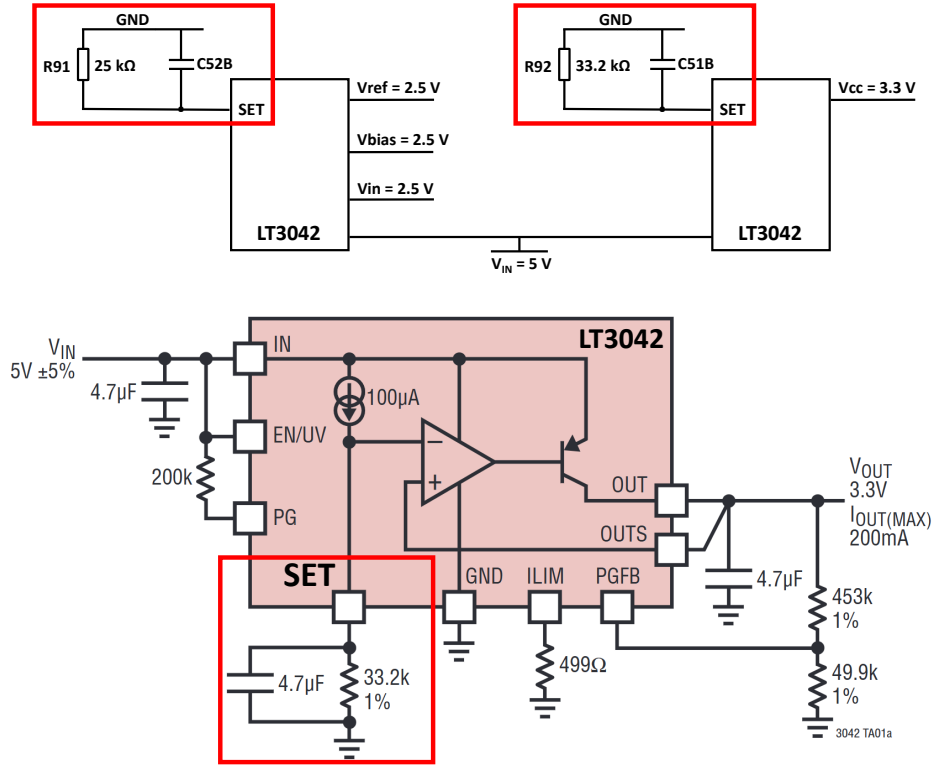


Figure F.2: An LT3042EMSE [47] voltage regulator typical application and the T2I application. The device is used to provide 2.5 V to the reference voltages of the card's input circuit, ADCs and reference circuits for the OP-AMPS, and 3.3 V to the positive power supply of the OP-AMPS. On the figure's upper side is the representation of the regulator application on the T2I board for both cases, respectively. On the centre is a typical application of the component taken from its datasheet. The red frame compares the SET pin of the component in the T2I application and in the typical application. This SET-GND port was used to configure the regulated voltage with the resistors' values.

According to the LT3042EMSE datasheet [47] the SET pin of the regulator enables an output voltage configuration, by mounting a precision resistor from the SET pin to the GND. The component injects a precision current of $100 \mu\text{A} \pm 1\%$ through the resistor. The LT3042's output voltage is determined by $V_{SET} = I_{SET} \cdot R_{SET}$, where I_{SET} and R_{SET} are the current and resistor installed in the SET port. For Figure F.2 which has a datasheet typical application, the output voltage would be $33.2 \text{ k}\Omega \cdot 100 \mu\text{A} = 3.32 \text{ V}$. The T2I cards are fed with 5 V. The typical application is similar to the T2I case it provides a V_{CC} of 3.3 V to supply the OP-AMPS (figure's upper right), with $R_{92} = 33.2 \text{ k}\Omega \pm 0.1\%$. Since it is a supply voltage and not a precise reference, the 3.32 V is acceptable instead of 3.3 V. For the reference voltage of 2.5 V (figure's upper left), the $R_{91} = 25 \text{ k}\Omega \pm 0.1\%$ resistor ensures the required precision for the reference voltage of the ADC, for the input channel circuit and for the reference circuit (R_{REF}) of the OP-AMPS. As referred in the component datasheet, Adding a capacitor from SET to GND improves noise, PSRR, and transient response, as it is the case for the C52B and C51B capacitors.

The accuracy of the regulated output voltage for both cases can be given by the output voltage offset of the manufacturer, which is 2 mV for $2 \text{ V} < V_{in} < 20 \text{ V}$, $0 \text{ V} < V_{OUT} < 15 \text{ V}$, when in the application there is $V_{in} = 5 \text{ V}$, $V_{OUT} = 2.5 \text{ V}$ and $V_{OUT} = 3.32 \text{ V}$.

