UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

TITULO: MONITORING AND CONTROL SYSTEM BASED ON MICROCONTROLLER FOR THE POWER SUPPLY OF A COMPLEX SYSTEM

AUTOR: Alberto Ibarrondo Luis

AÑO: 2015

TÍTULO: MONITORING AND CONTROL SYSTEM BASED ON MICROCONTROLLER FOR THE POWER SUPPLY OF A COMPLEX SYSTEM

AUTOR: Alberto Ibarrondo Luis

TUTOR: Jesús Grajal de la Fuente

DEPARTAMENTO: Señales, Sistemas y Radiocomunicaciones

MIEMBROS DEL TRIBUNAL CALIFICADOR

PRESIDENTE: D. ALBERTO ASENSIO LÓPEZ

VOCAL: D. MATEO BURGOS GARCÍA

SECRETARIO: D. JESUS GRAJAL DE LA FUENTE

SUPLENTE: D. JAVIER GISMERO MENOYO

FECHA DE LECTURA:

CALIFICACIÓN:

DEPARTAMENTO DE SISTEMAS, SEÑALES Y RADIOCOMUNICACIONES

ESCUELA TÉCNICA SUPERIOR DE INGENIEROS DE TELECOMUNICACIÓN
UNIVERSIDAD POLITÉCNICA DE MADRID

MONITORING AND CONTROL SYSTEM BASED ON MICROCONTROLLER FOR THE POWER SUPPLY OF A COMPLEX SYSTEM

TRABAJO FIN DE GRADO

Autor:
Alberto Ibarrondo Luis

ABSTRACT

The scope of this project is the design, encoding and implementation of a software system to monitor and control the power supply of a submillimetre wave radar developed for object detection under the clothes. The power supply system, which can generate up to ten positive and two negative voltages (twelve Power Supply Units or PSUs), includes a controller subsystem that allows both electronic adjustment of every voltage output and forced shutdown. It also features a monitoring subsystem that tracks voltage and current for every output, and an alarm management subsystem.

This software, written in C language, is implemented for an in-built Coldfire 5231 microcontroller (or MCF5213), embedded in the control card of the supply system. The software program manages digital inputs and outputs, an analogic input, an I2C bus, two USB ports, analogical multiplexers, digital potentiometers, I2c to SPI bus converters and relays. Due to the developed software, the power supply system can be adjusted and monitored through an USB port. The system's hardware design and implementation is independent of this project, and is being carried out by a different project in the SSR department where the radar is being built.

The software system is hereby named as **PSUControl**. The present document describes its development process. The lowest logic details are included as comments in the source code. Such code, as well as several auxiliary documents, are published in a GitHub repository:
https://github.com/AlbertoIbarrondoLuis/PSUControl

In order to describe PSUControl system thoroughly, a requirements engineering methodology called "Requirements elicitation" is drawn upon. The methodology is divided into three steps:

1. First, the User Requirements (a.k.a. UR) are defined, where all the initial specifications for the system are detailed and categorized.
2. Second, the system is divided into pieces to hold all the functionalities necessary to fulfil the URs, forming the System Requirements (or SR), and the system is constructed accordingly.
3. Finally, tests (a.k.a. Test Cases or TC) are designed and passed for each SR, verifying the fulfilment of all requirements. To assemble all the process, a Traceability matrix is generated, where every UR is traced to several satisfactory tests.

The requirements elicitation procedure is complemented with a loose interpretation of the 4+1 architectural view model, employed to describe PSUControl in full detail. The views are portrayed in top-to-bottom order: the Deployment view (relation between HW and SW systems), the Development view (software architecture), the Process view (activity and status diagrams) and the Logic view (communication diagrams and class diagram).

All the software tests have been successfully passed, and hardware tests have been designed for all the HW elements. These tests are left pending of verification, subordinated to the system hardware completion by an independent project in a near future. Several upgrades are also proposed for future development.

KEY WORDS

Power Supply Units, Voltage adjustment, Monitoring, Alarms, MCF5213, PSUControl, Requirements, Views.

RESUMEN DEL PROYECTO

El objetivo de este proyecto es el diseño, codificación e implantación de un software de monitorización y control de las fuentes de alimentación de un radar en banda submilimétrica para detectar de objetos bajo la ropa. El sistema de alimentación, capaz de generar hasta diez voltajes positivos y dos negativos (un total de doce fuentes de alimentación), posee un subsistema controlador que permite el ajuste electrónico de los voltajes y el apagado de las fuentes. Incluye un subsistema de monitorización de los voltajes y corrientes de todas las fuentes y un subsistema de gestión de alarmas.

Este software, escrito en lenguaje "C", se ha desarrollado para un microcontrolador Coldfire 5213 (o MCF5213), incrustado en la tarjeta controladora del sistema de alimentación. El programa software gestiona entradas y salidas digitales, una entrada analógica, un bus I2C, dos puertos USB, multiplexores analógicos, potenciómetros digitales, conversores de bus I2C a bus SPI y relés. Gracias al software desarrollado, el sistema de alimentación puede ser ajustado y monitorizado a través de un puerto USB en un ordenador. El desarrollo del hardware del sistema es independiente de este proyecto, llevado a cabo en un proyecto diferente del departamento donde se construye el radar (SSR).

Este sistema software se ha definido como **PSUControl**. En el presente documento se describe el desarrollo del mismo. Los detalles de más bajo nivel se incluyen en forma de comentarios en el propio código fuente. Dicho código, junto con documentos auxiliares, está publicado en un repositorio de GitHub: *https://github.com/AlbertoIbarrondoLuis/PSUControl*.

Con el objetivo de describir detalladamente PSUControl, se emplea una metodología de ingeniería de requisitos llamada "Educción de Requisitos". Dicho método se divide en tres pasos:

1. Primero se definen los Requisitos de Usuario (UR), que detallan las especificaciones del sistema obtenidas del departamento de SSR y las clasifican.
2. A continuación el sistema se divide en elementos para cumplir los URs, conformando los Requisitos de Sistema (SR), y el sistema se construye acorde a éstos.
3. Finalmente, se definen y pasan tests (también llamados Casos de Test o TC), verificando el cumplimiento de los requisitos. Una Matriz de Trazabilidad engloba todo el proceso enlazando todos los requisitos de usuario a uno o varios tests satisfactorios.

El procedimiento de educción de requisitos se complementa con una interpretación libre del modelo de arquitectura de 4+1 vistas, empleado para describir PSUControl en detalle. Las vistas se desarrollan en orden descendente: la vista de Despliegue (relación entre subsistemas hardware y software), la vista de Desarrollo (arquitectura del software), la vista de Procesos (diagramas de actividades y estados) y la vista Lógica (diagrama de clases y diagramas de comunicaciones).

Todos los tests de software han sido ejecutados con éxito, y se han definido test para todos los elementos hardware. Estos últimos quedan pendientes de verificación, supeditados a la finalización del sistema hardware por parte de un proyecto independiente en el futuro cercano. Así mismo, se plantean una serie de mejoras opcionales para su desarrollo posterior.

PALABRAS CLAVE

Fuentes de Alimentación, Ajuste de voltaje, Monitorización, Alarmas, MCF5213, PSUControl, Requisitos, Vistas.

INDEX

1 Introduction 1

2 Hardware System Overview 2

2.1 Rectification slotcards 2

2.2 Functional slotcards (a.k.a. SFs) 3

2.3 Controller slot card 5

2.3.1 MUX HW Module 5

2.3.2 AGC HW Module 7

2.3.3 Relays 8

2.3.4 MCF5213 Microcontroller 9

2.3.5 MCF5270 Microcontroller 9

3 User Requirements 10

3.1 General 10

3.2 Monitoring 11

3.3 Configuration 11

3.4 Switch-on 12

3.5 Alarm 12

3.6 Windows 12

3.7 IP 12

4 System Requirements 13

4.1 Type Interfaces 13

4.1.1 Hardware 13

4.1.2 Software 14

4.1.3 Nielsen Heuristics for User Interface 14

4.2 Type Functional 15

4.2.1 Monitoring 15

4.2.2 Configuration 16

4.2.3 Communication 17

4.2.4 Switch-On 17

4.2.5 Alarm 18

4.2.6 Windows 18

4.3 Type Complementary 18

4.3.1 Efficiency 18

4.3.2 Reliability 19
4.3.3 Unavailability 19
4.3.4 Maintenance 19
4.4 Type Upgrade 19
4.4.1 Windows 19
4.4.2 Alarm 19
4.4.3 Monitor 19
4.4.4 Error 19
5 Traceability Matrix UR – SR 20
6 Deployment View 21
7 Development View 22
7.1 Top Layer – Main process 22
7.2 Bottommost layer – definitions and headers 24
7.3 Layer I – Libraries Package 26
7.4 Layer II – Controller Package 27
7.5 Layer III – Tests Package 28
7.5.1 Libraries Tests 28
7.5.2 Controller Tests 28
7.6 Layer IV – Interface Package 28
7.6.1 General/Interface Menu – First Level 29
7.6.2 Display Menu – Second Level 29
7.6.3 Program menu – Second Level 30
7.6.4 Libraries Menu – Second Level 30
7.6.5 Test Menu – Second Level 32
7.6.6 Config Menu – Second Level 32
8 Process View 33
8.1 Alarm Task 33
8.2 Monitor Task 34
8.3 Switch On Process 35
9 Logic View 37
9.1 Communications 38
9.1.1 RDAC Programming and Reading 38
9.1.2 Relay configuration using I2C Bus Expander 39

10 Test Cases 40
10.1 Type Interfaces 40
10.1.1 Hardware 41
10.1.2 Software 41
10.1.3 Nielsen Heuristics 41
10.2 Type Functional 42
10.2.1 Monitoring 42
10.2.2 Configuration 43
10.2.3 Switch On 43
10.2.4 Alarm 43
10.3 Type Complementary 44
10.3.1 reliability 44
11 Traceability Matrix SR – TC 44
12 PSUControl progress status 45
13 User Manual 46
13.1 System turn on and off 47
13.2 Administrator configuration 48
13.3 Common errors handling 48
14 Conclusions 49
15 References & Acronyms 50

FIGURES INDEX

Figure 1 - Systems forming the Radar 1
Figure 2 - Slotcards structure or Rack 2
Figure 3 - Transformer used to generate two 12V supply voltages 3
Figure 4 - PSUs' hardware voltage generation using a regulator 3
Figure 5 - Functional slotcard simplified hardware diagram 4
Figure 6 - Controller Slotcard hardware diagram 5
Figure 7 - MUX hardware diagram 6
Figure 8 - Magnitude channels introduced to MUX Module for monitoring 6
Figure 9 - Automatic Gain Control hardware diagram 8
Figure 10 - Relays hardware diagram 8
Figure 11 - MCF5213 connections diagram 9
Figure 12 - MCF5270 with Ethernet connector 9
Figure 13 - Deployment diagram 21
Figure 14 - Interrelations in Development diagram 22
Figure 15 - Development diagram 23
Figure 16 - Alarm arrays for PSU_TYPE and SnI_TPYE 24
Figure 17 - MTTTY Console 29
Figure 18 - Alarm Task status diagram 33
Figure 19 - Monitor Task status diagram 35
Figure 20 - Switch On activity view 36
Figure 21 - Class Diagram 37
Figure 22 - I2C and SPI Communication diagram 38
Figure 23 - Relay configuration using I2C Bus Expander communication diagram 39
Figure 24 - User Manual (I) 46
Figure 25 - User Manual (II) 47

TABLES INDEX

Table 1 - Magnitudes being measured using MUX Module 7
Table 2 - User Requirements 10
Table 3 – SR Type Interfaces/I 13
Table 4 – SR Type Functional/F 15
Table 5 – SR Type Complementary/C 18
Table 6 – SR Type Upgrade/U 19
Table 7 - Traceability matrix UR/SR 20
Table 8 - PSU_TYPE and SnI_TYPE atributes 25
Table 9 - TC Type Interfaces/I 40
Table 10 - TC Type Functional/F 42
Table 11 - TC Type Complementary/C 44
Table 12 - Traceability Matrix SR - TC 44
Table 13 - Common errors handling 48
Table 14 - Acronyms 50

1 INTRODUCTION

In the last decade, enormous developments in high frequency technology have been achieved. Those improvements have reduced the costs of microwave circuit components, thus opening a brand new set of potential applications based on the use of gigahertz waves, such as under-the-clothes' object detection.

This application, which could be implemented in airports as a less harmful substitution of X-Ray detection, was chosen several years ago by a group of researchers in the SSR department of ETSIT-UPM University. They designed and built a 300GHz Radar [1], obtaining promising results. The next step towards its future commercialization is the optimization of most of the radar's subsystems, one of them being the power supply units. This is the aim of the present TFG.

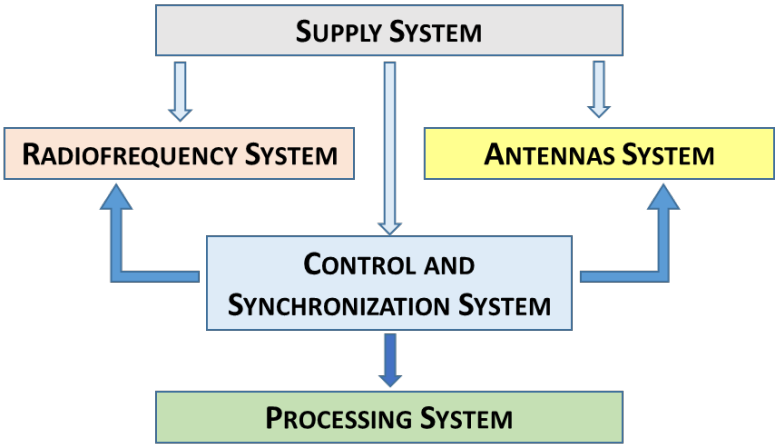


Figure 1 - Systems forming the Radar

In the original Radar, a large number of independent power supply units (similar to the ones you could find in any electronics laboratory) fed the rest of the equipment with electricity, turning out to be quite inefficient due to the use of a different power control for each. Also, temporary instabilities in the output voltage generated voltage peaks, a risk for the most sensitive Radar subsystems and a cause for concern regarding their considerable price.

The upgrade consists on a completely new and tailored supply system, where all the power supply units (a.k.a. PSUs) are generated from a single 220V A.C. input, and only one microcontroller based software controls a total of twelve voltage outputs in substitution of the former independent power supply units. This upgrade would lower the manufacturing expenses due to the use of PCB as circuitry basis, reduce the required space by using a single equipment instead of several separated elements, and greatly simplify the configuration of all the power supply units thanks to being managed by this software system.

The hardware regarding the upgraded supply system has already been designed and is in current implementation by an independent team in the SSR department, leaving the software system to be manufactured in this TFG. The whole software system will be named **PSUControl** from now on.

2 HARDWARE SYSTEM OVERVIEW

The hardware of this project has been developed entirely by other researchers. In this section only the surface will be approached, leaving out the lower details.

The radar supply system has his architecture formed by 10 PCB Boards that fit into parallel slots of a metallic structure or Rack, which will be referred to as Slot Cards (or **slotcards**) from now on. The slotcards are interconnected by means of a backplane.

The first three are the Rectification slotcards, where the alternate current is transformed into lower positive and negative voltage values (which can either be regulated or unregulated) that will serve as supply for the programmable voltage outputs as well as providing internal voltages for digital components (such as +-12V for operational amplifiers).

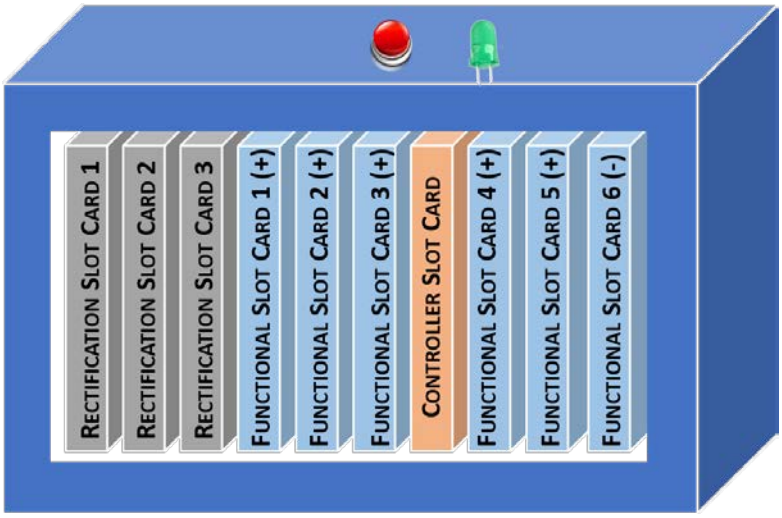


Figure 2 - Slotcards structure or Rack

The following slotcards implement the hardware required for voltage adjustment: six slotcards generating two controlled voltage outputs each (making for a total of twelve PSUs), that will be referred to as Functional slotcards (or SF); The Controller slotcard, which houses the microcontroller and several other components required for various functions.

The only external hardware components designed for interaction with the user are a button (switches on/off all the outputs) and a LED (emitting light when one or more functional slotcards are functioning).

A MicroColdFire 5213 (MCF5213) has been selected as microcontroller [2], enhanced with Netburner Software Libraries [3] to ease up the use of several low level functionalities (e.g. a2d module, i2c communication) and allowing the programmer's efforts to focus on high level logic. The software system that runs in this controller, PSUControl, will be in charge of both PSU controlling (output voltage adjustment) monitoring (react to voltage peaks) and alarm management.

2.1 RECTIFICATION SLOTCARDS

The first three slotcards, the Rectification slotcards, provide the rest of the system with eleven DC voltage supplies, some of them being regulated whilst others are left unregulated. This supplies are generated from the 220 AC current using transformers to lower the voltage and diode-based structures to rectify it (AC to DC conversion). This is the complete list of Supply Voltages:

- 42 Volts Unregulated
- 35 Volts Unregulated
- 16 Volts Unregulated
- 32 Volts Regulated
- 16 Volts Regulated
- 4 voltages with 12 Volts each, numbered from A to D.
- -16 Volts Regulated (Negative)
- -20 Volts Unregulated (Negative)

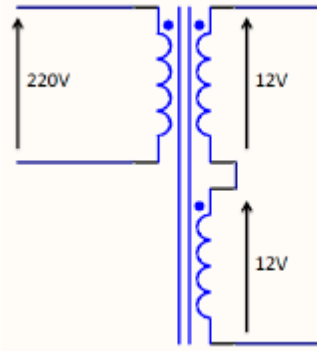


Figure 3 - Transformer used to generate two 12V supply voltages

2.2 FUNCTIONAL SLOTCARDS (A.K.A. SFS)

The functional slotcards are able to generate two voltages each, which will be stated as SFn_A and SFn_B for Functional slotcard number "n", or alternatively as **PSUs** (a much more frequent term in this project, where PSU 0 represents SF1_A, PSU 1 is SF1_B, ... , and PSU 11 is SF6_B). They receive the Supply voltages from the Rectification slotcards and convert them to a variable voltage output in a range from 1.26 V to 32 V in positive SFs (numbers 1 to 5), and -16 to -1.26 in negative SFs (number 6). The conversion is attained by the use of an adjustable voltage regulator (LT1083 from Linear Technology for positive voltages) and two resistances, one of the resistors being an RDAC (Rheostat Digital to Analog Converter). The RDACs (model AD5292 from Amidata [4]) can vary their resistor values from 20Ω to 20KΩ, causing the output voltage to scale following this formula for positive voltages:

$$PSU\ voltage = 1.25 * \left(1 + \frac{20K\Omega - RDAC_{Value}}{Rshunt}\right)$$

Negative voltages are compliant with the same formula, adding a negative sign and dividing by two.

Rshunt allows electric current to pass around him by creating a low resistance path (see schematic). Its value is around 820 Ω.

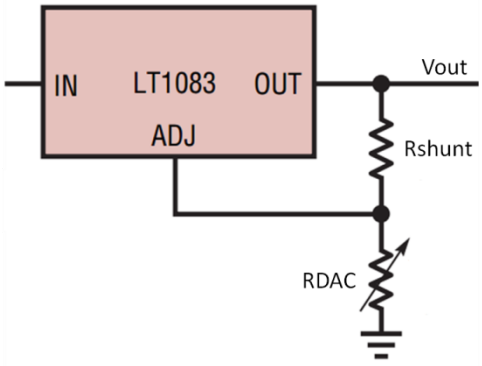


Figure 4 - PSUs' hardware voltage generation using a regulator

The RDAC is digitally adjusted by configuring its Value Register (10 bits long, allowing $2^{10} = 1024$ different values linearly distributed) through an SPI Bus. This type of bus presented two main issues: it allows few connected elements and it requires 4 connections in the backplane. An I2C Bus (only two connections and with room for several connected devices) solved the problems: by implementing an I2C to SPI converter (a.k.a. ***I2CtoSPIBridge***) in each Functional slotcard and a single I2C Bus to link up the MCF5213 with all the SFs, there is enough space in each SF for it to have its own private SPI bus communicating the two RDACs with the *I2CtoSPIBridge* (model SC18IS602B from Amidata [4]).

To connect/disconnect the voltage generated in the Regulators, relays have been installed between their outputs and the PSUs' outputs (the connectors for BNC cables where other Radar subsystems are plugged in). This relays are digitally managed by the MCF5213.

Both relays and Regulators have their manual counterpart set parallel to them, with a pair of jumpers to select either. RDACs can be substituted by the neighbour variable potentiometer (whose resistor value varies with a small screw on top of it), while Digital relays are accompanied by a manual switcher. These devices provide the system with an alternative in case the digitally controlled elements weren't available.

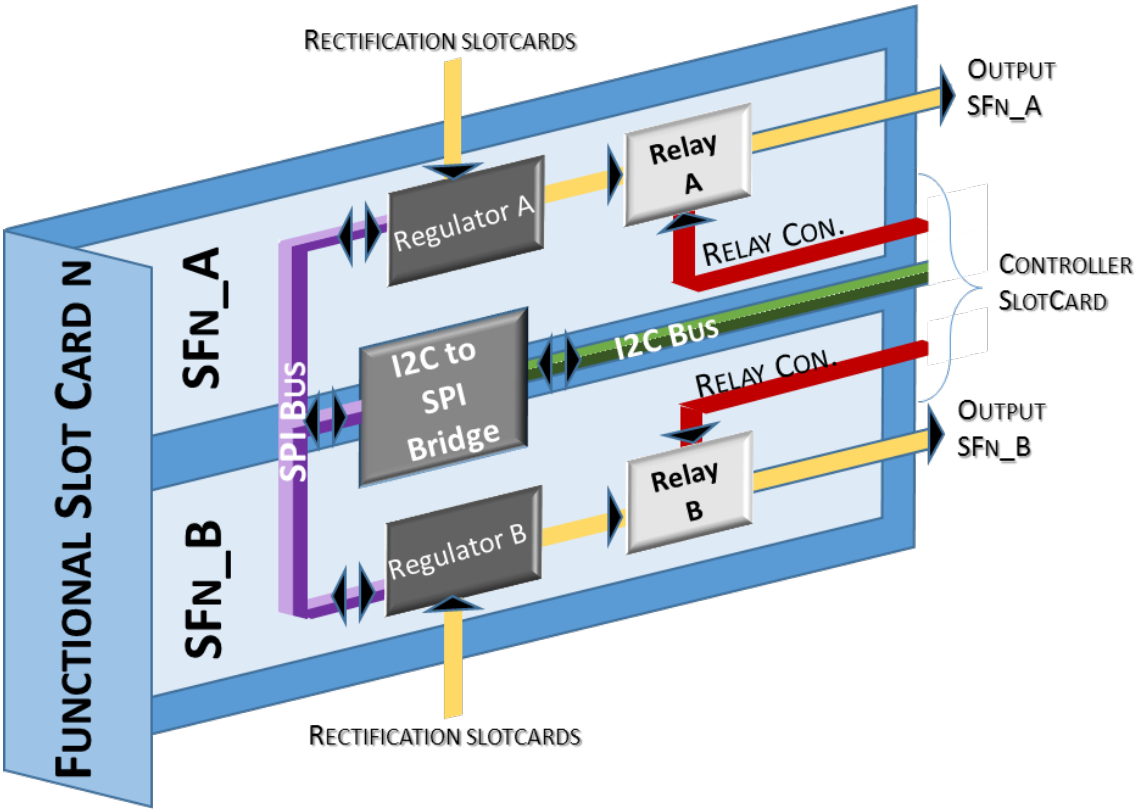


Figure 5 - Functional slotcard simplified hardware diagram

2.3 CONTROLLER SLOT CARD

The Controller slot card, located between SF3 and SF4, contains the MCF5213, where all the PSUControl software is executed. Aside from the microcontroller, there are several modules in the Controller card which serve various purposes:

- MUX Module: selects the magnitude (voltage or current channel) to be introduced in the MCF5213 analogical to digital converter (A2D).
- AGC Module: Scales the magnitude selected by the MUX Module so that the final value is high enough for the A2D from the microcontroller.
- Relay connections: connection/disconnection of each PSU's digital relay.
- UART to USB: Connection with Windows PC. Since no software is involved in this conversion, it will be treated as straight USB connection, avoiding further analysis.

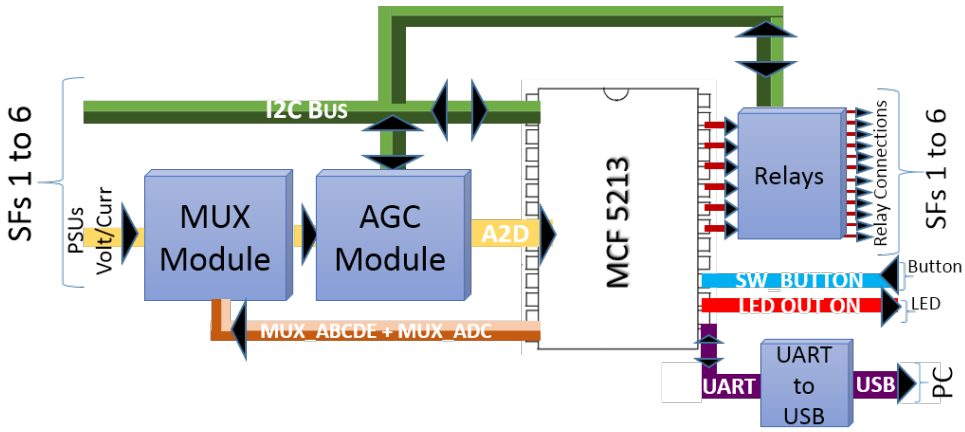


Figure 6 - Controller Slotcard hardware diagram

2.3.1 MUX HW MODULE

There are two sets of multiplexers (ADG508 from Amidata [4]), used to select the magnitude channel to be measured in two steps:

1. The first set, MUX_A to MUX_E, receive all the output signals from both the functional slotcards (SF voltages and currents) and the rectification slotcards (Supply voltages). Also, three internal voltages required by digital components in the controller slotcard have been included (±12V and 3,3V). *Supply* voltages extracted from the rectification slotcards and *Internal* voltages in the Controller slotcard will be shortened as **SnI** (**S**upply a**n**d **I**nternal). A total of 38 channels are being introduced to the MUX module (12 PSU voltages, 12 PSU currents and 14 SnI voltages).
2. The second set, MUX_ADC, has his inputs connected to all the firt set muxes' outputs, obtaining a single output to be sent to AGC input to be measured.

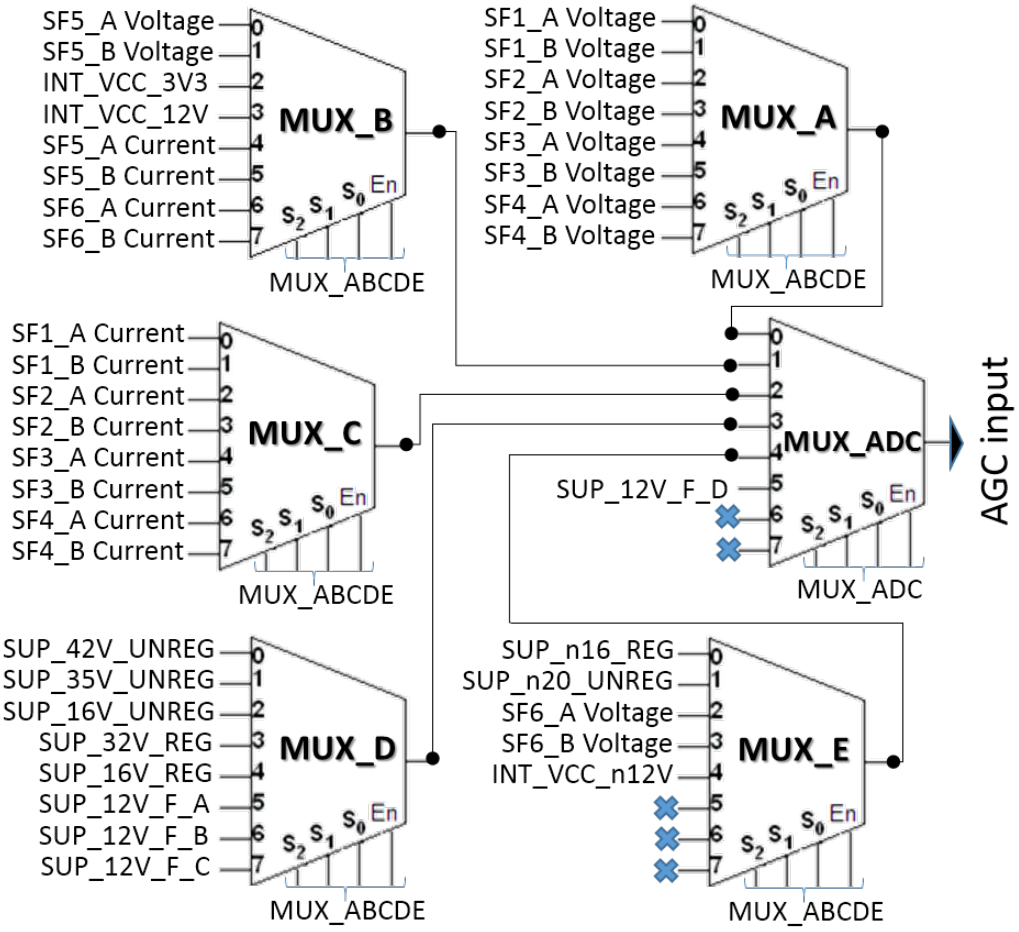


Figure 7 - MUX hardware diagram

Voltages from PSUs and SnIs are scaled down using resistor dividers to obtain a maximum value of 12V at the entrance of the AGC input, with the negative magnitudes being inversed. The scale factor is different for each magnitude, depending on its maximum value. Currents from PSUs are converted to voltages using a small circuit that obtains a voltage value of 1.25 times the current in amperes.

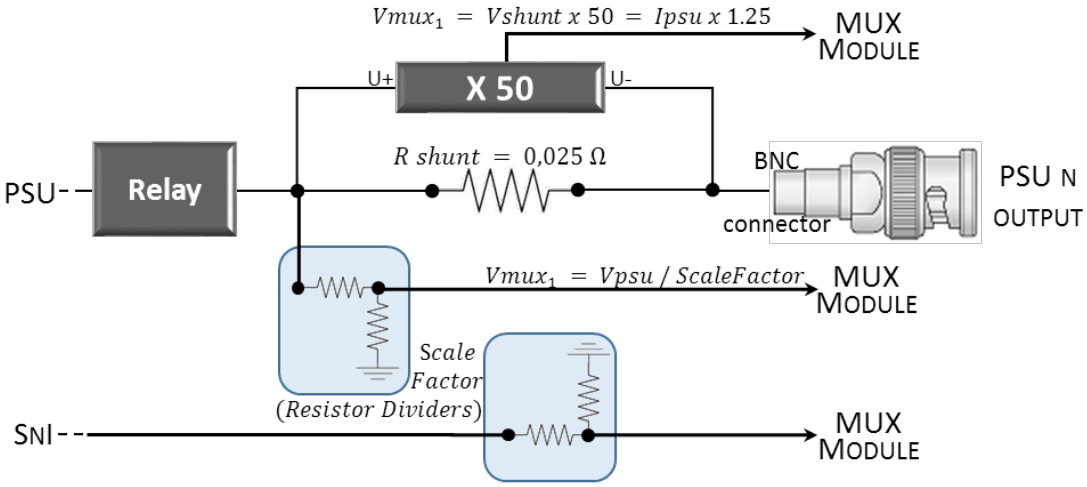


Figure 8 - Magnitude channels introduced to MUX Module for monitoring

Table 1 - Magnitudes being measured using MUX Module

	MAGNITUDE CHANNEL	MUX_ABCDE	MUX_ADC	SCALE FACTOR
PSU Voltages	SF1_A Voltage	0	0	3
	SF1_B Voltage	1	0	3
	SF2_A Voltage	2	0	3
	SF2_B Voltage	3	0	3
	SF3_A Voltage	4	0	3
	SF3_B Voltage	5	0	3
	SF4_A Voltage	6	0	3
	SF4_B Voltage	7	0	3
	SF5_A Voltage	0	1	3
	SF5_B Voltage	1	1	3
	SF6_A Voltage	2	4	-1,68
	SF6_B Voltage	3	4	-1,68
PSU Currents	SF1_A Current	0	2	1/1,25
	SF1_B Current	1	2	1/1,25
	SF2_A Current	2	2	1/1,25
	SF2_B Current	3	2	1/1,25
	SF3_A Current	4	2	1/1,25
	SF3_B Current	5	2	1/1,25
	SF4_A Current	6	2	1/1,25
	SF4_B Current	7	2	1/1,25
	SF5_A Current	4	1	1/1,25
	SF5_B Current	5	1	1/1,25
	SF6_A Current	6	1	-1/1,25
	SF6_B Current	7	1	-1/1,25
SnI Voltages	SUP_42V_UNREG	0	3	5
	SUP_35V_UNREG	1	3	3
	SUP_16V_UNREG	2	3	1,68
	SUP_32V_REG	3	3	3
	SUP_16V_REG	4	3	3
	SUP_12V_F_A	5	3	3,68/3
	SUP_12V_F_B	6	3	3,68/3
	SUP_12V_F_C	7	3	3,68/3
	SUP_12V_F_D	-	5	3,68/3
	SUP_n16_REG	0	4	-1,68
	SUP_n20_UNREG	1	4	-2
	INT_VCC_3V3	2	1	1
	INT_VCC_12V	3	1	1
	INT_VCC_n12V	4	4	-1

2.3.2 AGC HW MODULE

The AGC Module scales the magnitude selected by the MUX Module so that its maximum value is lower than 3,3V for the MCF5213 analogic input. It is formed by three different steps:

1. A first resistor divider not to overload the O.A. setting a factor of 4.

2. The amplification step, based on an O.A. and controlling its gain through a RDAC similar to those in the functional slotcards. Thanks to it, each magnitude can be scaled differently, reaching a high value in the A2D scale (0-3,3V) regardless of their instantaneous values, thus augmenting the system's sensitivity.
3. A second resistor divider to set the maximum value for the A2D input below 3,3V (max allowed value for MCF5213 A2D module).

The AGC module does not modify the gain by itself, relying on PSUControl software to control the gain by communicating with RDAC_AGC following the same procedure as for any other RDAC.

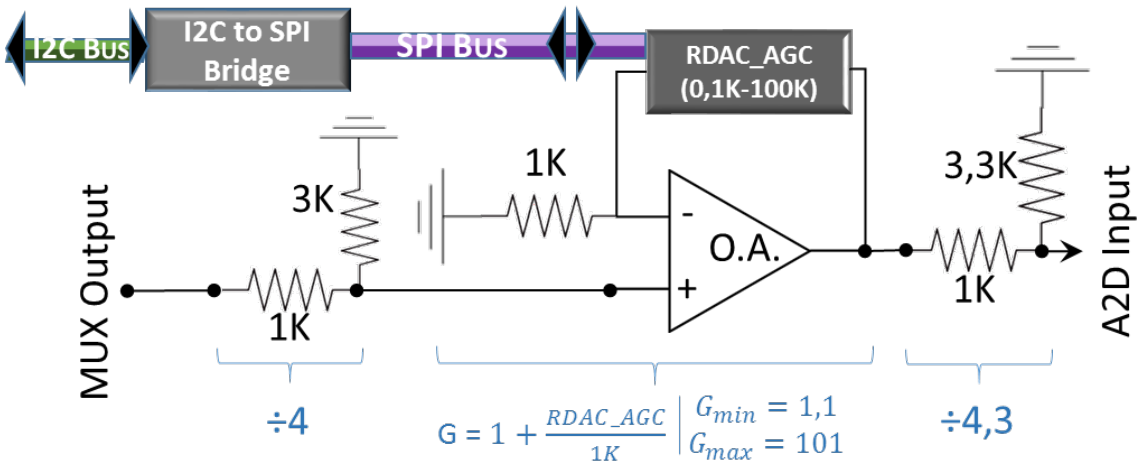


Figure 9 - Automatic Gain Control hardware diagram

2.3.3 RELAYS

The digital relays hosted inside every SF are connected/ disconnected by two different methods depending on the number of SF they are installed into:

- Numbers one to three: Their SFs are directly connected to 6 pins (3 slotcards x 2 power supply units in every SF) in the MCF5213.

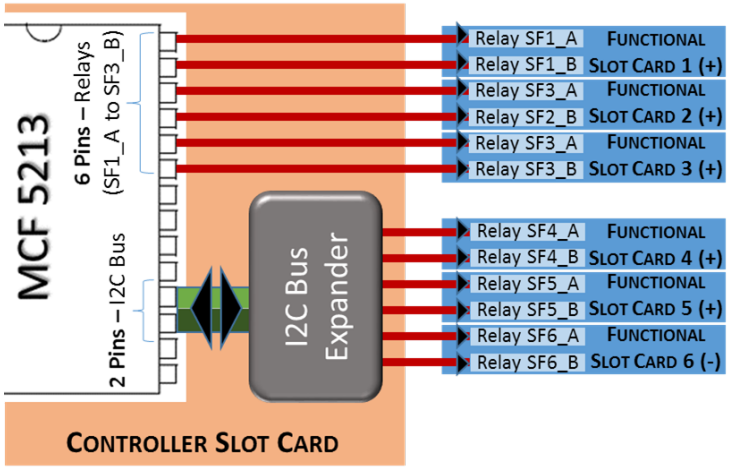


Figure 10 - Relays hardware diagram

- Numbers four to six: their connection is achieved via the I2C bus and an I2C expander (PCA8574/74A from Amidata [4]), obtaining the required 6 wires. The I2C Bus Expander takes the 8 bits that form each I2C package and, after a serial to parallel conversion, they are send through its 8 exit pins. Only six of the eight pins are used (for an example of configuration go to Logic View, subsection of Communications).

2.3.4 MCF5213 MICROCONTROLLER

The MCF5213 hosts all the PSUControl software. The microcontroller has most of his pins used for communications and controlling:

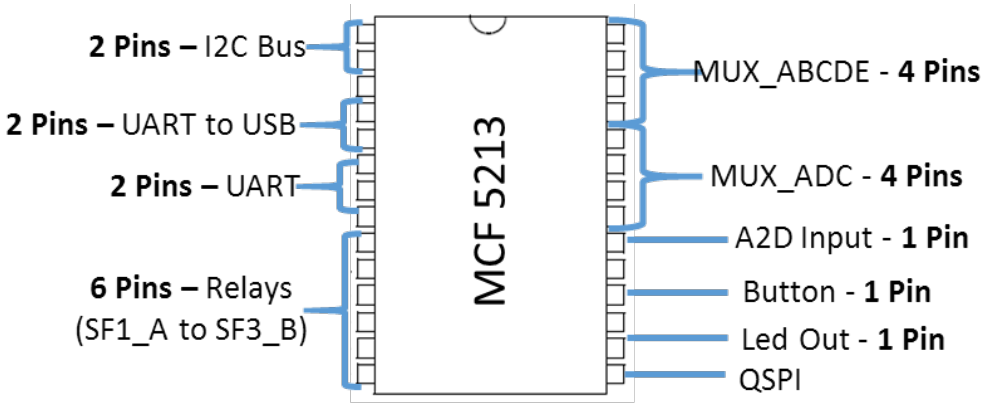


Figure 11 - MCF5213 connections diagram

- I2C Bus: Communication with I2CtoSPIBridge in every SF to adjust RDACs, Relays for SFs 4 to 6 and AGC adjustment.
- UART to USB: UART port converted to USB serial port to communicate with a Windows PC.
- UART: Used for Debug.
- Relays: communication with digital relays for SFs 1 to 3.
- MUX_ABCDE: Control of selected input for multiplexers A to E.
- MUX_ADC: Control of selected input for MUX_ADC.
- A2D Input: Analogic input that will be measured.
- Button: Analogic button that will switch on/off the whole system.
- Led Out: Output signal that will serve as indicator of the system status (on/off)
- QSPI: Communication with the other microcontroller (MCF5270).

2.3.5 MCF5270 MICROCONTROLLER

A second microcontroller is included in the hardware design of the supply system: a MicroColdFire 5270 (or MCF5270). It's connected to the main microcontroller (MCF5213) via the QSPI bus, and it allows external connectivity via Ethernet (and IP), implementing a RJ45 connector for this purpose. This second controller would host and manage an alternate User Interface based on HTML, with the MCF5270 acting as a web server. This second UI would communicate with the MCF5213 across a QSPI (Queued Serial Peripheral Interface Bus) to execute the orders given by the user.

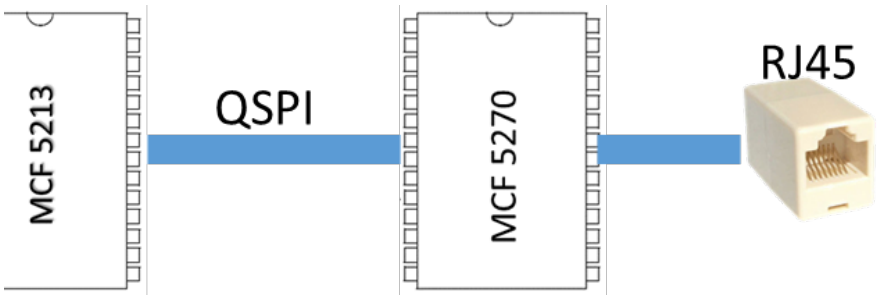


Figure 12 - MCF5270 with Ethernet connector

3 USER REQUIREMENTS

User Requirements (or URs) specify what the *client* expects the software to be able to do [5], understanding the *client* as the SSR department that defined the previous Radar .Their importance is rated employing MoSCoW [6] prioritisation methodology:

1. Mandatory: A must be, obligatory.
2. Desirable: Should be included, although it is not essential.
3. Optional: Could be included, less priority than any other URs, it may not end up being implemented.
4. Future: Won't be included in the system. Left for future implementation as a possible upgrade.

Priority Levels
MANDATORY - MUST
DESIRABLE - SHOULD
OPTIONAL - COULD
FUTURE - WON'T

Table 2 - User Requirements

CATEGORY	UR ID	NAME	Priority
GENERAL	UR_100	UR_GENERAL_HARDWARE	MANDATORY
	UR_101	UR_GENERAL_SOFTWARE	MANDATORY
	UR_102	UR_GENERAL_COMMUNICATIONS	MANDATORY
MONITORING	UR_200	UR_MONITORING_SCAN_VOLTAGE	MANDATORY
	UR_201	UR_MONITORING_SCAN_CURRENT	MANDATORY
	UR_202	UR_MONITORING_PSUs&SNIs	MANDATORY
	UR_203	UR_MONITORING_AVERAGE_PEAK_DATA	OPTIONAL
CONFIGURATION	UR_300	UR_CONFIGURATION_VOLTAGE	MANDATORY
	UR_301	UR_CONFIGURATION_ALLOWED_SCALE	MANDATORY
	UR_302	UR_CONFIGURATION_PROGRAMMING	DESIRABLE
SWITCH-ON	UR_400	UR_SWITCH-ON_RELAYS	MANDATORY
	UR_401	UR_SWITCH-ON_TIMING	DESIRABLE
	UR_402	UR_SWITCH-ON_BUTTON	DESIRABLE
ALARM	UR_500	UR_ALARM_GENERATE	MANDATORY
	UR_501	UR_ALARM_PROCESS	MANDATORY
	UR_502	UR_ALARM_DISCONNECT_PSUS	MANDATORY
WINDOWS	UR_600	UR_WINDOWS_APLICATION	MANDATORY
	UR_601	UR_WINDOWS_COMMUNICATION	MANDATORY
IP	UR_700	UR_IP_MOD5270	FUTURE
	UR_701	UR_IP_UC_COMMUNICATION	FUTURE

3.1 GENERAL

General URs define both software and hardware platforms to be used, as well as their connections. They don't require to be checked for completion, because they merely describe the system foundations. Nevertheless, they are needed for an all-around perspective of the radar supply system.

3.1.1.1 UR_100 – UR_GENERAL_HARDWARE

Use of MCF 5213 (Netburner MOD5213) as the main microcontroller, where all the main processes will be hosted. Use of a Windows PC for the external application, the programming & debug. Use of

specific hardware (AD5292, I2CtoSPIBridge, Digital Multiplexers, Relays, I2C bus expander) for particular functions.

3.1.1.2 UR_101 – UR_GENERAL_SOFTWARE

Use of Netburner Libraries as bedrock for PSUControl. Use of Netburner Software (MTTTY & Eclipse IDE) in a Windows PC for PSUControl development and debug. Use of specific components' libraries extracted from their datasheets. All the project will be programmed in 'C' Language [7].

3.1.1.3 UR_102 – UR_GENERAL_COMMUNICATIONS

Use of I2C bus for communications between MCF5213 and I2CtoSPIBridges. Use of SPI bus for internal communications with RDACs in each functional slotcard. Use of USB for serial communications with a PC (OS: Windows 7 or higher), and a Serial bus for debug purposes. Use of a twelve wires for communications with all relays.

3.2 MONITORING

URs Related to the continuous sampling of voltages & currents.

3.2.1.1 UR_200 - UR_MONITORING_SCAN_VOLTAGE

Sampling the output voltage of both PSUs and Rectification slotcards, and Controller slotcard internal voltages (SnIs): configure the digital multiplexers and an Automatic Gain Control System and reverse the scaling in both modules.

3.2.1.2 UR_201 - UR_MONITORING_SCAN_CURRENT

Sample the output current of all PSUs by configuring the digital multiplexers and an Automatic Gain Control System: configure the digital multiplexers and an Automatic Gain Control System and reverse the scaling in both modules.

3.2.1.3 UR_202 - UR_MONITORING_PSUs&SNIs

Monitors the PSUs' (current & voltage) and SnIs' (voltage) functioning, recording their status throughout time. Create a non-stop circular thread that continuously goes over all the magnitudes and measures them.

3.2.1.4 UR_203 - UR_MONITORING_AVERAGE_PEAK_DATA

Store mean and peak values for all monitored magnitudes.

3.3 CONFIGURATION

This category involves all the symbolic data management about PSUs and SnIs, as well as voltage adjustment for PSUs.

3.3.1.1 UR_300 - UR_CONFIGURATION_VOLTAGE

Modify resistor values of digital potentiometers (RDACs) to adjust each PSU's output voltage.

3.3.1.2 UR_301 - UR_CONFIGURATION_ALLOWED_SCALE

Determine the permitted interval of values for digital potentiometers.

3.3.1.3 UR_302 - UR_CONFIGURATION_PROGRAMMING

Configure the PSUs' operation (on/off, output voltage value, alarm configuration) digitally.

3.4 SWITCH-ON

3.4.1.1 UR_400 - UR_SWITCH-ON_RELAYS

Control each PSU's output relay to connect/disconnect each of the twelve generated tensions.

3.4.1.2 UR_401 - UR_SWITCH-ON_TIMING

Activate relays after each PSU's programmed timing (delayed switch-on).

3.4.1.3 UR_402 - UR_SWITCH-ON_BUTTON

Global activation/deactivation of all PSUs when the exterior button is pressed.

3.5 ALARM

Alarm generation and triggering responses.

3.5.1.1 UR_500 - UR_ALARM_GENERATE

Compare the values of the magnitudes being monitored with inferior and superior thresholds (which can be configured) and count the number of times they trespass each limit. If that counter reaches a certain value (configurable), trigger an alarm for the concrete magnitude and threshold /Voltage/Current, Inferior/Superior).

3.5.1.2 UR_501 - UR_ALARM_PROCESS

Execute several processes (notifying the system, disconnecting PSU, adjusting local PSU's output voltage) when a certain alarm is triggered.

3.5.1.3 UR_502 - UR_ALARM_DISCONNECT_PSUS

Choose which PSUs will be disconnected when an alarm is triggered.

3.6 WINDOWS

Interface for system control & monitoring via a Windows program.

3.6.1.1 UR_600 - UR_WINDOWS_APLICATION

Interface for Windows Platform which will monitor & configure the whole PSUControl system.

3.6.1.2 UR_601 - UR_WINDOWS_COMMUNICATION

Communication between the PC and the MOD5213 system for PSU monitoring and configuration. Response to all PC inquiries.

3.7 IP

Both have been discarded for the present project, leaving them as future optional upgrades.

3.7.1.1 UR_700 - UR_IP_MOD5270

Inclusion of an extra microcontroller MOD5270 to allow IP connectivity through a RJ45 connector.

3.7.1.2 UR_701 - UR_IP_UC_COMMUNICATION

Use of a QSPI bus for inter-microcontroller communication (MOD5213 - MOD5270).

4 SYSTEM REQUIREMENTS

System requirements specify what the software WILL be able to do. Again, a MoSCoW scale is used for rating their importance (so that most important req. are developed first). They are divided into four different types: Interfaces, Functional, Complementary and Upgrades [8].

Along with the SR definitions, the software modules that fulfil each SR are included in the side. There are four packages: Controller, Libraries, Interface and Tests. Each package holds in several modules. There are also some isolated modules, left unpackaged. "Tests" package isn't included, as it affects every part of the system (further details in Test Cases section). The PSUControl detailment is carried out in Development View section.

PACKAGE(PCK): ...
MODULES(MOD): ...

4.1 TYPE INTERFACES

Interfaces are shared boundaries across which two separate components of the PSUControl system exchange information. The exchange is between software, computer hardware, peripheral devices, the user and combinations of these. They include the General URs (definition of all HW/SW components and their connections). An interface diagram known as Component Diagram (see development view) details all the HW/SW interfaces with their interconnections.

Table 3 – SR Type Interfaces/I

CATEGORY	SR ID	NAME	PRIORITY
HARDWARE	SR_I_100	SR_I_HARDWARE_GENERAL	MANDATORY
	SR_I_101	SR_I_HARDWARE_COMMUNICATIONS	MANDATORY
SOFTWARE	SR_I_200	SR_I_SOFTWARE_GENERAL	MANDATORY
	SR_I_201	SR_I_SOFTWARE_COMMUNICATIONS	MANDATORY
	SR_I_202	SR_I_SOFTWARE_USER_INTERFACE	MANDATORY
NIELSEN HEURISTICS FOR USER INTERFACE	SR_I_300	SR_I_HEUR_STATUS_VISIBILITY	DESIRABLE
	SR_I_301	SR_I_HEUR_UNDO_REDO	OPTIONAL
	SR_I_302	SR_I_HEUR_STANDARD	MANDATORY
	SR_I_303	SR_I_HEUR_ERROR_PREVENTION	OPTIONAL
	SR_I_304	SR_I_HEUR_INSTRUCTIONS	DESIRABLE
	SR_I_305	SR_I_HEUR_ACCELERATORS	OPTIONAL
	SR_I_306	SR_I_HEUR_AESTHETIC	MANDATORY
	SR_I_307	SR_I_HEUR_ERROR_MESSAGES	MANDATORY
	SR_I_308	SR_I_HEUR_HELP	DESIRABLE

4.1.1 HARDWARE

4.1.1.1 SR_I_100 – SR_I_HARDWARE_GENERAL

The hardware system serving as foundation of PSUControl will be as described in Hardware System Overview with no changes. The conditions established in UR_100 are accepted as they are.

4.1.1.2 SR_I_101 – SR_I_HARDWARE_CONNECTIONS

The connections between hardware modules and components will be as described in Hardware System Overview with no changes. The conditions established in UR_102 are accepted as they are.

4.1.2 SOFTWARE

4.1.2.1 SR_I_200 – SR_I_SOFTWARE_GENERAL

The software general characteristics depicted in UR_101 are left untouched. The conditions established in UR_101 are accepted as they are.

4.1.2.2 SR_I_201 – SR_I_SOFTWARE_COMMUNICATIONS

The MCF5213 will communicate with the PC using the MTTTY utility console. The MCF5213 will communicate with all the external modules (MUX, AGC, Relay, RDAC, I2C&SPI) using the module libraries that will be designed for them.

4.1.2.3 SR_I_202 – SR_I_SOFTWARE_USER_INTERFACE

Create a User Interface based on MTTTY Console to configure and monitor all the system, as well as execute tests. It may also include support for all the functions in the module libraries.

PCK: Interfaces
MOD: All

4.1.3 NIELSEN HEURISTICS FOR USER INTERFACE

The main goal of heuristic evaluations is to identify any potential problems associated with the design of a user interface (a.k.a. UI) and prevent them, judging their compliance with recognized usability principles (the "heuristics"). The most widely used set of heuristics are Nielsen's heuristics [9], which are being applied to the console-based UI in PSUControl.

4.1.3.1 SR_I_300 – SR_I_HEUR_STATUS_VISIBILITY

Receive status updates within a 1 second period, and display them in a box with a size similar to or smaller than the total window (no scroll required to see all info at the same time).

4.1.3.2 SR_I_301 – SR_I_HEUR_UNDO_REDO

Create "Undo" and "Redo" options for alarm configuration, as well as an "Exit" option for every panel to leave the unwanted state without having to go through an extended dialogue.

4.1.3.3 SR_I_302 – SR_I_HEUR_STANDARD

Create a "Term List" for every term use in the interface, avoiding the use of different words for a same meaning

4.1.3.4 SR_I_303 – SR_I_HEUR_ERROR_PREVENTION

Present the user a confirmation option before using any function which has a high error probability

4.1.3.5 SR_I_304 – SR_I_HEUR_INSTRUCTIONS

The use of every panel in the interface must be clear, or otherwise accompanied by a command instructions list.

4.1.3.6 SR_I_305 – SR_I_HEUR_ACCELERATORS

Shortcut methods which will carry out several processes at once to speed up configuration

4.1.3.7 SR_I_306 – SR_I_HEUR_AESTHETIC

Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.

4.1.3.8 SR_I_307 – SR_I_HEUR_ERROR_MESSAGES

Error messages include, in addition to its code, a short explanation of the error, and if possible, suggest a solution

4.1.3.9 SR_I_308 – SR_I_HEUR_HELP

Libraries & Help information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.

4.2 TYPE FUNCTIONAL

Functional system requirements comprehend everything PSUControl can do. They are directly related to all the UR, covering all the software modules.

Table 4 – SR Type Functional/F

CATEGORY	SR ID	NAME	PRIORITY
MONITORING	SR_F_100	SR_F_MONITORING_SCAN_VOLTAGE	MANDATORY
	SR_F_101	SR_F_MONITORING_SCAN_CURRENT	MANDATORY
	SR_F_102	SR_F_MONITORING_TASK	MANDATORY
	SR_F_103	SR_F_MONITORING_STORE_AVERAGE	DESIRABLE
	SR_F_104	SR_F_MONITORING_STORE_PEAKS	OPTIONAL
	SR_F_105	SR_F_MONITORING_MUX_LIBRARY	MANDATORY
	SR_F_106	SR_F_MONITORING_AGC_LIBRARY	MANDATORY
CONFIGURATION	SR_F_200	SR_F_CONFIGURATION_RDAC_LIBRARY	MANDATORY
	SR_F_201	SR_F_CONFIGURATION_ADJUST_VOLTAGE	MANDATORY
	SR_F_203	SR_F_CONFIGURATION_DATA_TYPES	MANDATORY
	SR_F_204	SR_F_CONFIGURATION_DATA_LISTS	MANDATORY
	SR_F_205	SR_F_CONFIGURATION_FLASH_MEM	DESIRABLE
	SR_F_206	SR_F_CONFIGURATION_SYSTEM	DESIRABLE
COMMUNICATION	SR_F_300	SR_F_COMMUNICATION_I2C&SPI_LIBRARY	MANDATORY
SWITCH-ON	SR_F_400	SR_F_SWITCH-ON_RELAY_LIBRARY	MANDATORY
	SR_F_401	SR_F_SWITCH-ON_TASK	MANDATORY
	SR_F_402	SR_F_SWITCH-ON_BUTTON	MANDATORY
ALARM	SR_F_500	SR_F_ALARM_TASK	MANDATORY
	SR_F_501	SR_F_ALARM_PROTOCOLS	MANDATORY
WINDOWS	SR_F_600	SR_F_WINDOWS_INTERFACE	MANDATORY
	SR_F_601	SR_F_WINDOWS_COMMUNICATION	MANDATORY

4.2.1 MONITORING

4.2.1.1 SR_F_100 - SR_F_MONITORING_SCAN_VOLTAGE

Set Muxes and AGC to read an analog 0-3.3V tension value using the MOD5213 ADC and scale it to voltage for PSUs and SnIs. The voltage will be selected by the muxes and augmented by the AGC so as to obtain a scaled value higher than 40% of the fullscale in the A2D section of the MCF5213.

PCK: Controller
MOD: VoltCurrCTRL

4.2.1.2 SR_F_101 - SR_F_MONITORING_SCAN_CURRENT

Set Muxes and AGC to read an analog 0-3.3V tension value using the MOD5213 ADC and scales it to current for PSUs. The current will be

selected by the muxes and augmented by the AGC so as to obtain a scaled value higher than 40% of the fullscale in the A2D section of the MCF5213.

4.2.1.3 SR_F_102 - SR_F_MONITORING_TASK

RTOS task who will periodically sample all voltages & currents and update their values in RAM.

PCK: Controller
MOD: MonitorCTRL

4.2.1.4 SR_F_103 - SR_F_MONITORING_STORE_AVERAGE

Store voltage & current mean values using an arithmetic mean (or a geometric decay sequence)

4.2.1.5 SR_F_104 - SR_F_MONITORING_STORE_PEAKS

Store Max/min peaks for voltages and currents in a matrix

4.2.1.6 SR_F_105 - SR_F_MONITORING_MUX_LIBRARY

Setting the 8 Mux configuration pins (4xMUX_ABCDE & 4xMUX_ADC) to set the Mux Module output to a desired voltage/current magnitude.

PCK: Libraries
MOD: MUXLibrary

4.2.1.7 SR_F_106 - SR_F_MONITORING_AGC_LIBRARY

Use the I2CtoSPIBridge (with I2C&SPILibrary) located in the Controller slotcard to communicate with RDAC_AGC, and update the AGC gain to obtain a high sensitivity towards measurement of voltages/currents.

PCK: Libraries
MOD: AGCLibrary

4.2.2 CONFIGURATION

4.2.2.1 SR_F_200 - SR_F_CONFIGURATION_RDAC_LIBRARY

Configuration and Reading of Digital Rheostats AD5292 using I2C&SPILibrary.Configure RDAC Control Register, to allow or reject updates for the RDAC Value Register. Change Value Register in RDACs, thus updating the PSUs' output voltages. Those changes will take place for an allowed interval of values, set by the conversion method from RDAC Register values to PSU output voltages. Read RDAC CTRL and Value Registers. Program RDACs' 20TP Memory to define a new default value.

PCK: Libraries
MOD: RDACLibrary

4.2.2.2 SR_F_201 - SR_F_CONFIGURATION_ADJUST_VOLTAGE

Use of RDAC Library to adjust the output voltage value, checking for correct value updating. Store set values in RAM if right, report errors otherwise.

PCK: Controller
MOD: VoltCurrCTRL

4.2.2.3 SR_F_203 - SR_F_CONFIGURATION_DATA_TYPES

Create a symbolic structure to accommodate all of the required info for each PSU and SnI, using a "C" language structure (typedef PSU_TYPE and SnI_TYPE)

PCK: None
MOD: Typedefs

4.2.2.4 SR_F_204 - SR_F_CONFIGURATION_DATA_LISTS

Define a list of 12 PSU_TYPE objects that will contain all the PSUs' configuration parameters in RAM. Define a list of 14 SnI_TYPE objects that will contain all the SnIs' configuration parameters in RAM. Both will have methods to set default values and console printing methods.

PCK: Controller
MOD: DataListsCTRL

4.2.2.5 SR_F_205 - SR_F_CONFIGURATION_FLASH_MEM

Store & retrieve the current PSU & SnI lists in and from a flash memory section of the MCF5213 to keep data when PSUControl is shutdown. Includes a data verification process.

PCK: Controller
MOD: FlashMemCTRL

4.2.2.6 SR_F_206 - SR_F_CONFIGURATION_SYSTEM

Definition and modification of several configuration values (alarm update period, SnI included/excluded from alarmTask and monitorTask). Methods to pause alarmTask and monitorTask (setting them into idleMode and sequentialMode respectively). Methods to set testMode for monitorTask and SwitchONPSUsTask (recording their status throughout the process and printing extra information in console).

PCK: Controller
MOD: ConfigCTRL

4.2.3 COMMUNICATION

4.2.3.1 SR_F_300 - SR_F_COMMUNICATION_I2C&SPI_LIBRARY

General I2C communication module (could be used outside of PSUControl system). Allows communication with multiple devices (I2CtoSPIBridges, I2CBusExpander in Relays) by using I2C Serial bus. Detailed information in case of error. Communication across I2CtoSPIBridges, to configure SPI channel & send messages through SPI serial bus. Used to communicate with RDACs.

PCK: Libraries
MOD: I2C&SPILibrary

4.2.4 SWITCH-ON

4.2.4.1 SR_F_400 - SR_F_SWITCH-ON_RELAY_LIBRARY

Use of GPIO pins to connect/disconnect relays for SF1_A to SF3_B. Use of I2C bus to communicate with I2C Bus expander and connect/disconnect relays for SF4_A to SF6_B.

PCK: Libraries
MOD: RelayLibrary

4.2.4.2 SR_F_401 - SR_F_SWITCH-ON_TASK

Process in charge of controlling the asymmetrically-delayed PSU switching-on process (defined by value of “initializationTimer” from PSU_TYPE).

4.2.4.3 SR_F_402 - SR_F_SWITCH-ON_BUTTON

Setting Button as an external interruption source, switching off all the connected PSUs when first pressed, and switching them on when pressed again.

PCK: Controller
MOD: SwitchOnCTRL

4.2.5 ALARM

4.2.5.1 SR_F_500 - SR_F_ALARM_TASK

RTOS task in charge of monitoring and triggering the alarms. Checks, for each and every PSU, the limit exceeding (by using the last sample) and the time it's been exceeded.

PCK: Controller
MOD: AlarmCTRL

4.2.5.2 SR_F_501 - SR_F_ALARM_PROTOCOLS

Execute several protocols when an alarm is triggered. Protocols are processes such as notifying the system via console message, disconnecting PSUs and/or modifying local PSU's output voltage.

4.2.6 WINDOWS

4.2.6.1 SR_F_600 - SR_F_WINDOWS_INTERFACE

Fill up the main task of MCF5213 with the User Interface (UI) defined in SR Type "Interfaces", executing all the interface methods in the MCF5213 and receiving the input from the PC's keyboard.

PACKAGE: Interface
MOD: All

4.2.6.2 SR_F_601 - SR_F_WINDOWS_COMMUNICATION

Use the in-built communication between MCF5213 and MTTTY Console in the PC (through the UART to USB connection) as the software communication required for UI usage.

4.3 TYPE COMPLEMENTARY

Complementary System Requirements aren't explicitly included in the User Requirements definition. They serve as metric of PSUControl's functioning parameters (efficiency, reliability, unavailability and maintenance).

Table 5 – SR Type Complementary/C

CATEGORY	SR ID	NAME	PRIORITY
EFFICIENCY	SR_C_100	SR_C_EFFICIENCY_REFRESH	DESIRABLE
	SR_C_101	SR_C_EFFICIENCY_ALARM	DESIRABLE
	SR_C_102	SR_C_EFFICIENCY_VOLTAGE	DESIRABLE
RELIABILITY	SR_C_200	SR_C_RELIABILITY	DESIRABLE
UNAVAILABILITY	SR_C_300	SR_C_UNAVAILAVILITY	**FUTURE**
MAINTENANCE	SR_C_400	SR_C_MAINTENANCE	OPTIONAL

4.3.1 EFFICIENCY

4.3.1.1 SR_C_100 - SR_C_EFFICIENCY_REFRESH

PSUControl system monitors each PSU's status once every 100ms top

4.3.1.2 SR_C_101 - SR_C_EFFICIENCY_ALARM

PSUControl system can update each PSU's alarm once every 100ms

4.3.1.3 SR_C_102 - SR_C_EFFICIENCY_VOLTAGE

PSUControl system programmed voltage for PSUs will fall within a 5% margin of the real voltage 90% of the time.

4.3.2 RELIABILITY

4.3.2.1 SR_C_200 - SR_C_RELIABILITY

The system will be reliable on its console outputs a 100% of the time it's turned on and working (proper value updating)

4.3.3 UNAVAILABILITY

4.3.3.1 SR_C_300 - SR_C_UNAVAILAVILITY

Define an unavailability period for PSUControl System, using the mean time between failures and the mean time to repair. They will be calculated based on experience.

4.3.4 MAINTENANCE

4.3.4.1 SR_C_400 - SR_C_MAINTENANCE

Create & Update a "Test List" to check the correct system functioning.

4.4 TYPE UPGRADE

Upgrade requirements involve several ideas left for future implementation, as an upgrade for PSUControl System. They won't be detailed any further in this document.

Table 6 – SR Type Upgrade/U

CATEGORY	SR ID	NAME	PRIORITY
WINDOWS	SR_U_100	SR_U_WINDOWS_PROGRAM_UI	**FUTURE**
ALARM	SR_U_200	SR_U_ALARM_DISCONECTION_PROTOCOLS	**FUTURE**
MONITOR	SR_U_300	SR_U_MONITOR_AGC_MEMORY	**FUTURE**
ERROR	SR_U_400	SR_U_ERROR_MODULE	**FUTURE**

4.4.1 WINDOWS

4.4.1.1 SR_U_100 - SR_U_WINDOWS_PROGRAM_UI

Create a windows program and a communication protocol between MCF5213 and the PC, to re-place the console-based UI

4.4.2 ALARM

4.4.2.1 SR_U_200 - SR_U_ALARM_DISCONECTION_PROTOCOLS

Execute a new set of protocols when an alarm status goes from ON to OFF (connect PSUs, retrieve the voltage value for local PSU, send console messages)

4.4.3 MONITOR

4.4.3.1 SR_U_300 - SR_U_MONITOR_AGC_MEMORY

Set the value of the AGC to the last recorded value for each magnitude

4.4.4 ERROR

4.4.4.1 SR_U_400 - SR_U_ERROR_MODULE

Create an Error module to manage and record all the known errors (such as communication errors), accessible via the UI.

5 TRACEABILITY MATRIX UR – SR

This matrix defines the relations between User Requirements and System Requirements in order to assure that everything the *client* asked to do is incorporated as part of PSUControl system.

Table 7 - Traceability matrix UR/SR

UR	NAME	SR	NAME
UR_100	UR_GENERAL_HARDWARE	SR_I_100	SR_I_HARDWARE_GENERAL
UR_101	UR_GENERAL_SOFTWARE	SR_I_200	SR_I_SOFTWARE_GENERAL
UR_102	UR_GENERAL_COMMUNICATIONS	SR_I_101	SR_I_HARDWARE_COMMUNICATIONS
		SR_I_201	SR_I_SOFTWARE_COMMUNICATIONS
		SR_F_300	SR_F_COMMUNICATION_I2C&SPI_LIBRARY
UR_200	UR_MONITORING_SCAN_VOLTAGE	SR_F_100	SR_F_MONITORING_SCAN_VOLTAGE
		SR_F_105	SR_F_MONITORING_MUX_LIBRARY
		SR_F_106	SR_F_MONITORING_AGC_LIBRARY
		SR_F_300	SR_F_COMMUNICATION_I2C&SPI_LIBRARY
UR_201	UR_MONITORING_SCAN_CURRENT	SR_F_101	SR_F_MONITORING_SCAN_CURRENT
		SR_F_105	SR_F_MONITORING_MUX_LIBRARY
		SR_F_106	SR_F_MONITORING_AGC_LIBRARY
		SR_F_300	SR_F_COMMUNICATION_I2C&SPI_LIBRARY
UR_202	UR_MONITORING_PSUs&SNIs	SR_F_102	SR_F_MONITORING_TASK
UR_203	UR_MONITORING_AVERAGE_PEAK_DATA	SR_F_103	SR_F_MONITORING_STORE_AVERAGE
		SR_F_104	SR_F_MONITORING_STORE_PEAKS
UR_300	UR_CONFIGURATION_VOLTAGE	SR_F_200	SR_F_CONFIGURATION_RDAC_LIBRARY
		SR_F_201	SR_F_CONFIGURATION_ADJUST_VOLTAGE
		SR_F_300	SR_F_COMMUNICATION_I2C&SPI_LIBRARY
UR_301	UR_CONFIGURATION_ALLOWED_SCALE	SR_C_102	SR_C_EFFICIENCY_VOLTAGE
		SR_F_201	SR_F_CONFIGURATION_ADJUST_VOLTAGE
UR_302	UR_CONFIGURATION_PROGRAMMING	SR_F_203	SR_F_CONFIGURATION_DATA_TYPES
		SR_F_204	SR_F_CONFIGURATION_DATA_LISTS
		SR_F_205	SR_F_CONFIGURATION_FLASH_MEM
		SR_F_206	SR_F_CONFIGURATION_SYSTEM
UR_400	UR_SWITCH-ON_RELAYS	SR_F_400	SR_F_SWITCH-ON_RELAY_LIBRARY
		SR_F_300	SR_F_COMMUNICATION_I2C&SPI_LIBRARY
UR_401	UR_SWITCH-ON_TIMING	SR_F_401	SR_F_SWITCH-ON_TASK
UR_402	UR_SWITCH-ON_BUTTON	SR_F_402	SR_F_SWITCH-ON_BUTTON
UR_500	UR_ALARM_GENERATE	SR_F_500	SR_F_ALARM_TASK
		SR_C_101	SR_C_EFFICIENCY_ALARM
UR_501	UR_ALARM_PROCESS	SR_F_501	SR_F_ALARM_PROTOCOLS
UR_502	UR_ALARM_DISCONNECT_PSUS	SR_F_501	SR_F_ALARM_PROTOCOLS
UR_600	UR_WINDOWS_APLICATION	SR_I_202	SR_I_SOFTWARE_USER_INTERFACE
		SR_C_200	SR_C_RELIABILITY
		SR_C_100	SR_C_EFFICIENCY_REFRESH
		SR_I_300	SR_I_HEUR_STATUS_VISIBILITY
		SR_I_301	SR_I_HEUR_UNDO_REDO
		SR_I_302	SR_I_HEUR_STANDARD
		SR_I_303	SR_I_HEUR_ERROR_PREVENTION
		SR_I_304	SR_I_HEUR_INSTRUCTIONS
		SR_I_305	SR_I_HEUR_ACCELERATORS
		SR_I_306	SR_I_HEUR_AESTHETIC
		SR_I_307	SR_I_HEUR_ERROR_MESSAGES
		SR_I_308	SR_I_HEUR_HELP
UR_601	UR_WINDOWS_COMMUNICATION	SR_I_101	SR_I_HARDWARE_COMMUNICATIONS
		SR_I_201	SR_I_SOFTWARE_COMMUNICATIONS
		SR_F_601	SR_F_WINDOWS_COMMUNICATION

6 DEPLOYMENT VIEW

The deployment view [10] illustrates PSUControl system HW and SW, portraying the relations between the already defined hardware and the software modules that interact with each hardware element.

It must be pointed out that all the PSUControl system is loaded as a whole into MCF5213 and executed there. Exterior software would only include the in-built logic that is explained in each component's datasheet. Nonetheless, the Deployment View depicts Libraries as being executed in different HW components, which should be understood as **Libraries** being communication protocols between each hardware element and the MCF5213.

All **Controller** modules are logic-based, connecting with hardware via the Libraries and storing their configuration and values ready to be asked by the interface.

Both the **Interface** and the **Tests** (which belong to the higher layers in the PSUControl architecture as shown in the Development View) are pictured as being part of the PC, where the UI interacts with humans.

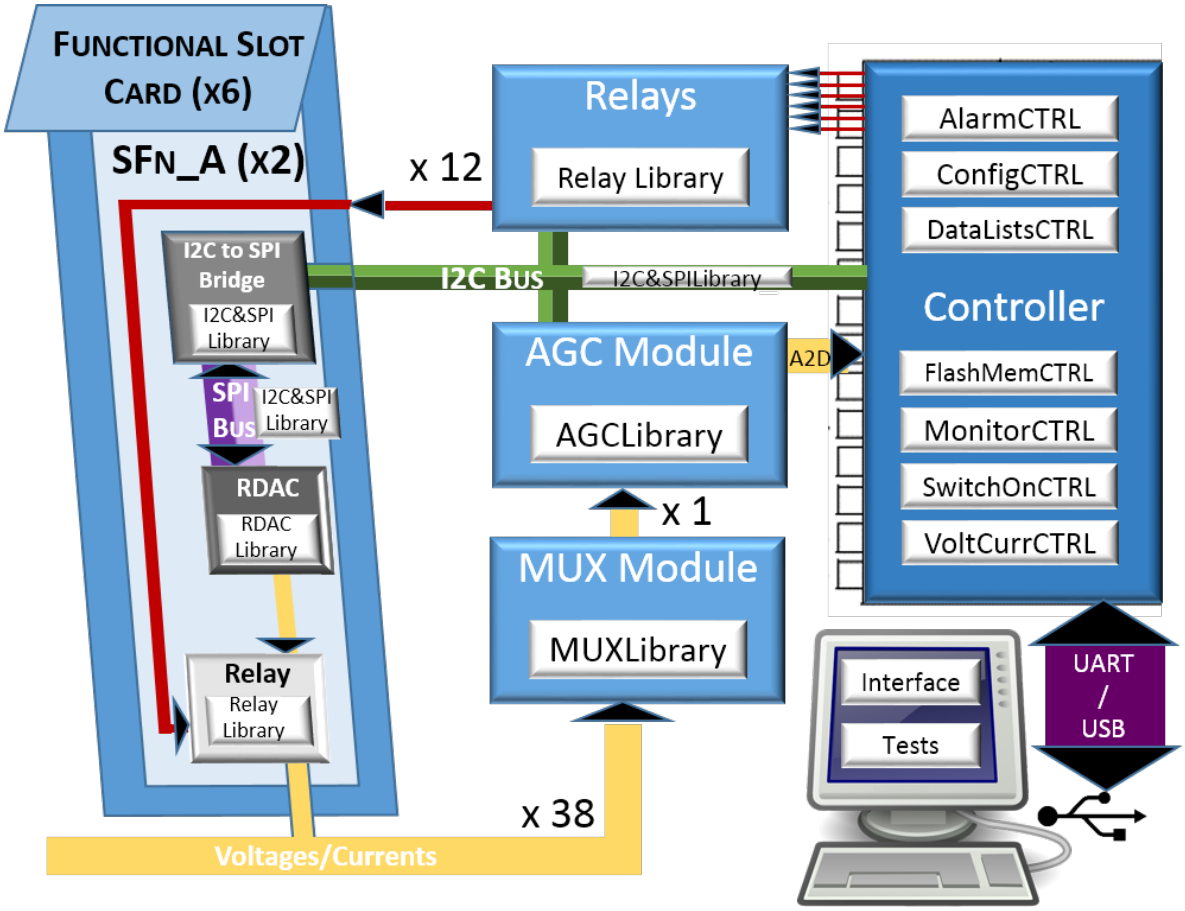


Figure 13 - Deployment diagram

7 DEVELOPMENT VIEW

PSUControl is separated into four main packages, as well as several unpackaged modules. Each package can be interpreted as a different layer in the system, where the bottom layers define interfaces (or methods) used and called by the ones above them.

Before expanding each package, a quick tip is in order. When a C module includes a method that is used by other modules, it's expressed as to "***Define an Interface***". When the other modules use the aforementioned method, it's expressed as to "***Implement an Interface***". There is yet another relation between modules: when module A can alter module B's functioning (e.g. stop a process created by another module), it's worded as "***B is dependant of A***", and "**A has control over B**" [11].

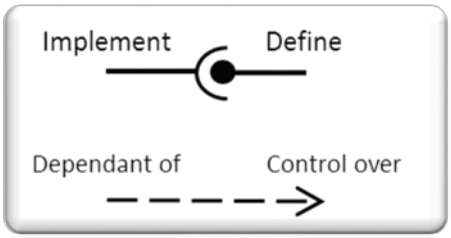


Figure 14 - Interrelations in Development diagram

7.1 TOP LAYER – MAIN PROCESS

The system is based on the Operative System embedded in the microcontroller (uCOS) from Netburner [3]. Its main process, or top layer, lies in main.cpp and includes the initialization of all the system as well as smaller processes (a.k.a. Tasks). There are three tasks in PSUControl. In this view they are only being located in the system, leaving the exhaustive description of purpose and functioning for the Process View:

- **mainTask(0):** created by default, starts executing the contents of main.cpp. It starts at the Interface menu from Interface package, granting access to all the UI.
- **alarmTask(1):** OS task that pops once every 100 *ms*, with its definition in AlarmCTRL and its initialization in main.cpp. It is in charge of alarm update and generation, as well as alarm protocols execution.
- **monitorTask(2):** RTOS task that executes once every *ms*, with its definition in MonitorCTRL, its initialization in main.cpp and the definition of its periodical interruption (PIT1 timer with a period of 1ms) in Interruption.cpp. It samples all the voltage and current magnitudes (form PSUs and SnIs) in a continuous cycle.

Main.cpp, after the initialization, calls the first level of the Interface Package, granting access to all the interface menus defined further on.

Interruption.cpp populates the interruption vectors in MCF5213 with the right values for the timed interruption (monitorTask) and external interruption (triggered when the external button is pushed). It is also in charge of their Interruption Service Routines, that handle each interruption parameters and redirects them to the methods to be executed.

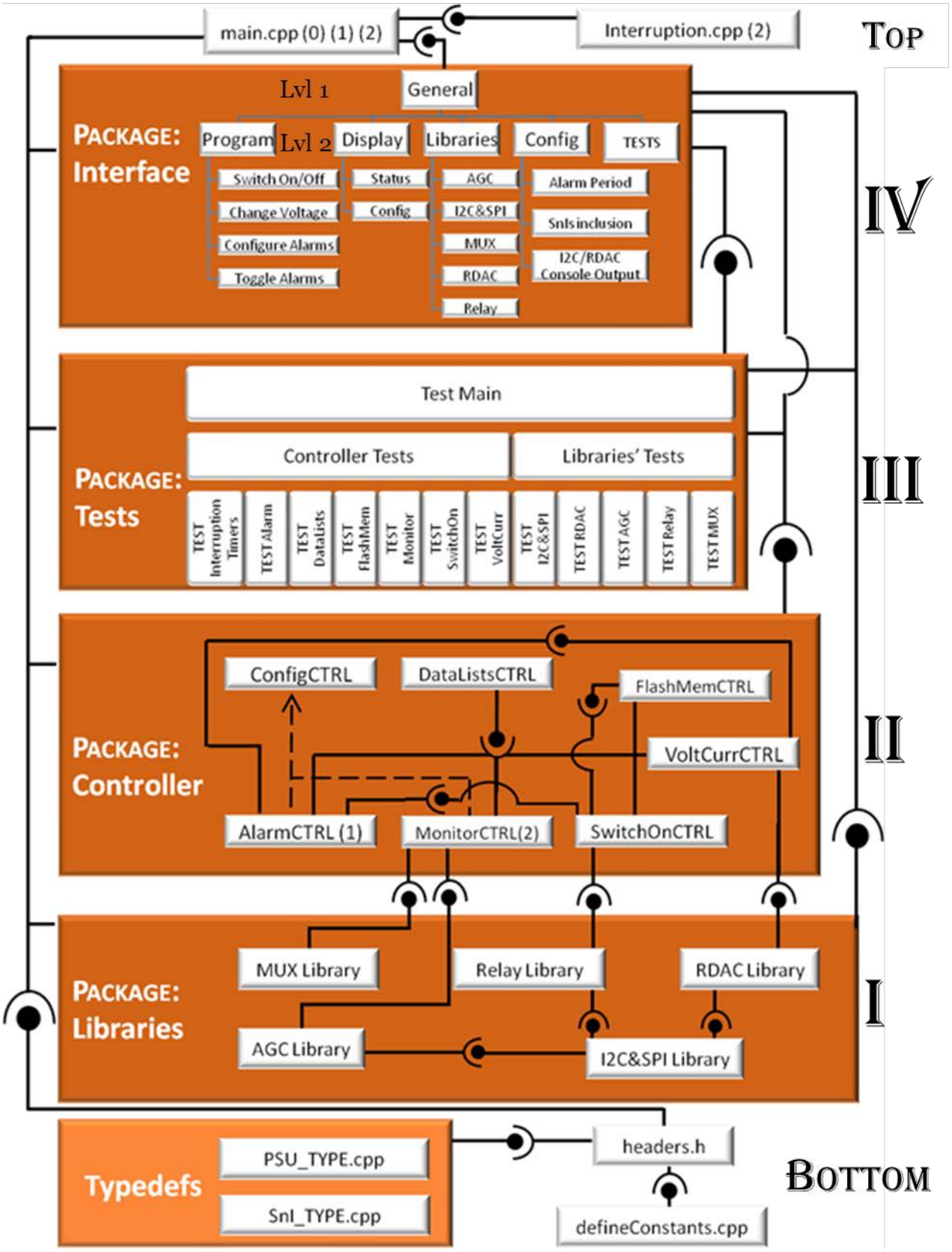


Figure 15 - Development diagram

7.2 BOTTOMMOST LAYER – DEFINITIONS AND HEADERS

7.2.1.1 Typedefs: PSU_TYPE and SnI_TYPE

The foundations of the whole system are the definitions of "c objects" or typedefs to host all the data (alarms, configuration and measured values) for PSUs and SnIs. These are defined in PSU_TYPE.cpp and SnI_TYPE.cpp. The objects are created for both in two separated lists/arrays in DataLists Module (Controller package).

Both objects share the same alarm variables structure, while differing in other configuration values (e.g. only the PSUs have timers for initialization). Auxiliary variables can't be accessed by the user (as they are used by the tasks).

Alarm arrays are defined for four different alarm categories: INFERIOR/SUPERIOR thresholds (limits to be reached/ not exceeded in order to avoid alarm triggering) and VOLTAGE/CURRENT (magnitude type being measured). The alarm arrays have been defined as unidimensional for easier processing, with access methods that portrait the alarm it's related to (*e.g. alarmWatch[_(SUPERIOR, VOLTAGE)] refers to the same value as alarmWatch[1]*)

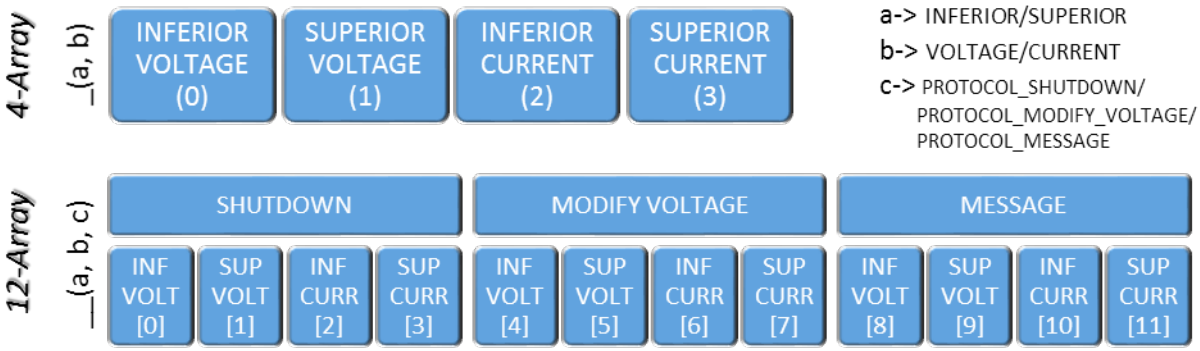


Figure 16 - Alarm arrays for PSU_TYPE and SnI_TPYE

defineConstants.cpp

Contains most of the #define statements, which set the initial status of system configuration parameters, addresses for HW components (I2C and SPI) , PSU/SnI default values, and simplifies the syntax of certain dense methods in PSUControl.

Headers.h

Contains all the external files required for the system: imported files from C and C Netburner libraries. Implements Typedefs and defineConstants.cpp, and defines an interface that is implemented by all the upper layers.

Table 8 - PSU_TYPE and SnI_TYPE atributes

Category	Variable Type	Variable Name	Description	Used by Package	PSU/ SnI
General Status	BOOL	**psuStatus**	PSU connected and functioning (T) or disconnected and with minimum voltage (F)	SwitchOnCTRL	PSU
	BOOL	**relayStatus**	PSU's Relay connected (T) or disconnected (F)	RelayLibrary	PSU
	float	**progValue**	Programmed value for a PSU's RDAC when psuStatus=ON	VoltCurrCTRL	PSU
	BOOL	**sniStatus**	SnI defined as always connected and functioning (T)	-	SnI
	float	**nominalVoltage**	Theoretical voltage value for the voltage source. Detailed list in MUX HW Module.	-	SnI
Addressing	BYTE	**bridgeI2CAdr**	I2C Address of the I2CtoSPIBridge embedded in the PSU's Functional SlotCard (same for each two PSUs).	I2C&SPILibrary	PSU
	BYTE	**rdacAdr**	SPI Address of the PSU's RDAC.	I2C&SPILibrary	PSU
Alarm Arrays	float[4]	**alarmLimitValues**	If measured value exceeds/doesn't reach these values, the alarm task will trigger an alarm	AlarmCTRL	PSU/ SnI
	float[4]	**alarmLimitTimes**	number of measures that have to exceed/not reach the limit values to trigger an alarm	AlarmCTRL	PSU/ SnI
	BOOL[12]	**alarmProtocols**	Activate or ignore an alarm protocol (Shutdown, modify voltage, send message) when an alarm is triggered	AlarmCTRL	PSU/ SnI
	WORD[4]	**alarmProtocol-Shutdown**	PSUs to be switched off when alarm protocol shutdown is executed	AlarmCTRL	PSU/ SnI
	float[4]	**alarmProtocol-Voltage**	New voltage value to be programmed when alarm protocl modify voltage is executed	AlarmCTRL	PSU
	int[4]	**alarmCounters**	AUXILIARY. Used by alarmTask to count the number of times a measure has exceeded/not reached the limit values	AlarmCTRL	PSU/ SnI
	BOOL[4]	**alarmStatus**	AUXILIARY. Set ON when alarmCounters reach the limit times, triggering the protocols.	AlarmCTRL	PSU/ SnI
	BOOL[4]	**alarmLimitReached**	AUXILIARY. Last received measure exceeds/doesn't reach the limit value	AlarmCTRL	PSU/ SnI
	BOOL[4]	**alarmWatch**	Alarm surveillance connected (Auxiliary values are updated and alarms can be triggered) or disconnected (Auxiliary values left to "0").	AlarmCTRL	PSU/ SnI
Resistors	int	**rShunt**	Internal Resistor value (see Functional Slotcards) used to generate a voltage value.	-	PSU
ADC	float	**vOut**	measured Voltage obtained from monitorTask	MonitorCTRL	PSU/ SnI
	float	**cOut**	measured Voltage obtained from alarmTask	MonitorCTRL	PSU
Initialization	int	**initializationTimer**	x100ms initial count for PSU switching-on	SwitchOnCTRL	PSU
	BOOL	**ReadyToConnect**	AUXILIARY. Used for switch on.	SwitchOnCTRL	PSU
FLASH	DWORD	**VerifyKey**	AUXILIARY. Used for data verification	FlashMem-CTRL	PSU/ SnI

7.3 LAYER I – LIBRARIES PACKAGE

The libraries define the lowest level of PSUControl, posing as communication protocols between hardware elements and the MCF5213. Most of them implement I2C&SPI Library, as they have to communicate with hardware components connected to the I2C Bus (such as I2C to SPI Bridge or I2C bus expander). All the libraries are related to a HW module previously defined in section Hardware System Overview.

There is one extra library not being considered as part of the system: *General Library*. This small module defines general purpose methods that couldn't be found in C libraries (such as float to ASCII conversion, int to ASCII conversion, hamming weight calculus). Due to their auxiliary nature, General Library is left unanalysed.

7.3.1.1 I2C&SPI Library

Communication protocols using I2C bus. It includes communications across an I2C to SPI Bridge, arriving into SPI buses (used to communicate with RDACs). Clarification: The SPI communication does not use the MFC-5213 in-built pins defined for that purpose (as no direct SPI connection is carried out between MCF5213 and any other HW). No functions have developed for that purpose.

7.3.1.2 RDAC Library

Configuration and Reading of both Control and Value registers for Digital Rheostats AD5292 sending SPI messages through an I2C to SPI Bridge (implements I2C&SPILibrary). Further inquiries can be pointed to AD5292 Datasheet. An example of RDAC configuration using I2C and SPI buses is found in the section Logic View, subsection Communications.

7.3.1.3 AGC Library

Automatic Gain Control based on a RDAC and an operational amplifier, with gain values between 1.1 and 101 (explained in Hardware System Overview). It converts the desired Gain value into RDAC counts and sends the proper configuration messages through I2C Bus (implements I2C&SPILibrary) to configure the RDAC_AGC properly.

7.3.1.4 MUX Library

It configures all the multiplexers using eight MCF5213 pins set as GPIO (General Purpose Input Output). There are 2 steps (as stated in Hardware System Overview):

- Selecting the desired output for multiplexers A to E from various voltage inputs (all muxes simultaneously configured).

- Selecting the desired output of MUX_ADC (whose inputs are the outputs of multiplexers A to E and a loose magnitude).

7.3.1.5 Relay Library

Connects/Disconnects relays for all twelve PSUs. PSUs 0 to 5 (SF1_A to SF3_B) are controlled by six GPIO pinsfrom the MCF5213, while PSUs 6 to 11 (SF4_A to SF6_B) are switched using an I2C Bus expander (implements the I2C&SPI Library).

7.4 LAYER II – CONTROLLER PACKAGE

Contains the true functionality of PSUControl, where most User Requirements are made to happen [12]. They implement all the Libraries, using them for interacting with hardware modules. All the modules implement DataLists CTRL, because it contains the arrays of PSU/SnI objects where all the information regarding configuration, voltages, alarms, etc… is stored.

7.4.1.1 DataLists CTRL

Definition of an array of **twelve** PSU_TYPE objects to accommodate data regarding PSUs, and an array of **fourteen** SnI_TYPE objects to house data related to SnIs. Methods to set default values and print in console for both object types, as well as Getters & Setters for all their attributes.

7.4.1.2 FlashMem CTRL

Use of flash memory section in the MCF5213 to save and restore both data Lists, making them available after a system shutdown. Checks for data integrity for each data object before loading, setting default values if its data is corrupted. Updates data lists in RAM with all the previous values.

7.4.1.3 VoltCurr CTRL

Adjusts output voltages for all PSUs configuring their RDACs and checking for correct value updates. Single reads of PSUs and SnIs (both voltage and current). Conversion between A2D discrete scale and Voltage/Current Values importing the scale factors from MUX Library (which only depends on the PSU/SnI number) and AGC Library (which varies in time). Updates data lists in RAM with programmed voltage values.

7.4.1.4 SwitchOn CTRL

Switch on process with a configurable waiting period before connecting (based on *initializationTimer*) and a programmed voltage update at start-up. It also switches off PSUS and sets them into lowest voltage, minimizing power consumption when idle. Relay connection & disconnection. Updates data lists in RAM with ON/OFF status, and relay status (connected/disconnected). Implements Flash Mem module in order to initialize values at start-up.

7.4.1.5 Monitor CTRL

Periodical sampling task (1) of all the magnitudes defined in MUX HW Module. Sets the muxes at a different magnitude each time, and sets the AGC so as to obtain a high value in the A2D discrete scale. Then, it uses MCF5213 A2D controller to obtain a discretization of the selected magnitude, converts it to the real value using VoltCurrCTRL conversion (implements VoltCurrCTRL), and Updates data lists in RAM with measured voltages (*vOut*) and currents (*cOut*). Is dependent of ConfigCTRL. Further explanation of this process is found in Process View section.

7.4.1.6 Alarm CTRL

Task executed once every 100ms that checks the voltage/current of all the PSUs and SnIs and triggers alarms if a programmed limit is reached. Executes alarm protocols when an alarm is triggered, such as shutting down a selection of PSUs, adjusting the voltage for the local PSU (the one triggering the alarm) and sending a console message to warn the user. Updates data lists in RAM with several alarm auxiliary variables.

It is also in charge of the external button (control of button to switch on/off all the PSUs). Implements VoltCurr and SwitchOn modules so to execute alarm protocols, and is dependent of Config CTRL. Further explanation of this process is found in Process View section

7.4.1.7 Config CTRL

Definition and modification of several configuration values (alarm update period, SnI included/excluded from alarmTask and monitorTask). Methods to pause alarmTask and monitorTask (setting them into idleMode and sequentialMode respectively). Methods to set testMode for monitorTask and SwitchONPSUsTask (recording their status throughout the process and printing extra information in console). It does NOT affect data lists. Has control over Alarm CTRL and Monitor CTRL.

7.5 LAYER III – TESTS PACKAGE

To build a robust system it's required to test every module in order to check for proper functioning. To do so, a Test package has been developed, containing an in-depth test of each module from Libraries and Controller Modules. Full details of all the tests are depicted in section Test Cases.

7.5.1 LIBRARIES TESTS

Since libraries define communication protocols with HW module, libraries' tests require the connection to all the HW system. Also, because of some libraries implementing others, there is a recommended order for test execution:

TEST_I2CnSPI → TEST_RDAC → TEST_AGC → TEXT_MUX →TEST_Relay

Even though these tests are aimed at Libraries, they eventually call certain functions from Controller Package in order to validate the library's functionality.

7.5.2 CONTROLLER TESTS

Controller Tests are software-based, demanding only a MCF5213 hardware environment to be properly executed, with little to none calls to Libraries. Again, a specific order is enforced so as to bring to pass the interface implementations within the Controller Package, leaving the modules where tasks are defined for last.

TEST_InterruptionTimers → TEST_DataLists → TEST_FlashMem→ TEST_VoltCurr →
TEXT_SwitchOn →TEST_Monitor → TEST_Alarm

The functionality of the Config module gets implicitly validated in the controller tests, which is why no test has been defined for this module.

7.6 LAYER IV – INTERFACE PACKAGE

The definition of the UI (User Interface) is carried out on this package [12]. MTTTY utility software from Netburner is used to show console messages and receive PC keyboard input. This package implements all the other packages, offering most of their methods as options in different menus.

In order to link the MTTTY (windows utility console) with the MCF5213, the port where the USB connection has been set is selected (COM1 in this case, but it varies), with a baud rate of 115200, parity "None", 8 Data bits and 1 stop bit, and then press "Connect".

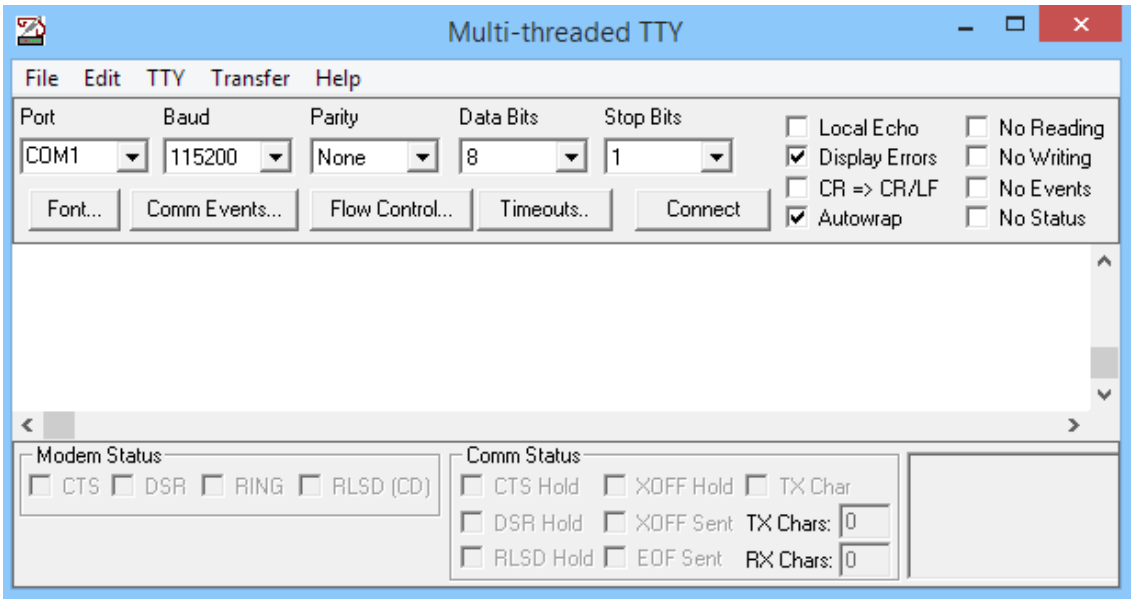
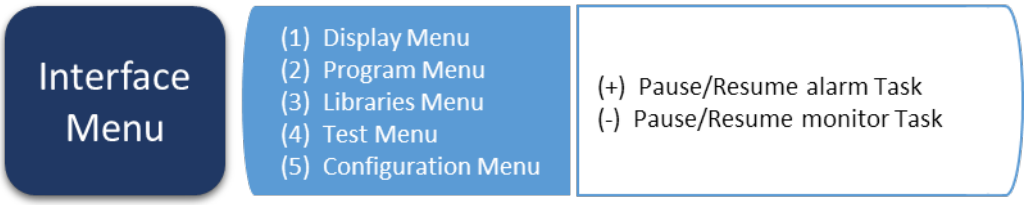


Figure 17 - MTTTY Console

The UI has been designed as a hierarchy of menus with three menu levels. The menus are presented in a self-explanatory diagram, with navigation between menus set in blue background and Functions in white. Option selection is carried out by pressing the number/sign in brackets on the PC keyboard.

7.6.1 GENERAL/INTERFACE MENU – FIRST LEVEL



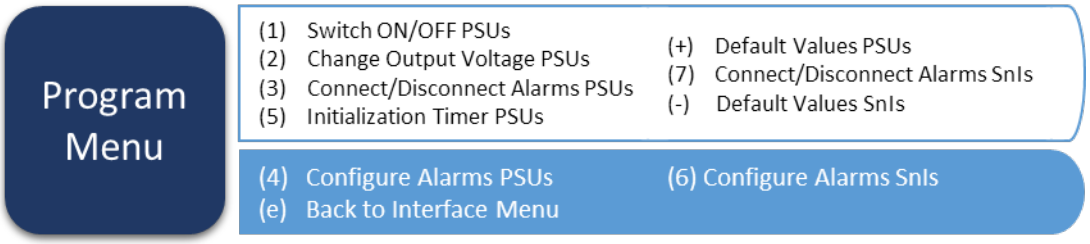
Grants access to all second level menus. Allows Pause/Resume of alarm and monitor tasks.

7.6.2 DISPLAY MENU – SECOND LEVEL

Menu used to check the current system status in a console window size (no need to scroll). Should be used as default when no programming or testing is being executed.

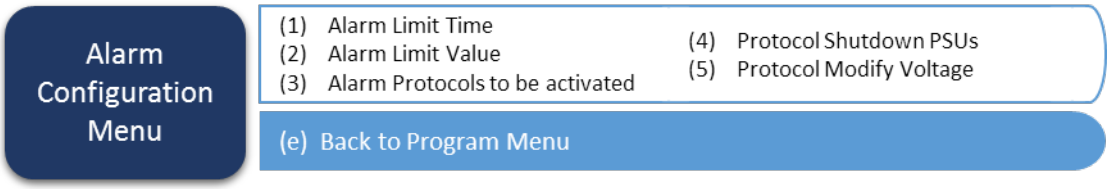
(1) Display all the read values from voltages and currents, as well as alarms' status
(2) Display the configuration values (limiting values, timers) for PSUs.
(3) Display the configuration values (limiting values, timers) for SnIs.

7.6.3 PROGRAM MENU – SECOND LEVEL



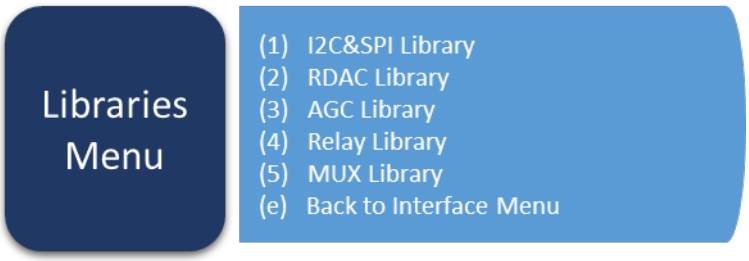
Most important menu, as it incorporates all the functionalities from Controller Package. All the functions include multiple selection steps, in order to quicken system manipulation (PSU/SnI selection, Alarm selection, and so on).

7.6.3.1 Alarm Configuration Menu – Third Level



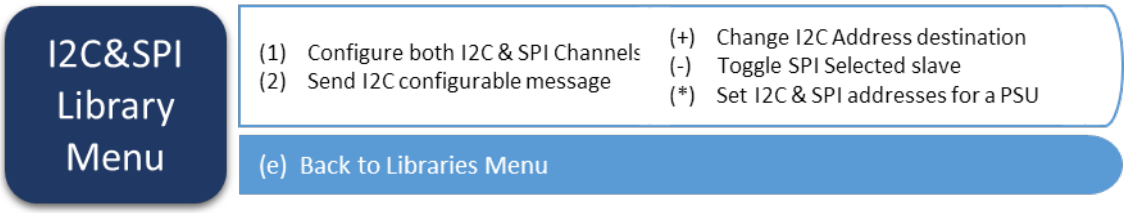
Programming of all the configurable values related to alarms (same for PSUs and SnIs).

7.6.4 LIBRARIES MENU – SECOND LEVEL



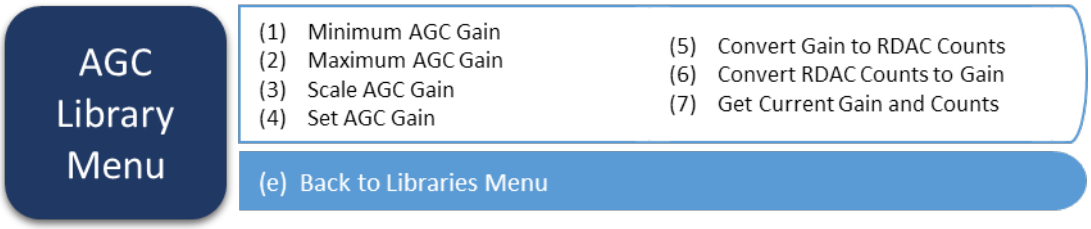
Grants access to all the libraries' functions, subdivided into a menu for each library.

7.6.4.1 L2C&SPI Library Menu – Third Level



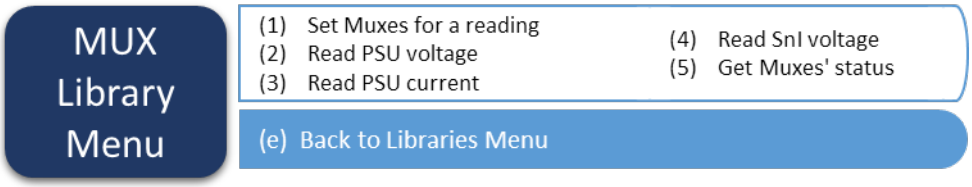
Send a message to a selected I2C address, as well as to an SPI selected Slave (configuration SPI messages for RDACs are included in RDAC Library).

7.6.4.2 AGC Library Menu – Third Level



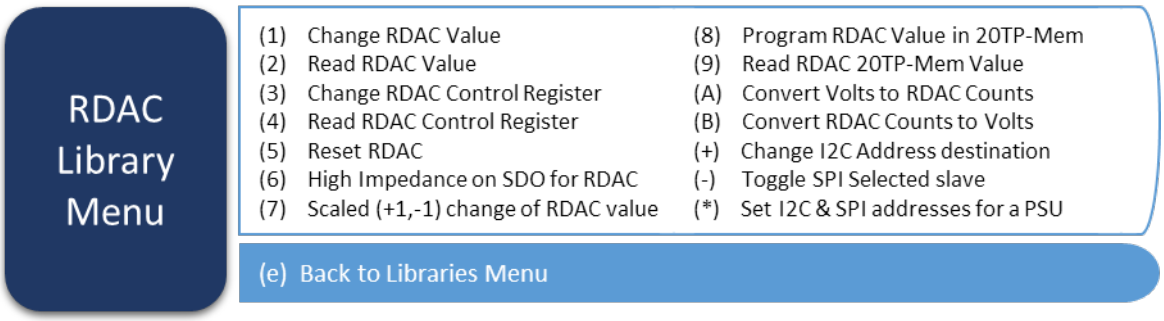
Sets a gain from 1.1 to 101 in the AGC module, configuring the RDAC_AGC.

7.6.4.3 MUX Library Menu – Third Level



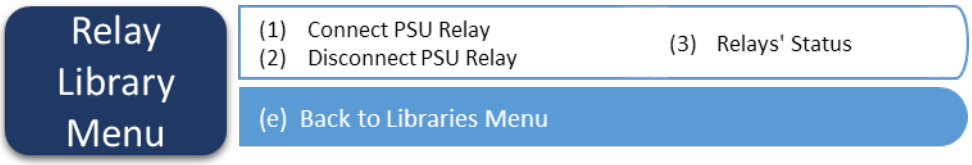
MUX configuration, as well as reading a single measure of voltage current (using methods from VoltCurrCTRL).

7.6.4.4 RDAC Library Menu – Third Level



Use of all the functions defined in RDAC Library to configure and read the RDAC, based on the datasheet from AD5292. Most important ones (used in Controller Package) are the Read/Change methods for both Value and Control registers.

7.6.4.5 Relay Library Menu – Third Level



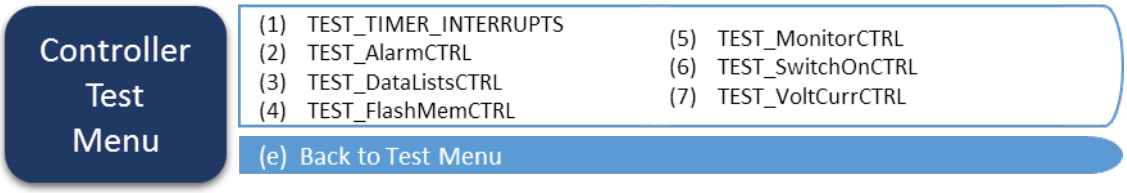
Connect and disconnect Relays, as well as check for relays' status.

7.6.5 TEST MENU – SECOND LEVEL

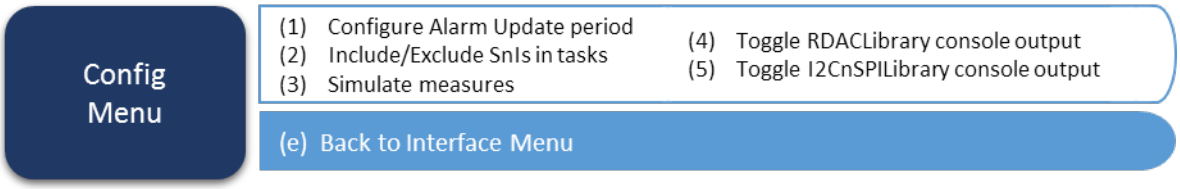
Grants access to module tests (implementing all the Test Package), separated into Libraries and Controller.

7.6.5.1 Libraries' Test Menu – Third Level

7.6.5.2 Controller Test Menu – Third Level



7.6.6 CONFIG MENU – SECOND LEVEL



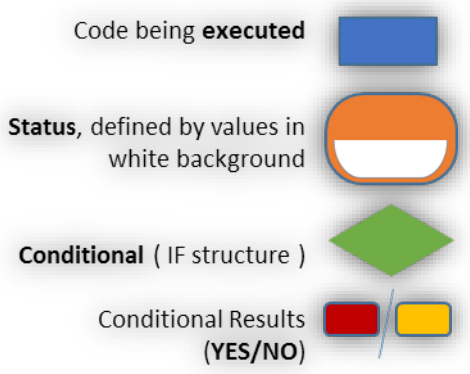
Configure tasks, toggle console output in communication-related libraries.

8 PROCESS VIEW

This section details the most important processes of PSUControl. As stated in the Development View, the system is formed by three independent tasks running in parallel:

- Main Task: in charge of the UI. Already explained in the Layer IV of Development View.
- Alarm Task: alarm updating and triggering every 100 ms, with higher priority than Main.
- Monitor Task: measuring of voltages/currents once every millisecond. Highest priority.

Aside from the tasks, the switching on process also holds enough importance to be explained.



8.1 ALARM TASK

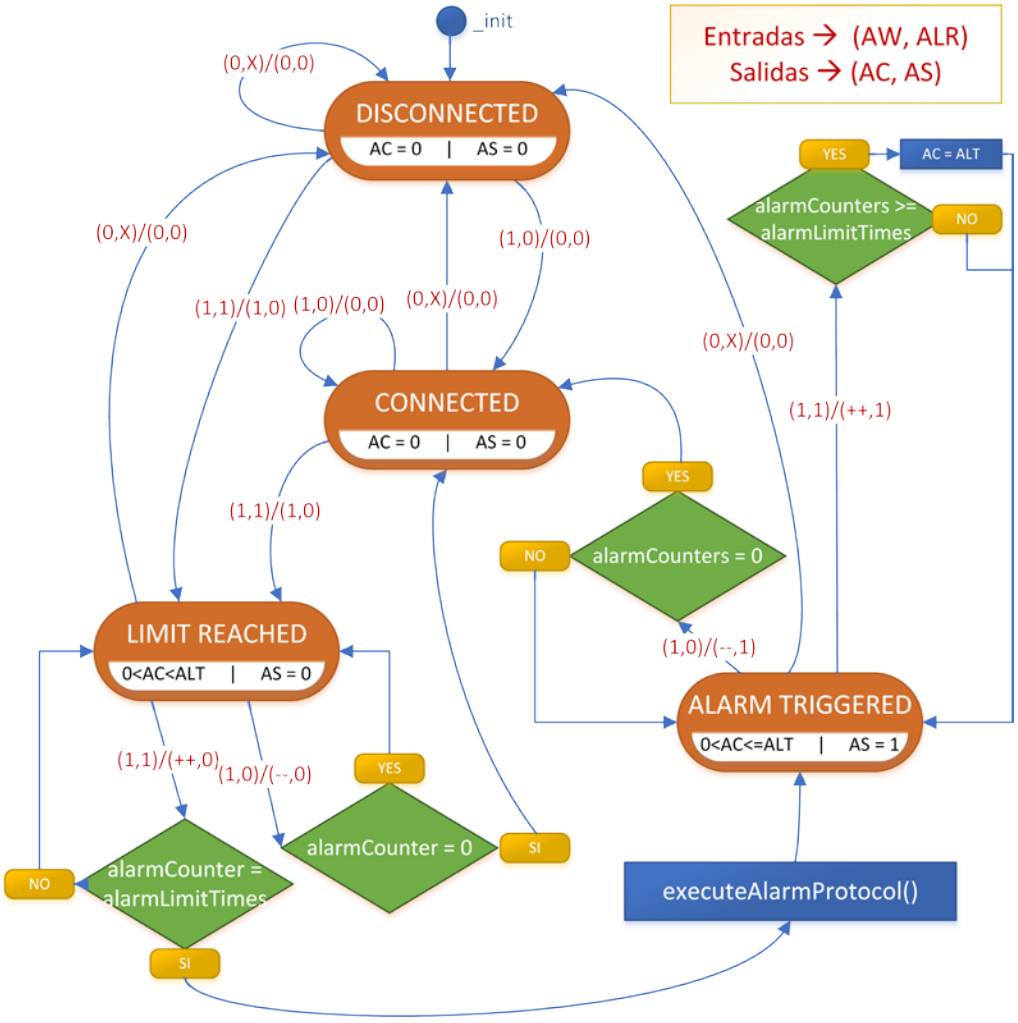


Figure 18 - Alarm Task status diagram

This process is carried out for voltages and currents in PSUs, and voltages in SnIs. All those magnitudes are contrasted with a superior and an inferior limit, for a total of six consequent subprocesses based on this activity diagram.

Once every 100 ms, the magnitudes are compared with the limits, and on each case a position of ALR array is updated (check typedefs in Development view), setting that position to TRUE if the limit is surpassed (superior) or not reached (inferior), and FALSE otherwise. With that input and the AW information, the alarm task advances once.

All the alarms begin at ***Disconnected*** status. Once the User configures AW to 1 for a certain alarm (PSU/SnI, PSU or SnI number, VOLTAGE/CURRENT, INFERIOR/SUPERIOR), the alarm is set into ***Connected*** status, checking ALR every period. When ALR contains a TRUE, the alarm changes to ***Limit Reached*** status, increasing the AC every time an ALR TRUE is received, and decreasing it when ALR is FALSE. If AC was to exceed ALT, the alarm would be triggered, setting AS to TRUE and executing the programmed protocols (send console message, shutdown several PSUs, change voltage to the local PSU). The ***Alarm Triggered*** status would be exited when AC decreased until reaching 0, going back to ***Connected*** Status.

alarmLimitReached – ALR
Boolean that activates if the measured voltaje current (vOut/cOut) surpasses/doesn't reach the limit.
alarmWatch - AW
Control Boolean – Alarm is set to disconnected status when set to false.
alarmCounter - AC
Number used to count the number of times ALR is set to 1.
alarmStatus - AS
Boolean used to define the "Alarm Triggered" status.
alarmLimitTimes - ALT
Number to be reached by AC in order to trigger the alarm and execute protocols.

8.2 MONITOR TASK

Sampling all the 38 magnitudes defined in MUX HW module, one at a time with a period of 1ms, in a never-ending cyclic task. The task begins at PSU voltages. In all the cases the procedure is the same:

1. Read a value from the A2D and check if is in the accepted range (30% to 95% of the A2D scale), configuring lastMeasureValid (the only input to the diagram).
2. If the measure is valid, it's stored in the corresponding variable (*vOut* for voltages, *cOut* for currents) inside the set PSU/SnI object (defined by Num and Sampling Function). Then, the AGC is set to minimum gain and the muxes are configured for the next reading.
3. If the measure isn't valid, the ACG gain is scaled so to get the next reading in the center of the range (about 60% of A2D scale).

The process is executed twelve times for PSU voltages, twelve times for PSU currents, and fourteen times for SnI voltages.

Num
Number of the PSU/SnI to be measured
samplingFunction
Measure type (Volt/Curr; PSU/SnI)
Store Measure
Stores the A2D register scalating it properly (scale factors in MUX HW module).
minGainAGC()
Configure AGC to minimum Gain
changeGainAGC()
Configure AGC to set the next simple in the middle of the scale.
setMUX()
Configure muxes for next reading
lastMeasureValid
True if last measure is in accepted range or max AGC gain is reached.

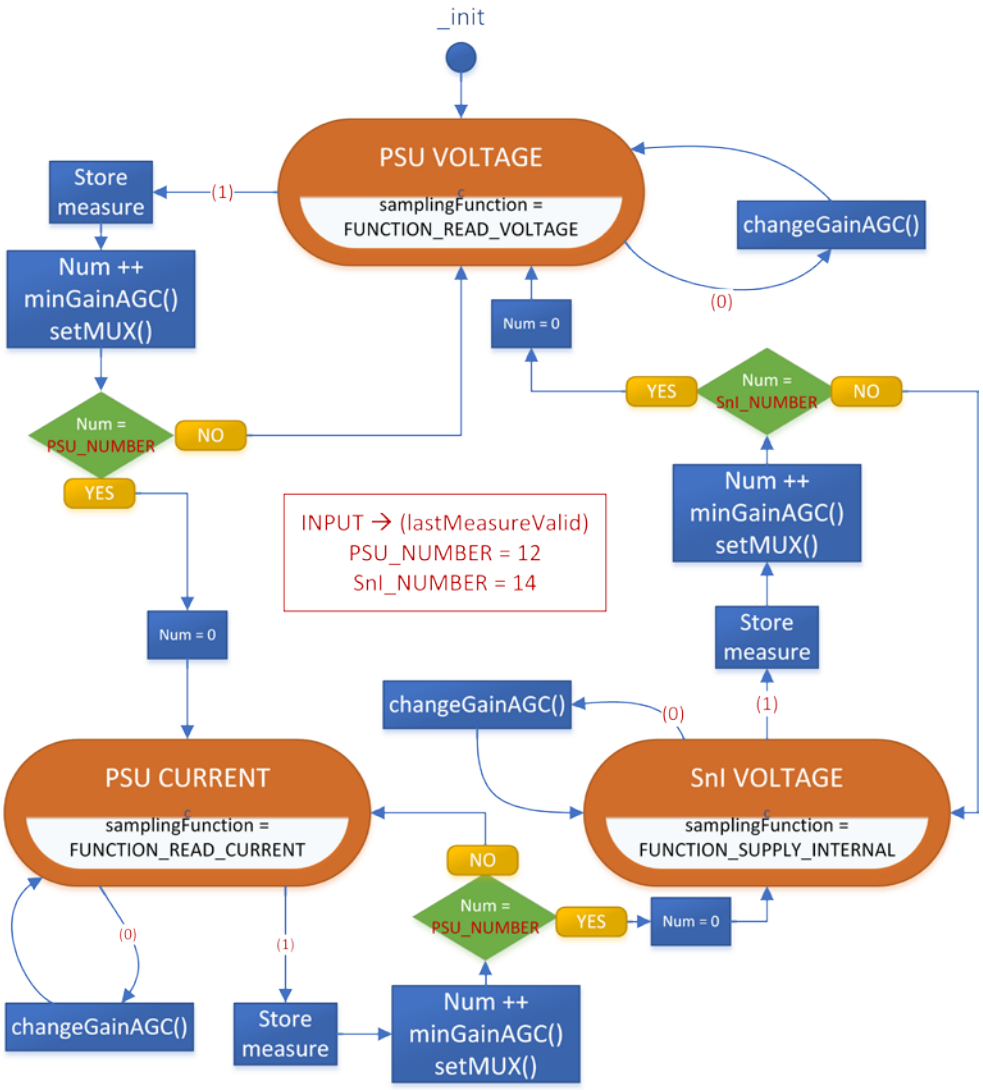


Figure 19 - Monitor Task status diagram

8.3 SWITCH ON PROCESS

The switch on process is in charge of connecting the PSUs and configuring their initial voltages. This process, as an opposite of alarmTask and monitorTask, is non-recurrent (has a beginning and an end). The process is called upon system initialization, and when the user chooses to connect several PSUs (option 1 of the Program menu in the UI – see Development View Layer IV). The process follows this execution sequence:

1. Pause monitorTask and alarmTask until completion.
2. Initialize values for PSUs and SnIs, loading the values stored in Flash Memory and checking for their validity; values are set to default for each invalid object.
3. Update voltage PSUs: configures the PSUs' output voltages to match the stored values.
4. Starts the connection process: adds one to a counter (ticks) every 100 ms, and when it reaches the initializationTimer of a PSU, this PSU's relay is connected. This subprocess ends

when all the PSUs that were being initialized are connected (the diagram shows the example for all the PSUs).

5. Resumes the tasks once the process is ended.

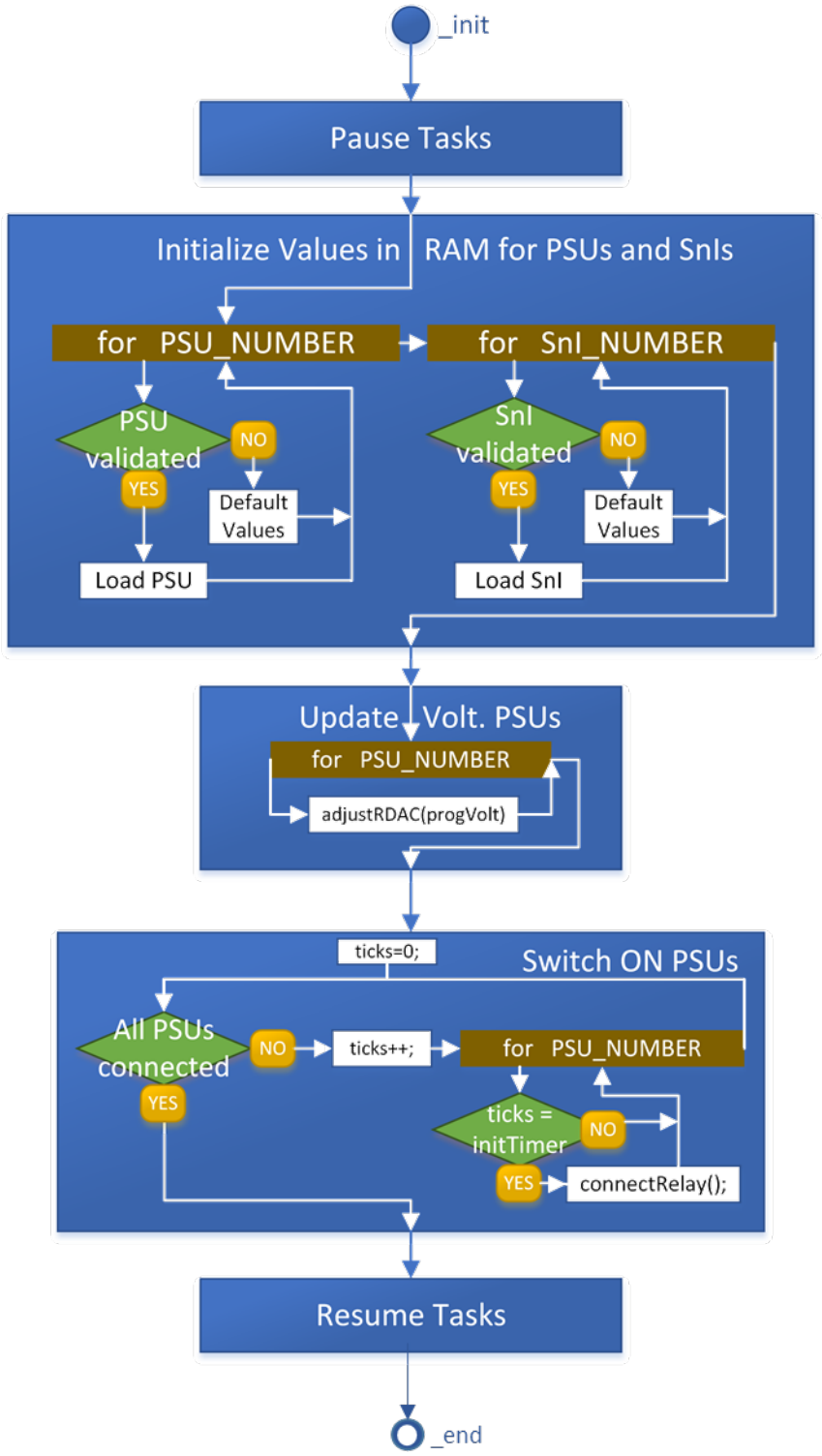


Figure 20 - Switch On activity view

9 LOGIC VIEW

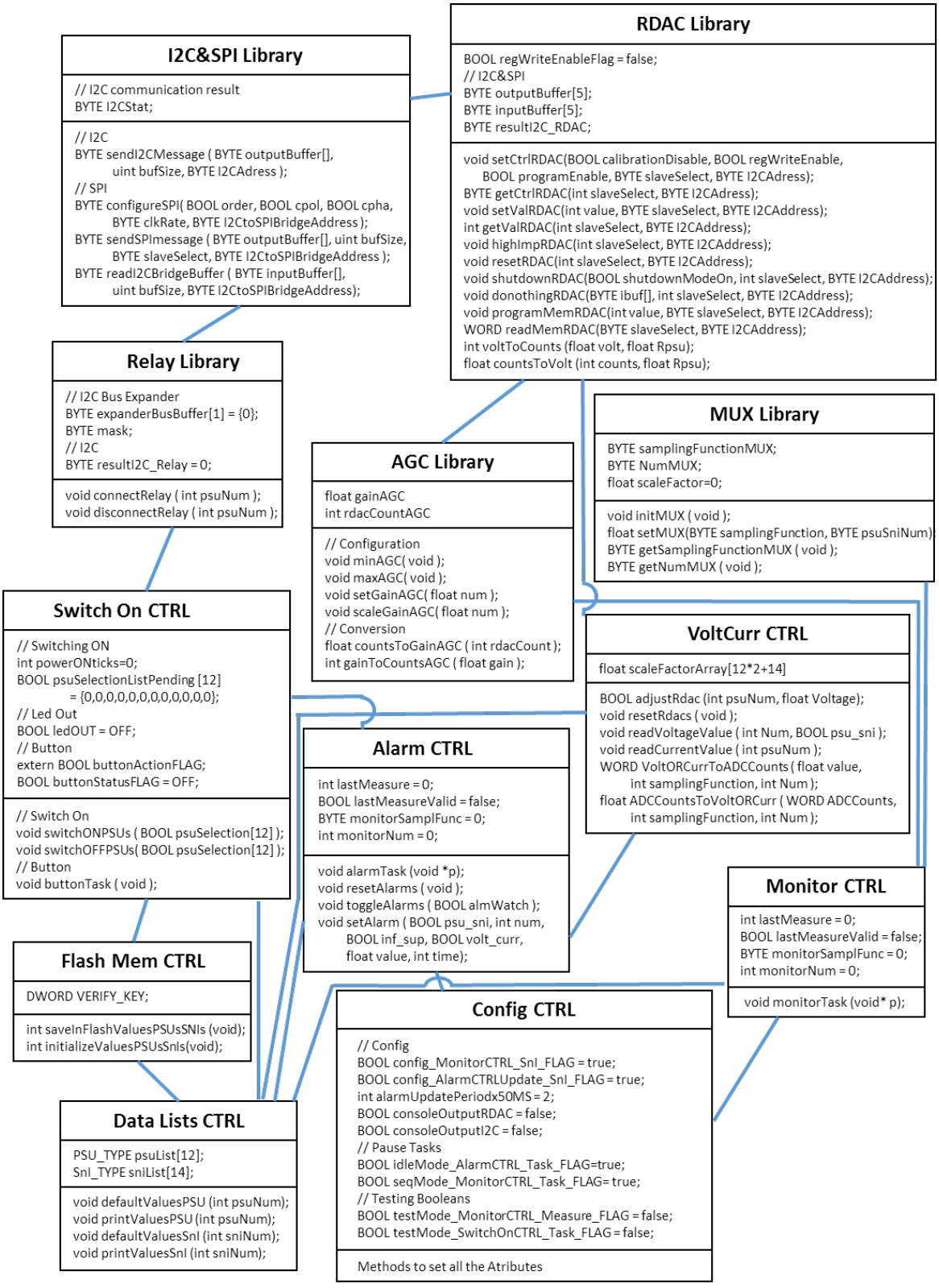


Figure 21 - Class Diagram

The Class diagram of PSUControl portrays the main attributes (below the module title) and methods (below the attributes) of the software modules. As a design choice, only the Libraries and Controller package are depicted, which are the ones containing the functionality.

9.1 COMMUNICATIONS

This section represents two examples of communications via I2C Bus with the two purposes they're used for in PSUControl: RDAC communications and relay connection/disconnection.

9.1.1 RDAC PROGRAMMING AND READING

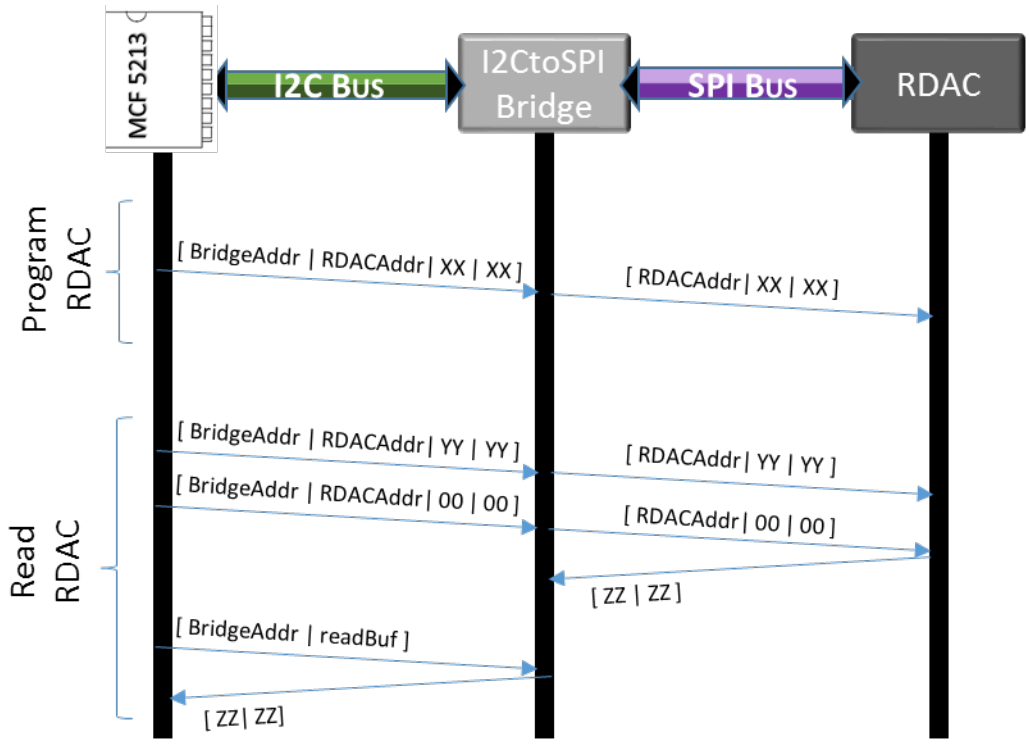


Figure 22 - I2C and SPI Communication diagram

In order to **program** a RDAC (e.g. set a new value), an I2C message with 4 Bytes is sent. The first byte contains the I2C direction of the I2CtoSPIBridge located in the Functional slotcard "n" (these directions vary from 0x28 to 0x2F). The second byte can either be 0x01 or 0x02, referring to the SPI direction of the RDAC in SFn_A or the one in SFn_B. The last 2 Bytes (XX|XX) contain the order to program the RDAC, extracted from the datasheet of AD5292.

Reading the value of an RDAC involves several messages. The first one, with a composition similar to the one used for programming, contains in the last two bytes (YY|YY) the order for the RDAC to send his configuration value in the next message. A second message, with the order bytes left empty, makes the RDAC to answer with its configured value (ZZ|ZZ) and send it to the I2CtoSPIBridge's buffer. Finally, the MCF5213 asks the I2CtoSPIBridge for the contents of its buffer, retrieving the RDAC configured value.

The communication protocols are defined in I2C&SPI Library, while the methods to program and read RDACs are implemented in RDAC library. This type of communication is used to configure the RDAC that adjust the voltage in each PSU, as well as for the RDAC embedded in the AGC module (and controlled by AGC Library).

9.1.2 RELAY CONFIGURATION USING I2C BUS EXPANDER

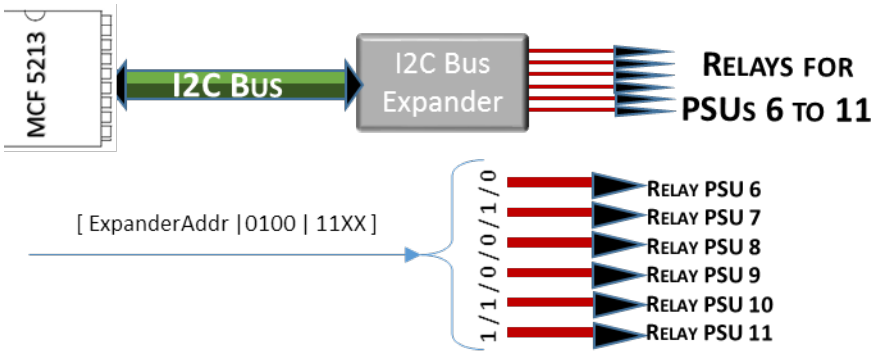


Figure 23 - Relay configuration using I2C Bus Expander communication diagram

The I2C bus is employed to connect the relays for PSUs six to eleven (SF4, SF5 and SF6). To do so, a two byte message is sent. The first byte contains the I2C address of the bus expander (0x38). The remaining byte is processed by the expander. This component acts as a demultiplexer, setting its eight outputs with the value of the eight bits transmitted in the second byte. The first six outputs are controlling the relays of PSUs 6 to 11: a value of “1” will connect the relay, while a value of “0” will disconnect it.

GPIO pins are used to configure the relays for PSUs 0 to 5 (SF1, SF2 and SF3).

The communication protocols with relays are implemented in Relay Library, who takes over the job of simplifying this process by offering a single interface for both relay configuration methods.

10 TEST CASES

Test Cases (TC) are sequences of actions that are defined to determine if the system was built according to the previously defined System Requirements (it does what it should do). Test cases may be designed for software **verification** (follow concrete steps and obtain the desired result, thus complying a SR; used with functionalities) and software **checking** (general evaluation of the fulfilment of a defined SR; used with UI heuristics and other abstract SRs). Every SR excluding the ones left for future implementation is connected to at least one TC. The Test Cases are divided into the same types as the System Requirements (Interfaces, Functional and Complementary). Additional documentation of several TCs (detailed information, input and output, detailed results report, date of the verification) is included in the GtiHub repository.

The Test Cases may serve as a useful tool to measure the progress made in PSUControl. With this idea in mind, each TC is assigned a status level:

- NOT STARTED: The TC hasn't been designed yet.
- DESIGNED: The TC has been methodically described in all his steps, and the software tests required have been programmed. The TC may have been executed with negative results.
- VERIFIED/CHECKED: The TC has been designed and executed, and their results were positive.

Many Test Cases rely on the execution of a programmed **test** from Test Package (Layer III in Development View). They are accessible via the UI, and contain several sections where different aspects of the module it's linked to are tested (the connection between a test and the module being tested is defined in the name of the test, e.g. TEST_MonitorCTRL is linked to Monitor module from Controller Package). The result of each section can either be "*PASSED*" or "*NOT PASSED*". A correctly passed test from Test Package will end with: "*OVERALL RESULT: PASSED*".

The **environment** required for a Test Case goes along with its nature. Four environments are defined: ***SOFTWARE*** (where only the MCF5213 connected to a PC is needed), ***SF*** (all the Software environment plus a Functional slotcard connected to the MCF5213 via the I2C bus), ***FULL*** (the whole system) and ***VOLTMETER*** (SF with an external measure of the SF's output voltages).

10.1 TYPE INTERFACES

Table 9 - TC Type Interfaces/I

CATEGORY	SR ID	NAME	STATUS
HARDWARE	TC_I_100	TC_I_HARDWARE_GENERAL	-
	TC_I_101	TC_I_HARDWARE_COMMUNICATIONS_I2C	**VERIFIED**
	TC_I_102	TC_I_HARDWARE_COMMUNICATIONS_SPI	**VERIFIED**
SOFTWARE	TC_I_200	TC_I_SOFTWARE_GENERAL&COMMUNICATIONS	-
	TC_I_201	TC_I_SOFTWARE_USER_INTERFACE	**CHECKED**
NIELSEN HEURISTICS FOR WINDOWS INTERFACE	TC_I_300	TC_I_HEUR_STATUS_VISIBILITY	**CHECKED**
	TC_I_301	TC_I_HEUR_UNDO_REDO	**DESIGNED**
	TC_I_302	TC_I_HEUR_STANDARD	**DESIGNED**
	TC_I_303	TC_I_HEUR_ERROR_PREVENTION	**NOT STARTED**
	TC_I_305	TC_I_HEUR_ACCELERATORS	**VERIFIED**
	TC_I_307	TC_I_HEUR_ERROR_MESSAGES	**CHECKED**
	TC_I_308	TC_I_HEUR_HELP	**VERIFIED**

10.1.1 HARDWARE

10.1.1.1 TC_I_100 – TC_I_HARDWARE_GENERAL

Hardware verification is out of the scope of PSUControl, leaving this TC to the hardware designer.

10.1.1.2 TC_I_101 – TC_I_HARDWARE_COMMUNICATIONS_I2C

SF Monitoring the I2C bus with the oscilloscope while transmitting any message. Use "send I2C Message (2)" from I2C Library menu (Libraries menu in the UI).

10.1.1.3 TC_I_102 – TC_I_HARDWARE_COMMUNICATIONS_SPI

SF Monitoring the SPI bus with the oscilloscope while transmitting any message. Use "Reset RDAC (5)" from RDAC Library Menu (Libraries Menu in UI) after selecting the RDAC's PSU in this same menu (option "*").

10.1.2 SOFTWARE

10.1.2.1 TC_I_200 – TC_I_SOFTWARE_GENERAL&COMMUNICATIONS

No verification is performed for the software platforms. Verification of software communications is left for Functional TC.

10.1.2.2 TC_I_201 – TC_I_SOFTWARE_USER_INTERFACE

SOFTWARE Navigation across all the defined menus, testing that all the defined options are accesible, and that an instructions list accompanies the menus where its use is unclear. Only vital information is displayed.

10.1.3 NIELSEN HEURISTICS

10.1.3.1 TC_I_300 – TC_I_HEUR_STATUS_VISIBILITY

SOFTWARE Enter the Display Menu in the UI and constantly press "Status Display" (1), measuring the number of refreshes in a period of 20 seconds. Obtained value should be above 15. Same procedure with the "Config Display" (2)

10.1.3.2 TC_I_301 – TC_I_HEUR_UNDO_REDO

SOFTWARE Enter the Program Menu in the UI and use every option at least once, exiting in the middle of the selection process. Undo and Redo the last executed action. *SR not implemented yet.*

10.1.3.3 TC_I_302 – TC_I_HEUR_STANDARD

SOFTWARE Requires of a third party (non-developer) to navigate through all the menus, identifying the conflicting or unclear words and checking if the Term List contains them with an explanation.

10.1.3.4 TC_I_303 – TC_I_HEUR_ERROR_PREVENTION

Due to the easy configuration of all the system elements via UI, the developer has defined all the system as "Critical error safe", presenting no confirmation for using any function. This may change in the future.

10.1.3.5 TC_I_305 – TC_I_HEUR_ACCELERATORS

SOFTWARE Enter the Program Menu in the UI and use a PSU programming option and a SnI one. Shortcut methods when selecting both should function using keys "P" and "O", and it should select/deselect all.

10.1.3.6 TC_I_307 – TC_I_HEUR_ERROR_MESSAGES

SF Indirectly checked while testing the I2C Library, where Error messages ended up being vital to interpret the error and find a solution. A higher logic error module is left as future upgrade in SRs.

10.1.3.7 TC_I_308 – TC_I_HEUR_HELP

Definition of a manual to explain the use of PSUControl (section User Manual of this document).

10.2 TYPE FUNCTIONAL

Table 10 - TC Type Functional/F

CATEGORY	SR ID	NAME	STATUS
MONITORING	TC_F_100	TC_F_MONITORING_SCAN_VOLTAGE	**DESIGNED**
	TC_F_101	TC_F_MONITORING_SCAN_CURRENT	**DESIGNED**
	TC_F_102	TC_F_MONITORING_TASK	**VERIFIED**
	TC_F_103	TC_F_MONITORING_STORE_AVERAGE&PEAKS	**NOT STARTED**
	TC_F_104	TC_F_MONITORING_MUX_LIBRARY	**DESIGNED**
	TC_F_105	TC_F_MONITORING_AGC_LIBRARY	**DESIGNED**
CONFIGURATION	TC_F_200	TC_F_CONFIGURATION_RDAC_LIBRARY	**VERIFIED**
	TC_F_201	TC_F_CONFIGURATION_ADJUST_VOLTAGE	**VERIFIED**
	TC_F_202	TC_F_CONFIGURATION_DATA_LISTS	**VERIFIED**
	TC_F_203	TC_F_CONFIGURATION_FLASH_MEM	**VERIFIED**
	TC_F_204	TC_F_CONFIGURATION_SYSTEM	**CHECKED**
COMMUNICATION	TC_F_300	TC_F_COMMUNICATION_I2C&SPI_LIBRARY	**VERIFIED**
SWITCH-ON	TC_F_400	TC_F_SWITCH-ON_RELAY_LIBRARY	**DESIGNED**
	TC_F_401	TC_F_SWITCH-ON_TASK	**VERIFIED**
	TC_F_402	TC_F_SWITCH-ON_BUTTON	**DESIGNED**
ALARM	TC_F_500	TC_F_ALARM_TASK	**VERIFIED**
	TC_F_501	TC_F_ALARM_PROTOCOLS	**DESIGNED**

10.2.1 MONITORING

10.2.1.1 TC_F_100 – TC_F_MONITORING_SCAN_VOLTAGE

VOLTMETER Use MUX Library Menu (from Libraries Menu in the UI), selecting the option to read a PSU voltage. Then select the option to read a SnI voltage. Both read values should be close to the programmed values.

10.2.1.2 TC_F_101 – TC_F_MONITORING_SCAN_CURRENT

VOLTMETER Use MUX Library Menu (Libraries Menu in the UI), select the option to read a PSU current.

10.2.1.3 TC_F_102 – TC_F_MONITORING_TASK

SOFTWARE Correct execution of software-based test TEST_MonitorCTRL from Controller tests (UI Test Menu). Check the interruption timing periods with TEST_TIMER_INTERRUPTS.

10.2.1.4 TC_F_103 – TC_F_MONITORING_STORE_AVERAGE&PEAKS

Define and execute a software-based test to generate, using *TestMode*, peak values and a known mean value, checking if the system properly stores them. *SR not implemented yet.*

10.2.1.5 TC_F_104 – TC_F_MONITORING_MUX_LIBRARY

FULL Correct execution of test TEST_MUXLibrary from Libraries Tests (UI Test Menu).

10.2.1.6 TC_F_105 – TC_F_MONITORING_AGC_LIBRARY

FULL Correct execution of test TEST_AGCLibrary from Libraries Tests (UI Test Menu).

10.2.2 CONFIGURATION

10.2.2.1 TC_F_200 – TC_F_CONFIGURATION_RDAC_LIBRARY

VOLTMETER Correct execution of test TEST_RDACLibrary from Libraries Tests (UI Test Menu).

10.2.2.2 TC_F_201 – TC_F_CONFIGURATION_ADJUST_VOLTAGE

VOLTMETER Correct execution of test TEST_VoltCurrCTRL from Controller Tests (UI Test Menu). Configure one of the PSUs' RDAC with a range of values that represent all its scale, while measuring the output generated from it, in order to compare the programmed voltage with the output voltage. Draft a table with the matching results.

10.2.2.3 TC_F_202 – TC_F_CONFIGURATION_DATA_LISTS

FULL Correct execution of test TEST_DataListsCTRL from Controller tests (UI Test Menu).

10.2.2.4 TC_F_203 – TC_F_CONFIGURATION_FLASH_MEM

FULL Correct execution of test TEST_FlashMemCTRL from Controller tests (UI Test Menu).

10.2.2.5 TC_F_204 – TC_F_CONFIGURATION_SYSTEM

SOFTWARE Execute each of the methods in Config Menu (UI), checking if each option does what it should do.

10.2.2.6 TC_F_300 – TC_F_COMMUNICATION_I2C&SPI_LIBRARY

FULL Correct execution of test TEST_I2CnSPILibrary from Libraries tests (UI Test Menu).

10.2.3 SWITCH ON

10.2.3.1 TC_F_400 – TC_F_SWITCH-ON_RELAY_LIBRARY

FULL Correct execution of test TEST_RelayLibrary from Libraries tests (UI Test Menu).

10.2.3.2 TC_F_401 – TC_F_SWITCH-ON_TASK

FULL Correct execution of test TEST_SwithOnCTRL from Controller tests (UI Test Menu).

10.2.3.3 TC_F_402 – TC_F_SWITCH-ON_BUTTON

FULL Switch on PSUs 0 and 1, leaving the rest siwtched off. Press the button once. Wait for disconnection. Press the button twice. Check that the only connected outputs are PSUs 0 and 1.

10.2.4 ALARM

10.2.4.1 TC_F_500 – TC_F_ALARM_TASK

FULL Correct execution of test TEST_AlarmCTRL from Controller tests (UI Test Menu).

10.2.4.2 TC_F_501 – TC_F_ALARM_PROTOCOLS

FULL Connect all the protocols for a PSU's alarms. Program an output value and connect this PSU. Configure one of the voltage thresholds to trigger the alarm (e.g. progVolt = 10V, set superior voltage limit at 8V). Check that a message is sent to the console, the PSU is disconnected and its programmed voltage is changed.

10.3 TYPE COMPLEMENTARY

Table 11 - TC Type Complementary/C

CATEGORY	SR ID	NAME	STATUS
RELIABILITY	TC_C_200	TC_C_RELIABILITY	**DESIGNED**

10.3.1 RELIABILITY

10.3.1.1 TC_C_200 – TC_C_RELIABILITY

FULL Connect and disconnect all the PSUs one by one, checking that the output voltage switches between 0 and a value close to the programmed voltage.

11 TRACEABILITY MATRIX SR – TC

This matrix links System Requirements (What does the system do?) with the Test Cases (The system does what it's supposed to do?). The verification/checking of the TCs directly ensure the fulfilment of the SRs they're linked to. Tracing the SRs to the User Requirements they belong to, it can be inferred that ***the PSUControl system does what the "client" (SSR Department) asked***, thus enclosing the "requirement elicitation" methodology (UR - SR - TC).

Table 12 - Traceability Matrix SR - TC

SR	NAME	TC	NAME
SR_I_100	SR_I_HARDWARE_GENERAL	TC_I_100	TC_I_HARDWARE_GENERAL
SR_I_101	SR_I_HARDWARE_COMMUNICATIONS	TC_I_101	TC_I_HARDWARE_COMMUNICATIONS_I2C
		TC_I_102	TC_I_HARDWARE_COMMUNICATIONS_SPI
SR_I_200	SR_I_SOFTWARE_GENERAL	TC_I_200	TC_I_SOFTWARE_GENERAL&COMMUNICA-TIONS
SR_I_201	SR_I_SOFTWARE_COMMUNICATIONS		
SR_I_202	SR_I_SOFTWARE_USER_INTERFACE	TC_I_201	TC_I_SOFTWARE_USER_INTERFACE
SR_I_300	SR_I_HEUR_STATUS_VISIBILITY	TC_I_300	TC_I_HEUR_STATUS_VISIBILITY
SR_I_301	SR_I_HEUR_UNDO_REDO	TC_I_301	TC_I_HEUR_UNDO_REDO
SR_I_302	SR_I_HEUR_STANDARD	TC_I_302	TC_I_HEUR_STANDARD
SR_I_303	SR_I_HEUR_ERROR_PREVENTION	TC_I_303	TC_I_HEUR_ERROR_PREVENTION
SR_I_304	SR_I_HEUR_INSTRUCTIONS	TC_I_201	TC_I_SOFTWARE_USER_INTERFACE
SR_I_305	SR_I_HEUR_ACCELERATORS	TC_I_305	TC_I_HEUR_ACCELERATORS
SR_I_306	SR_I_HEUR_AESTHETIC	TC_I_201	TC_I_SOFTWARE_USER_INTERFACE
SR_I_307	SR_I_HEUR_ERROR_MESSAGES	TC_I_307	TC_I_HEUR_ERROR_MESSAGES
SR_I_308	SR_I_HEUR_HELP	TC_I_308	TC_I_HEUR_HELP
SR_F_100	SR_F_MONITORING_SCAN_VOLTAGE	TC_F_100	TC_F_MONITORING_SCAN_VOLTAGE
SR_F_101	SR_F_MONITORING_SCAN_CURRENT	TC_F_101	TC_F_MONITORING_SCAN_CURRENT
SR_F_102	SR_F_MONITORING_TASK	TC_F_102	TC_F_MONITORING_TASK
SR_F_103	SR_F_MONITORING_STORE_AVERAGE	TC_F_103	TC_F_MONITORING_STORE_AVERAGE&PEAKS
SR_F_104	SR_F_MONITORING_STORE_PEAKS		

SR_F_105	SR_F_MONITORING_MUX_LIBRARY	TC_F_104	TC_F_MONITORING_MUX_LIBRARY
SR_F_106	SR_F_MONITORING_AGC_LIBRARY	TC_F_105	TC_F_MONITORING_AGC_LIBRARY
SR_F_200	SR_F_CONFIGURATION_RDAC_LIBRARY	TC_F_200	TC_F_CONFIGURATION_RDAC_LIBRARY
SR_F_201	SR_F_CONFIGURATION_ADJUST_VOLTAGE	TC_F_201	TC_F_CONFIGURATION_ADJUST_VOLTAGE
SR_F_203	SR_F_CONFIGURATION_DATA_TYPES	TC_F_202	TC_F_CONFIGURATION_DATA_LISTS
SR_F_204	SR_F_CONFIGURATION_DATA_LISTS		
SR_F_205	SR_F_CONFIGURATION_FLASH_MEM	TC_F_203	TC_F_CONFIGURATION_FLASH_MEM
SR_F_206	SR_F_CONFIGURATION_SYSTEM	TC_F_204	TC_F_CONFIGURATION_SYSTEM
SR_F_300	SR_F_COMMUNICATION_I2C&SPI_LIBRARY	TC_F_300	TC_F_COMMUNICATION_I2C&SPI_LIBRARY
SR_F_400	SR_F_SWITCH-ON_RELAY_LIBRARY	TC_F_400	TC_F_SWITCH-ON_RELAY_LIBRARY
SR_F_401	SR_F_SWITCH-ON_TASK	TC_F_401	TC_F_SWITCH-ON_TASK
SR_F_402	SR_F_SWITCH-ON_BUTTON	TC_F_402	TC_F_SWITCH-ON_BUTTON
SR_F_500	SR_F_ALARM_TASK	TC_F_500	TC_F_ALARM_TASK
SR_F_501	SR_F_ALARM_PROTOCOLS	TC_F_501	TC_F_ALARM_PROTOCOLS
SR_F_600	SR_F_WINDOWS_INTERFACE	TC_I_201	TC_I_SOFTWARE_USER_INTERFACE
SR_F_601	SR_F_WINDOWS_COMMUNICATION		
SR_C_100	SR_C_EFFICIENCY_REFRESH	TC_I_300	TC_I_HEUR_STATUS_VISIBILITY
SR_C_101	SR_C_EFFICIENCY_ALARM	TC_F_501	TC_F_ALARM_PROTOCOLS
SR_C_102	SR_C_EFFICIENCY_VOLTAGE	TC_F_201	TC_F_CONFIGURATION_ADJUST_VOLTAGE
SR_C_200	SR_C_RELIABILITY	TC_C_200	TC_C_RELIABILITY

12 PSUCONTROL PROGRESS STATUS

As shown in the Test Cases, the PSUControl system is far from finished, which is due to several delays in the hardware implementation, which at the present time isn't fully deployed. This situation didn't prevent several Test Cases to be carried out (those requiring *SOFTWARE*, *SF* or *VOLTMETER* environments).

Future work on PSUControl will rely on the completely implemented hardware to verify the Test Cases that have been labelled as "Designed" or "Not Started", as well as developing the URs and SRs prioritized as "*FUTURE*" (upgrades designed by the client and the PSUControl Developer respectively).

13 USER MANUAL

USER MANUAL

Numbers or symbols in brackets next to the menu options (seen in the UI) are the keys to be pressed.
The menus are navigated using regular characters (A-Z and 0-9) and several symbols (+ - *).
The decimal values use the dot (.) as separator (e.g. 5.45V).
The Program menu allows changes in the system, while the Displaymenu presents the system status and configuration

ACTION	SEQUENCE		
	INTERFACE MENU	PROGRAM MENU	SELECTIONS AND PROGRAMMING
Switch ON a PSU	2	1	PSUs Sel. → Switch ON
Switch OFF a PSU	2	1	PSUs Sel. → Switch OFF
Change Voltage PSU	2	2	PSUs Sel. → + or - → new Voltage
Connect/Disconnect alarm for PSUs	2	3	PSUs Sel. → Alarms → Con/Discon
Configure PSU alarms	2	4	PSUs Sel. → Alarms Sel. → ALARM MENU[1]
Configure Timing for Switch ON	2	5	PSUs Sel. → new Timing
Configure SnI alarms	2	6	SnIs Sel. → Alarms Sel. → ALARM MENU[1]
Configure Timing for Switch ON	2	7	SnIs Sel. → Alarms Sel. → Con/Discon
	DISPLAY MENU		
Read the programmed and measured volts/Currs	1	1	STATUS DISPLAY[2]
Check the alarms' status	1	1	
Check the alarms' configurations	1	2	CONFIG DISPLAY[3]

(1) ALARM MENU

Configure limit Time	1	New Time
Configure limit value	2	New Value
Configure Timing for Switch ON	3	Protocol Sel. → Activate/Deactivate
Configure PSUs to be shutdown with alarm	4	PSUs Sel.
Configure new Voltage to be set with alarm	5	New Voltage Value

Figure 24 - User Manual (I)

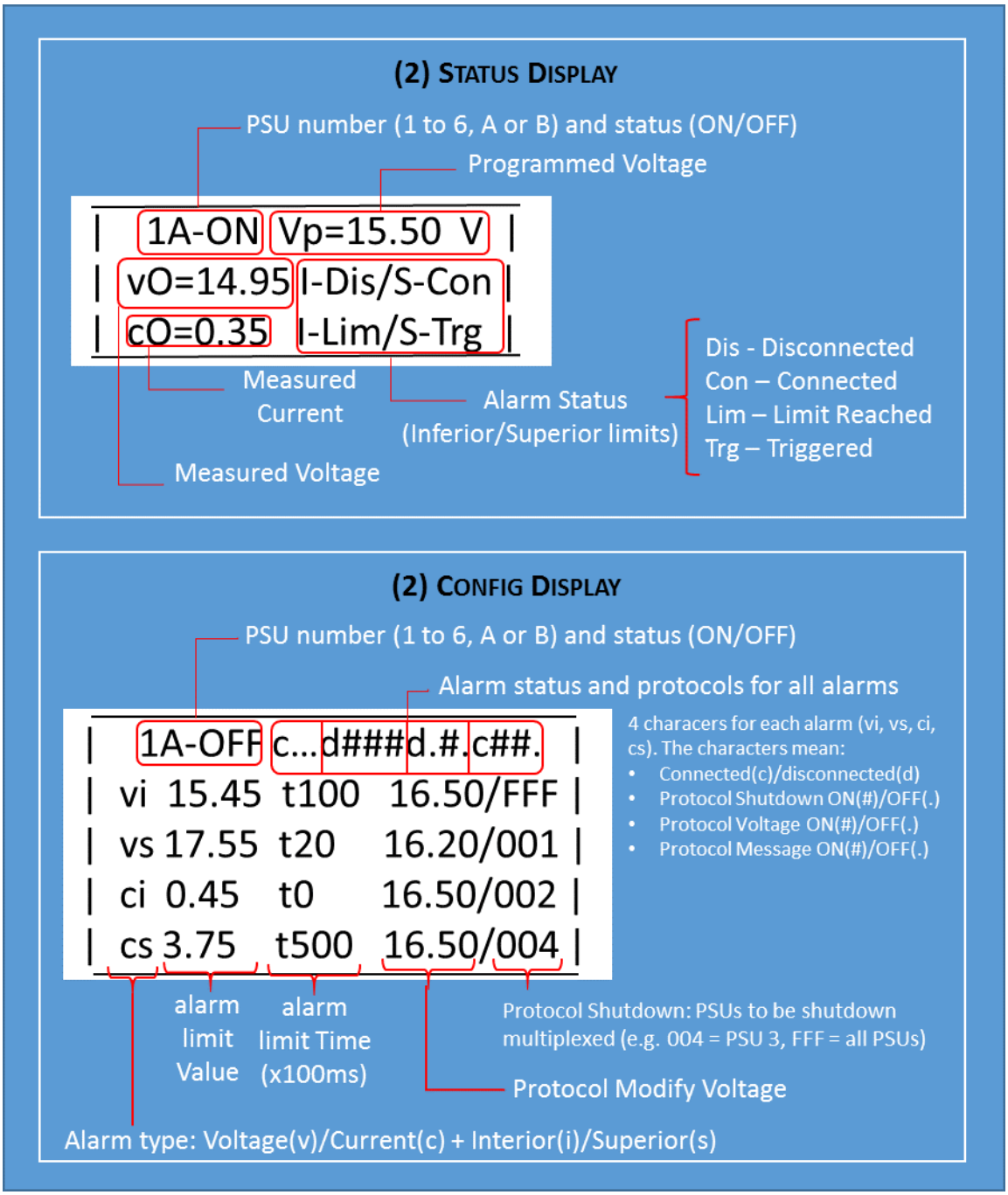


Figure 25 - User Manual (II)

13.1 SYSTEM TURN ON AND OFF

The PSUControl system is initialized the **first time** with all its PSUs disconnected and with default values set. The user may then connect any number of PSUs he desires and/or change any configuration value. The system will save its whole configuration every time a change is made. When the PSUControl system is turned off, the last configuration is saved in the flash memory module. The next time PSUControl is turned on, the previous configuration is retrieved, configuring the output

voltages and values as they were before, and connecting the PSUs that were connected the last time.

Knowing this, it is highly recommended to disconnect all the PSUs prior to turning off the system.

13.2 ADMINISTRATOR CONFIGURATION

Administrator configuration relates to the modification of default values for PSUs and SnIs, I2C addressing, A2D admitted range for monitoring, and several other parameters that are used to define the system's behavior. In order to reconfigure these otherwise constant values, the source code must be accessed. All the parameters are defined in several categories inside the file defineConstants.cpp, located in the root directory. Once the parameters have been reassigned, the PSUControl system must be dumped again to the MCF5213.

13.3 COMMON ERRORS HANDLING

Common errors found during PSUControl development are resumed here, with a probable solution.

Table 13 - Common errors handling

DESCRIPTION	SOLUTION
"ERROR 2: Create File" in the MTTTY	Check the selected port (COM5 normally used) and the USB are the same. If problem persists, disconnect and reconnect the Controller slotcard.
TRAP in UI	Exception in the code. Printf method may be causing the error. Avoid use of this command
Printf trap	Use ftos to convert float numbers to strings and use iprintf with %s instead
OSTask with PIT0 doesn't work	The PTI0 is being used as timer for an interruption, but it's required for OSDelay functions. Use PIT1.
RDAC values don't change but SPI and I2C busses are working	RDAC Control Register must be configured to allow value updates before reading. If problem persists, lower the SPI frequency. (<100KHz)
Output voltages change but the RDAC doesn't answer properly	Before receiving an answer in the SPI bus, the other RDAC (in the same SF) must be set to high impedance

14 CONCLUSIONS

The PSUControl system has been built taking into account all the objectives posed by the SSR department, who acted as the "client" of this project. These objectives have been synthesized into the User Requirements, and posteriorly transformed into system features called System Requirements. The design of the system has been carried out accordingly, with four major software packages: Libraries (communication between hardware components and the microcontroller), Controller (holding all the system's functionality), Tests (verify the previous packages) and a console-based User Interface (access to all the system functions). Finally, Test Cases have been specified to verify the fulfilment of all the requirements.

Watching a wider picture, the system's functionality is ensured by tracing all the User Requirements to several satisfactory Test Cases. Also, the use of PSUControl is greatly simplified in the User Manual, offering an easy way for someone who hasn't read this project deeply to employ its most important functionalities.

The software system has successfully passed all the software-based tests, and hardware tests have been designed for all the HW elements. These tests are left pending of verification, subordinated to the system hardware completion by an independent project from the SSR department in a near future. Several upgrades have been proposed, both by the client and by the PSUControl developer, which may be implemented in a further future.

With the supply system upgraded its manufacturing cost is lowered, the required space is reduced and the configuration of the power supply units is greatly simplified, bringing the under-the-clothes object detection Radar one step closer to its commercialization.

15 REFERENCES & ACRONYMS

[1] A. Badolato, «Diseño y fabricación de los sistemas de control y sincronismo digitales en radares de imagen en bandas milimétricas,» 2013.

[2] F. Semiconductor, «MCF5213 ColdFire Integrated Microcontroller Reference Manual,» March 2013.

[3] NetBurner, «Runtime and uCOS Libraries,» May 2012.

[4] RS-Amidata, «Datasheets for AD5292, SC18IS602B, PCA8574/74A and ADG508».

[5] IEEE , IEEE Std 830-1998: Recommended Practice for Software Requirements Specifications, 1998.

[6] D. Clegg and R. Barker, Fast-Track: A RAD Approach (Case Method), 1994.

[7] «C Programming Language, Standard ISO/IEC 9899:201x,» *Committee Draft,* April 12, 2011.

[8] E. Cerrato and R. García, «Proyecto TPR-Reservas de la UC3M,» Kiwi, May 2014.

[9] J. Nielsen, Usability Engineering, 1994.

[10] P. Kruchten, «The "4+1" View Model of Software Architecture,» *IEEE Software 12,* November 1995.

[11] OMG, «Unified Modeling Language,» Version 2.5, March 2015.

[12] G. E. Krasner and S. T. Pope, «A Cookbook for Using View-Controller User the Model Interface Paradigm in Smalltalk-80,» *Journal of Object-Oriented Programming,* September 1988.

Table 14 - Acronyms

Abbreviation	Stands for
A2D or ADC	Analog to digital Module in the Coldfire 5213
AC	alarmCounter (variable)
AC	Alternate current (voltage)
AGC	Automatic Gain Control
ALR	alarmLimitReached
ALT	alarmLimitTimes
AS	alarmStatus
AW	alarmWatch
BNC	Bayonet Neill–Concelman connector
COMx	serial COMmunication with number x
DC	Direct Current
ETSIT	Escuela Técnica Superior de Ingenieros de Telecomunicacion
GPIO	General Purpose Input Output pin
HW	Hardware
I2CtoSPIBridge	I2C bus to SPI bus Converter. Model
MCF5213	Microcontroller Coldfire model 5213
MoSCoW	Must, Should, Could, Won't
ms	milliseconds
MTTTY	Multi-Threaded Text Terminal using a Serial Communication
MUX	Multiplexer
O.A.	Operational Amplifier
PCB	Printed Circuit Board
PSU	Power Supply Unit
RDAC	Rheostat Digital to Analog Converter. Model AD5292 from Amidata
SF	Slotcard Functional
SnI	Supply and Inernal voltages
SR	System Requirement
SSR	departamento de Señales, Sistemas y Radiocomunicaciones
SW	Software
TC	Test Case
TFG	Trabajo Fin de Grado
UI	User Interface
UML	Unified modeling language
UPM	Universidad Politécnica de Madrid
UR	User Requirement