

# WAGASCI DAQ SYSTEM

## USER GUIDE

Author: Pintaudi Giorgio

Physics Department, Yokohama National University  
240-8501 Yokohama-shi Hodogaya-ku Tokiwadai 79-5  
Minamino Laboratory  
Email: giorgio-pintaudi-kx@ynu.jp  
Phone: (+81) 070-4122-3907

November 3, 2018

# Contents

<b>1 DAQ Hardware</b>	<b>3</b>
1.1 Overview . . . . .	3
1.2 SPIROC2D . . . . .	4
1.2.1 References . . . . .	4
1.2.2 Short description . . . . .	4
1.2.3 ASU . . . . .	5
1.3 Interface . . . . .	6
1.3.1 How to fix the interface . . . . .	6
1.4 DIF . . . . .	9
1.5 GDCC . . . . .	9
1.5.1 How to make a Low Voltage cable . . . . .	9
1.6 CCC . . . . .	10
1.7 ZED board . . . . .	10
1.8 Low and High Voltage PS . . . . .	10
1.9 Temperature Monitors . . . . .	10
1.10 Water Level monitor . . . . .	10
<b>2 DAQ software</b>	<b>11</b>
2.1 Pyrame . . . . .	11
2.1.1 State Machine . . . . .	12
2.1.2 The Configuration Module . . . . .	12
2.1.3 The Variables Module . . . . .	13
2.1.4 The Acquisition-Chain Module . . . . .	13
2.1.5 The Run Control Module . . . . .	16
2.1.6 The Storage Module . . . . .	16
2.1.7 Run and interact with Pyrame . . . . .	17
2.1.8 Pyrame Configuration . . . . .	17
2.2 Anpan . . . . .	17
<b>3 ROOT 6</b>	<b>18</b>
3.1 Preliminaries . . . . .	18
3.2 ROOT Installation . . . . .	18
<b>4 NEUT 5.4.0</b>	<b>20</b>
4.1 List of NEUT dependencies . . . . .	20
4.2 NEUT compatibility . . . . .	20
4.2.1 CentOS 7 - TO DO . . . . .	20
4.2.2 Ubuntu 18.04 . . . . .	21
4.3 Compile NEUT and its dependencies . . . . .	21
4.3.1 Preliminaries . . . . .	21
4.3.2 Compilers . . . . .	21
4.3.3 ROOT 5 . . . . .	22
4.3.4 CERNLIB 2005 . . . . .	24
4.4 NEUT 5.4.0 . . . . .	26

<b>5 Appendix</b>	<b>27</b>
5.1 GDCC cheat-sheet . . . . .	27
5.1.1 Fast command packet format . . . . .	27
5.1.2 GDCC packet format . . . . .	28
5.1.3 GDCC Register Packet Format . . . . .	28
5.1.4 GDCC DIF Event Packet Format . . . . .	29
5.1.5 GDCC Memory Map . . . . .	29

# Chapter 1

## DAQ Hardware

In this chapter the DAQ electronics of the WAGASCI experiment is described in as much detail as possible. With this statement, I don't mean that I am going to write down again everything that there is to know about the WAGASCI electronics: if there is any source that contains some relevant piece of information, I am going to cite that reference and consider that content as covered.

### 1.1 Overview

The WAGASCI DAQ system electronics is composed of many different boards (Figure 1.1). All of them were developed at LLR (Laboratoire Leprince-Ringuet) in France. Please refer to the following articles for an introduction to every board of the system[8, 5]. Be warned that these articles and all the ones that follow through the chapter, describe the general features of the DAQ system but don't explain how to actually use it. Moreover they are somewhat redundant, so if you choose to read them all, be prepared to read the same things over and over again. I cannot blame the authors too much for this kind of "publication" spamming. If I were them, after so much effort to develop a new DAQ system (both hardware and software), I would like at least to get as many publications as possible out of it, too.

On the other hand this very documentation is meant more as a "User Guide", so, while referring to the said literature for the more general and technical remarks, I will only focus on practical usage scenarios and examples.

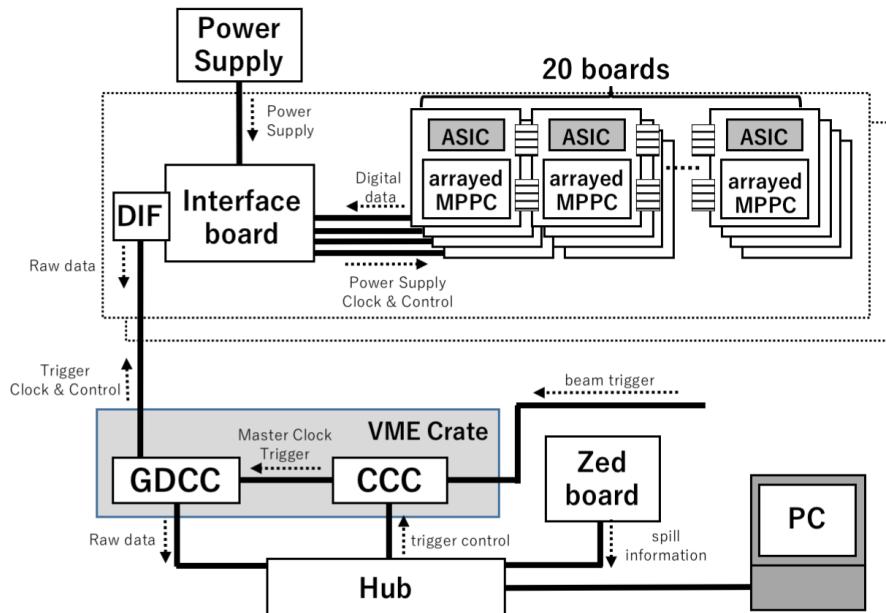


Figure 1.1: The schematics of the WAGASCI DAQ system. The very same schematics can be referred to the SideMRD where instead of 20 boards there are only 3 and instead of arrayed MPPCs there are Single MPPCs adapters

## 1.2 SPIROC2D

SPIROC is a dedicated very front-end chip developed originally for an ILC prototype hadronic Calorimeter with SiPM readout (CALICE experiment). It has been realized in  $0.35\mu\text{m}$  SiGe technology. It has been developed to match the requirements of large dynamic range, low noise, low consumption, high precision and large number of readout channels needed. The SPIROC version used for the WAGASCI DAQ is SPIROC2D.

The SPIROC ASIC that reads 36 SiPMs is an evolution of the FLC\_SiPM used in the CALICE experiment prototype. The first SPIROC prototype has been produced in June 2007 and packaged in a CQFP240 package. A second version, SPIROC2, was realized in June 2008 to accommodate a thinner TQFP208 package and fix a bug in the ADC.

SPIROC is an **auto-triggered** (it is possible to set a threshold value below which no data is acquired), **bi-gain** (there are two pre-amplifiers one with low gain for bigger signals and another with higher gain for smaller signals), **36-channel** ASIC which allows to measure on each channel the charge from one photoelectron to 2000 and the time with a 100ps accurate TDC (be warned that accuracy and precision are two distinct concepts). An analog memory array (Switched Capacitor Array) with a depth of 16 for each is used to store the time information and the charge measurement. Refer to Wikipedia for more info about the SCA (this should be more than enough if you are an experimental physicist like me).

A 12-bit Wilkinson ADC has been embedded to digitize the analog memory contents (time and charge on 2 gains). The data are then stored in a 4 kilobytes RAM. A very complex digital part has been integrated to manage all theses features and to transfer the data to the DAQ.

A small list of the most basic SPIROC properties:

- ASIC name: SPIROC (Silicon PM Integrated Read-Out Chip)
- Current available version: 2A,2B,2C,2D,2E
- Number of channel: 36
- Polarity of input signal: positive
- Detector read out: SiPM, MPPC, compliant with PM, MA-PM
- Max input signal: 2000 photoelectrons at minimum gain

### 1.2.1 References

The documentation about the SPIROC family is quite vast (when compared with the GDCC, CCC and DIF boards) but randomly dispersed through the net. Here I am providing a compilation of all the available literature.

- General articles about pre-amplifiers and amplifiers used for Physics measurements [12, 2, 14, 1] and of course everything about signal processing that you can find in the Knoll book [13].
- Articles and slide shows about the SPIROC characterization [4, 3, 6, 15, 18]
- SPIROC manuals and pin-out [10, 9, 17]

### 1.2.2 Short description

Please read this section only after having read at least some of the references above otherwise it probably won't make much sense.

Each channel of SPIROC2 is made of:

- An 8-bit input DAC with a very low power of  $1\mu\text{W}/\text{channel}$  as it is not power pulsed. The DAC also has the particularity of being powered with 5V whereas the rest of the chip is powered with 3.5V. Think of this DAQ as a way to fine tune the High Voltage supplied to the MPPCs in a range from -4V to +4V. TO-CHECK the range. This tuning directly reflects on the gain of that particular channel. It is possible to control this value by tweaking the TO-DO
- A high gain and a low gain pre-amp in parallel on each input allow handling the large dynamic range. A gain adjustment over 6 bits common for the 64 channels has been integrated in SPIROC2. TO-CHECK it is not clear!
- The charge is measured on both gains by a "slow" shaper (an amplifier with pulse duration of 50–150ns) followed by an analogue memory (SCA) with a depth of 16 capacitors.

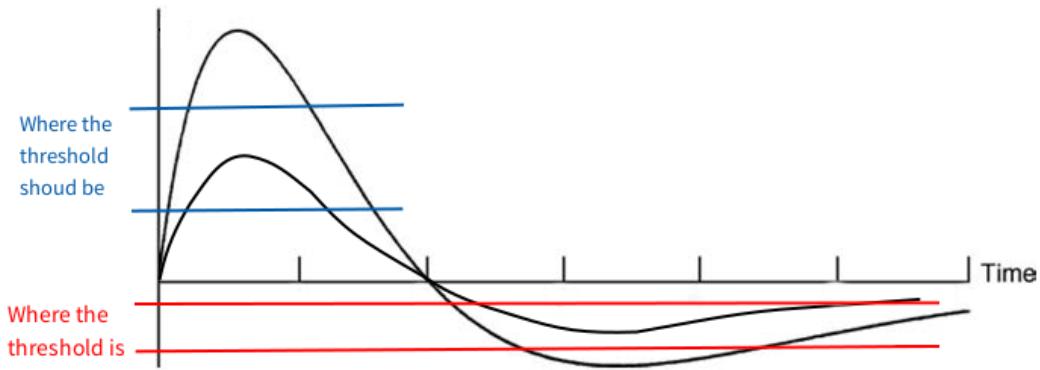


Figure 1.2: The threshold should be applied on the upper part of the signal and not on the lower. This figure shows two signals over the respective thresholds. The blue lines show where the threshold should be: in this case only signal ABOVE the threshold value trigger acquisition. The red lines show where the threshold actually is: in this case only signals BELOW the threshold value trigger acquisition.

- The auto-trigger is taken on the high gain path with a high-gain fast shaper followed by a low offset discriminator. In other words the input signal is first processed by the high-gain pre-amp and then compared with a given threshold (that can be adjusted). If the signal is “over” the threshold the acquisition is triggered, otherwise the signal is ignored. By low offset I mean that (due to a hardware error) the threshold is not set on the main part of the pulse but on the lower part of the pulse as one can see in figure 1.2. This erroneous behavior has been fixed in the SPIROC2E version but it has been shown that it doesn’t affect the measure so much as to require a replacement of all the chips. The discriminator output is used to generate the hold-and-track on the 36 channels. The threshold is common to the 36 channels, given by a 10 bit DAC with a subsequent 4 bit fine tuning per channel.
- The discriminator output is also used to store the value of a 300ns ramp in a dedicated analogue memory to provide time information with an accuracy of 100 ps.
- A 12 bit Wilkinson ADC is used to digitize the data at the end of the acquisition period.

The digital part is complex as it must handle the SCA write and read pointers, the ADC conversion, the data storage in a RAM and the readout process.

The chip has been extensively tested by many groups. The first series of tests has been mostly devoted to characterizing the analog performance, which meets the design specifications.

### 1.2.3 ASU

The Active Sensor Unit (ASU) board is just the name a PCB board containing the SPIROC2D chip. It is basically an adapter to connect the chip to the MPPCs and to the rest of the DAQ system. The ASUs can be daisy-chained together until a maximum of 40 units (4 rows of 5 ASUs) for every DIF as can be seen in Figure 1.3.

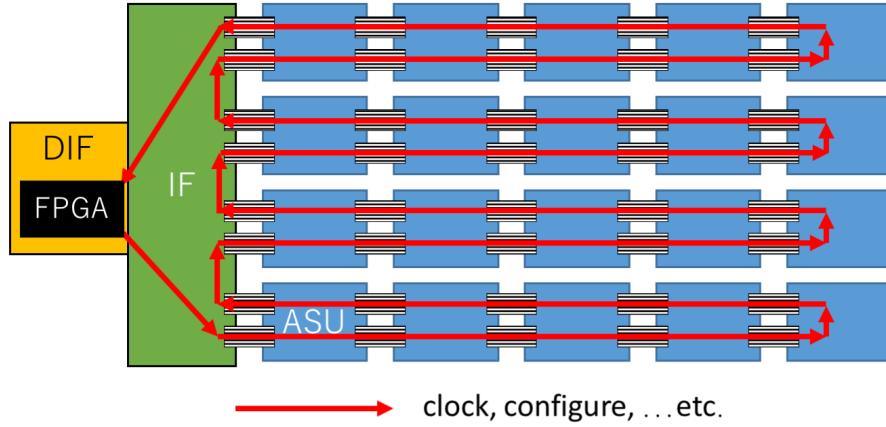


Figure 1.3: Schematic of ASU daisy chain

In Figure TO-DO it is shown a regular ASU board that IS NOT the last one of a row. TO-ADD-FIGURE The jumpers of the last ASU of every row must be set as shown if Figure TO-DO to be able to revert the signal back to the interfaces. TO-ADD-FIGURE

## 1.3 Interface

This board has no important function by itself. It is just a sort of adapter to connect all the ASUs to the DIF, to route the High Voltage to the MPPCs (through the SPIROC2C chip) and to route the Low Voltage to the SPIROC2D chip itself and to the DIF. Despite being the most trivial board of the system it is the component that gave more problems in the past.

The connectors are quite fragile and I counted at least 5 boards broken when disconnecting some cables (including one by myself). In particular take extra care when connecting-disconnecting the DIF and the Low Voltage.

Moreover, as can be read in Tamura Riku's thesis (after correcting typos):

*[...] in order to check the correct operation of the full setup, the daisy chain configuration is firstly tested. The test is done by increasing the number of daisy-chained ASU boards one by one. Up to about 10 boards, the daisy chain is correctly configured but at around 10 boards the configuration starts to fail. After several tests with different configurations, it appeared that this is due to the attenuation and reflection of the bunch crossing clock (BCID) when it travels through the chain: the signals, including the bunch crossing clock, are serially transported through the daisy chain so the length that they need to travel depends on the number of connected ASU boards. The total capacitance of the daisy chain depends on the number of connected ASU boards, too. This creates a mismatch of impedance between the endpoints and some DAQ signals are badly affected. In practice most of the DAQ signals are not so affected but it seems that the bunch crossing clock is strongly affected. The whole DAQ acquisition phase is synchronized to the bunch crossing clock so this is a very critical issue.*

*Fortunately, this problem can be fixed by “patching” the bunch crossing clock line. To prevent the attenuation of bunch crossing clock and match the impedance a 4ch buffer, CDCLVC1104[11], is applied to the bunch crossing clock line as shown in Figure TO-DO. This buffer is a highly performing and fast responding one. The delay it adds to the BCID is of 0.8-2ns, which is less than 1% of the period of bunch crossing clock, so the effect on the timing measurement due to the BCID is negligible. This patch is tested to work fine and the daisy chain is configured correctly even with the full setup (20 ASUs). [...]*

Long story short, we have to patch every interface with that chip if we want to daisy-chain more than 10 ASUs. Just to be on the safe side, all the interface boards, even if connected to less than 10 ASUs, were fixed with the following procedure.

### 1.3.1 How to fix the interface

Here I will show how to concretely fix the interface. I came to know about this procedure by reading two pdf files that were sent to YNU from I don't know where. I must admit that until now they hold the record for being the most unintelligible piece of paper that I have ever read. No matter how much I strove, I think that I could never write in such a cumbersome manner even if I want to. Anyway ...

1. First solder the CDCLVC1104 chip on the base and glue it on the board (any empty space on the interface is good). You can use a different support (the white piece of plastic in the pictures) and base (the small PCB with 10 holes on each side) if you want.

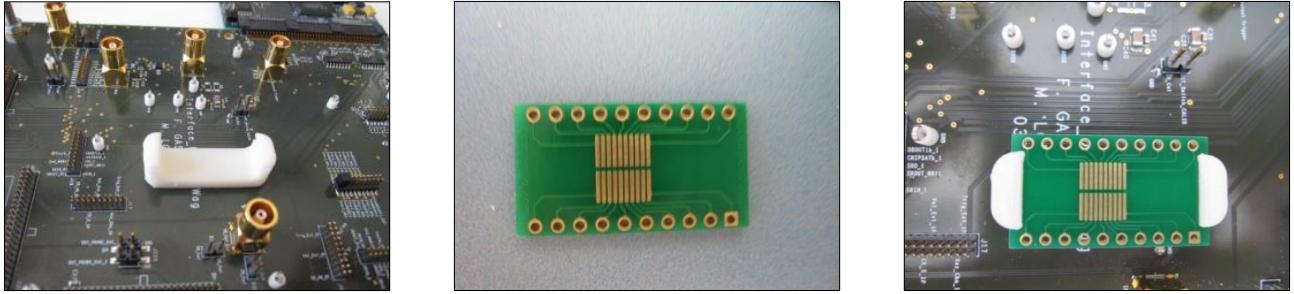


Figure 1.4: PCB Base (in the middle) glued to the interface board using a plastic support (on the left). The buffer chip must be soldered on that base (not shown in the pictures, yet). You can solder the buffer chip in any position on the base as long as it is soldered properly.

2. Then take a 50 pins flat cable (that you are then going to connect to the J2 connector). Cut in the middle the wires number 13, 49 and 50 (SR\_CK\_BUF, TRIG\_EXT\_N and TRIG\_EXT\_P respectively). Refer to Figure 1.5 for the flat cable pin-out. Strip the “ASU end” of these wires. By ASU end I mean the end that is not to be connected to the interface but to the ASU. If needed do the same for the other flat cables coming out the connectors J6, J8 and J10.

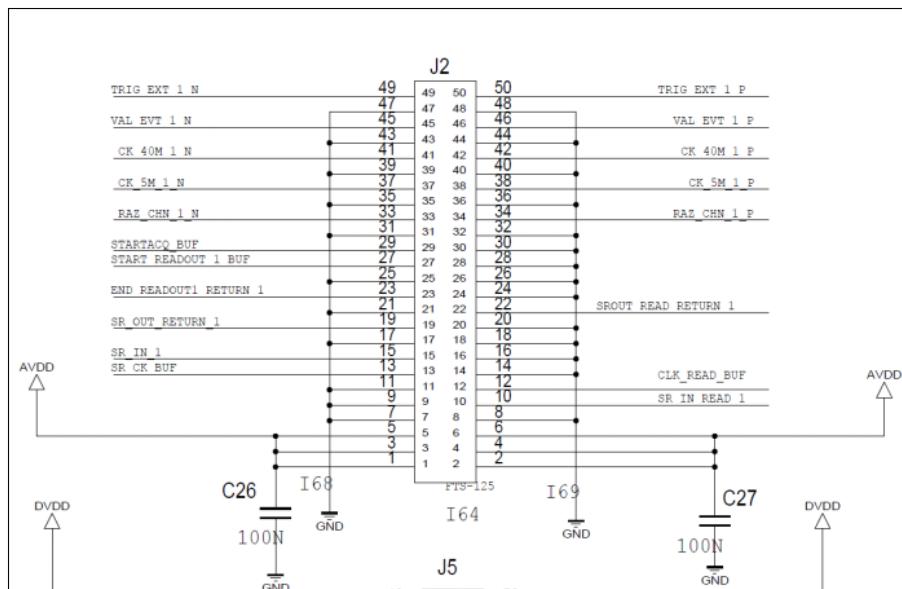


Figure 1.5: J2 connector pin-out

3. Desolder the M9 chip

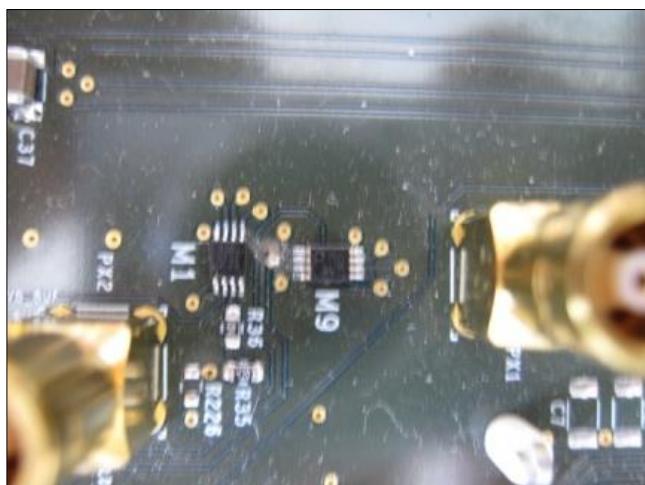


Figure 1.6: CDCLVC1104 chip pin-out

4. Referring to Figures 1.8 and 1.7, connect with some wires the pins in this way:

color	CDCLVC1104 pin	Interface pin
brown	1 CLKIN	SR_clk
red	6 VDD	(refer to picture)
black	4 ground	Interface ground

Table 1.1: Fast command packet format

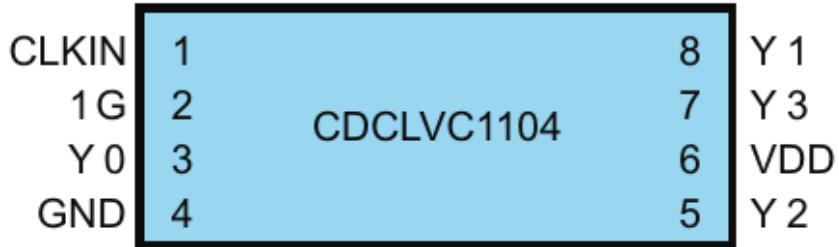


Figure 1.7: CDCLVC1104 chip pin-out

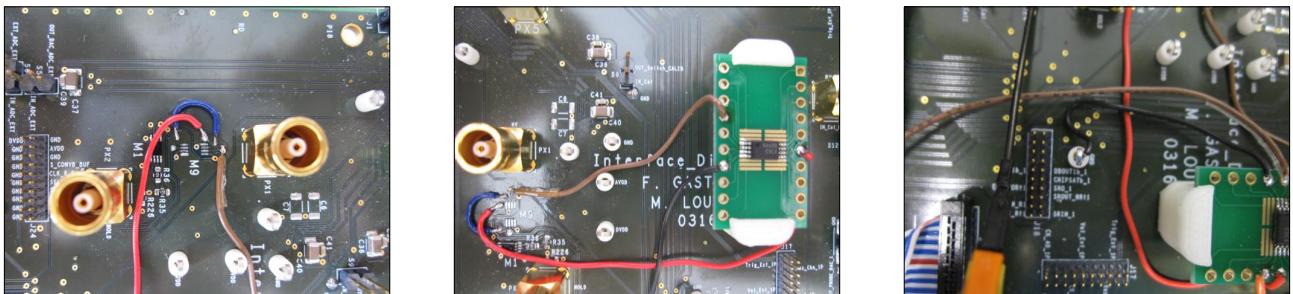
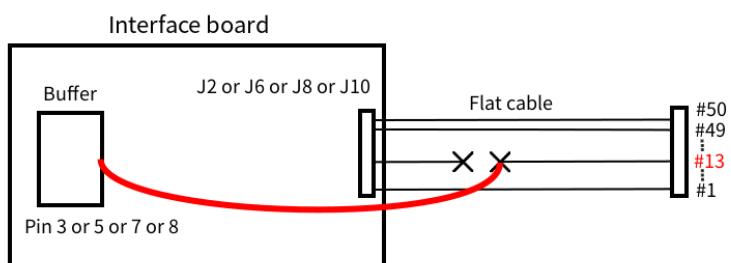


Figure 1.8: Interface connections

I am sorry but in the document that I was given there is no schematics regarding the holes around the M9 chip, so we had to solder referring only to the attached pictures.

5. (Optional but recommended) Solder a capacitor of 100nf to decouple power and ground between the pins 6 VDD and 4 GRD of the buffer. It is not shown in the pictures.
6. Now connect the interface holes around the M9 chip as shown in Figures 1.8 (blue cable). In the document I was given those are called pin number 1 and 7. Anyway, as I said, without the schematics those numbers are meaningless. Sometimes I wonder if Physicists are really so smart as they think to be.
7. Pins 3,5,7,8 of the buffer chip represent the output of the buffer. Connect each pin to the wire number 13 of the flat cables coming off J2, J6, J8 and J10. The order is not relevant. Of course, in the case of the SideMRD, only one connection is needed (for example J2).



8. Connect wire number 50 of the flat cable (it is white in our case) to any ground pin on the interface board. You have to connect the “ASU end” of the wire to ground in a similar way as shown in Figure ?? for the case of cable 13.
9. Connect wire number 49 of the flat cable (it is blue in our case) to the DVD pin on the interface board. The DVD pin is located near the M9 chip that you just desoldered. You have to connect the “ASU end” of the wire to the DVD pin in a similar way as shown in Figure ?? for the case of cable 13.
10. The final result should look more or less like this:

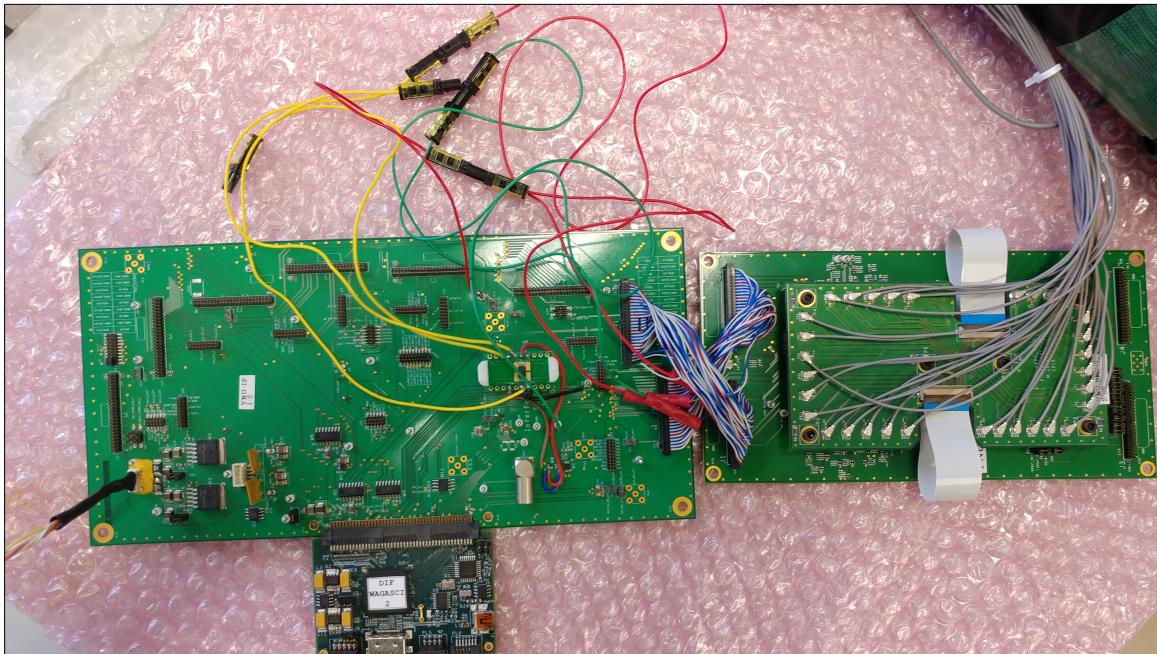


Figure 1.9: Final result. Notice that the ground wire coming from wire 50 of the flat cable is ambiguously red and not black as it should be. The green wire is coming off from wire number 49 and is connected to the DVD pin on the Interface. In total there are three yellow output wires coming off the buffer chip: of them only one is actually used.

## 1.4 DIF

## 1.5 GDCC

I have not much to add in addition to what is already in literature quoted in Section 1.1. This is the board that I know the least about, just because, fortunately, it just works and has never given any problem so far.

Please refer to the literature [7] or Appendix 5.1 if you want/need to know more about the GDCC.

The GDCC is built on standard Ethernet. Communication to and from it is done via RAW Ethernet packets. This is why it doesn't need an IP address. This way communication between the DAQ PC and the GDCC can be faster than if they traveled through the IP layer but, this way, the GDCC must necessarily be located on the same physical LAN network as the DAQ PC.

### 1.5.1 How to make a Low Voltage cable

For bench-testing purposes you may need to make your own Low Voltage cable to power the Interface and all the other boards connected to it.

The Low Voltage cable must be connected to the interface using a 5 pins connector to a vertical wire-to-board socket that looks like this:



Figure 1.10: Vertical wire-to-board 5-pins socket on the Interface board for the Low Voltage connection

To make the cable just buy a male connector (TO-DO insert link) and connect the pins following the pin-out of Figure 1.11.

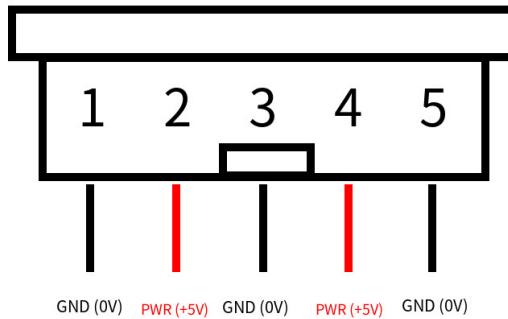


Figure 1.11: Vertical wire-to-board 5-pins socket on the Interface board for the Low Voltage connection

The ground wires can be grouped together in a single wire. The 5V wires can be grouped together in a single wire, too. The resulting 2 wires end of the cable can be terminated as you like and then connected to a 5V power supply. The power supply should be able to generate at least TO-DO Amperes of current.

## 1.6 CCC

## 1.7 ZED board

## 1.8 Low and High Voltage PS

## 1.9 Temperature Monitors

## 1.10 Water Level monitor

# Chapter 2

## DAQ software

The main software for data acquisition of the WAGASCI experiment is called Anpan (Acquisition Networked Program for Accelerated Neutrinos). It is based on the Calicoes software that was used for the Silicium Tungsten Electromagnetic Calorimeter (SiW-ECAL) prototype of the future International Linear Collider, developed by Frédéric Magniette and Miguel Rubio-Roy. Both Anpan and Calicoes are based on the Pyrame framework of which they two similar implementations.

Until Summer 2018, we were using the Calicoes software without almost any modification as the data acquisition software for the WAGASCI prototype (i.e. Proton Module + WAGASCI water-in detector + Ingrid module).

Between the end of the WAGASCI prototype commissioning run in Summer 2018 and the beginning of the WAGASCI Experiment Physics run in April 2019, I started my Ph.D. at YNU and I decided to further develop and improve the WAGASCI DAQ system. So I took the decision (quite one-sidedly I have to admit) to change the name of the acquisition software and release it under the GPLv3 license.

It is not clear which license is Calicoes released according to. Anyway Pyrame (which is written by the same developers) is released under GPLv3 so I assumed that Calicoes is released under the same license. This entitles me (Pintaudi Giorgio) to release again a modified version of the software under different name but giving due credit to the original authors (Frédéric Magniette and Miguel Rubio-Roy).

I have chosen that name because wagashi (after which the WAGASCI experiment is named) in Japanese means “traditional sweets” and anpan (a soft bun filled with red soy beans paste) is a kind of Japanese sweet snack (even if not properly an strictly traditional sweet). Anyway, I love anpan and I eat one of them almost everyday so we can say that the development of this software was fueled mainly by anpan. From that the name choice.

This chapter of the documentation isn’t a thorough guide of all the code that makes up anpan because much of that is already explained in the [Pyrame documentation website](#) and [Calicoes documentation website](#). Anyways those websites are mainly aimed at developers who are well-versed in scientific programming and are not afraid of reading through much of the source code. Instead, this guide is meant to be a user manual for the physicist who just wants to use the software (and maybe only write some simple scripts).

One must always keep in mind that most of the times scientific software is not written by a big team of programmers with a steady job and a steady income, but by a few researchers or worse grad students with little (or no) income whatsoever, unsteady future prospects and a very limited budget. Because of the lack of money and man-power, usually the first thing that get cut or reduced is documentation (because it is not strictly necessary for the software to run). This, as long as the original developers are also the users of the software, has no tangible consequences. But as new physicists start to use the software an evil-loop onsets where software is used just as-is and works-just-because-it-works. Luckily the Calicoes and Pyrame software is adequately documented and with this guide I would like to close the gap between users and software even further.

### 2.1 Pyrame

I won’t go into detail describing the Pyrame framework since much of what I have to say is already covered in the [Pyrame documentation website](#) or [16]. So before continuing to the following I would suggest the reader to take a quick look there and get a rough idea of what is happening inside of Pyrame. This section is merely intended as an appendix and a review of the content shown there. The only real difference with the online documentation is that, instead of trying to explain how the software works, I will take the point of view of the user and I will only focus on how to use Pyrame to get some work done.

First, what the user need to understand is that Pyrame is a distributed software. This means that each part or function of Pyrame can be located on a different machine. Each of those parts are called “modules” and each module communicates with the others through the exchange of TCP/IP packets. If you don’t know what the TCP/IP protocol is you can consult the Wikipedia pages: [TCP](#) and [IP](#), that should be enough.

Basically each machine able to get an IP address can open a network port through which a certain application (or in the case of Pyrame a certain module) can communicate and interact with the modules “outside”. For modules residing on the same machine the `localhost=127.0.0.1` IP address must be used. The purpose of the localhost IP address is simply to send an outgoing packet back in. So for example if we want to reach the Configuration Module (CMOD) that is running on the port `CMOD_PORT=9002` on the local machine we need the only to use the couple `localhost:CMOD_PORT` or explicitly `127.0.0.1:9002`. Or if we want to communicate with the Variables Module (VARMOD) that is running on the port `VARMOD_PORT=9001` on another machine in the same sub-net with IP `192.168.10.100` we would address it as `192.168.10.100:9001` and so on. The list of all the default ports is located in `/opt/pyrame/ports.txt`

There are 7 fundamental modules that must be always running during data acquisition for Pyrame to function correctly:

- The Variables module: `VARMOD_PORT=9001`
- The Configuration module: `CMOD_PORT=9002`
- The Mount Datadir Module: `MOUNTD_PORT=9005`
- The Acquisition-Chain Module: `ACQ_PORT=9010`
- The Run Control Module: `RC_PORT=9014`
- The Storage Module: `STORAGE_PORT=9020`

There may be other modules running depending on the specific configuration. The needed modules are usually started automatically when a configuration file is loaded so the user doesn’t need to worry about that. To learn more about the configuration file go to section TO-DO.

Starting from Section 2.1.2 I will give a short description of each of these modules, but before that I would like to talk about the “State Machine”.

### 2.1.1 State Machine

In Pyrame, the detector is always thought to be in one of four states. They are:

1. UNDEFINED: The detector state has not been probed yet. This is the state of the detector before Pyrame has started or after it has ended.
2. READY: The detector itself is in the same UNDEFINED state as before but the software has been initialized and the detector configuration has been loaded into memory.
3. CONFIGURED or RECONFIGURED: The configuration has been applied to the detector. This means that the firmware of the various boards has been configured, the power supplies turned on, the voltage set up and so on. The detector is ready to take data.
4. ACQUIRING The detector is acquiring data.

The principle at the base of the Pyrame workflow is that the user must go through all the steps from 1 to 4 in order to take data. The software will take charge of managing all the synchronization issues. All the transitions between machine state can be accomplished with the aid of shell scripts called from a terminal: `load_config_file.sh`, `initialize.sh`, `configure.sh`, etc.... It is also possible to use the CouchDB web-GUI to manage the transitions.

### 2.1.2 The Configuration Module

The configuration module (called also CMOD) is in charge of many things. First, it is dedicated to configuration storage and export. Every Pyrame module can register in the cmod and store its configuration parameters. The configuration parameters and the modules themselves are organized as a tree-like structure representing the actual structure of the detector. TO-DO how modules can edit the configuration?

Usually the user just load a configuration file in the cmod using the shell script `load_config_file.sh filename.xml`. The cmod contains at any time a copy of the configuration of all the modules and it can generate xml files containing all the parameters to be used with calxml.

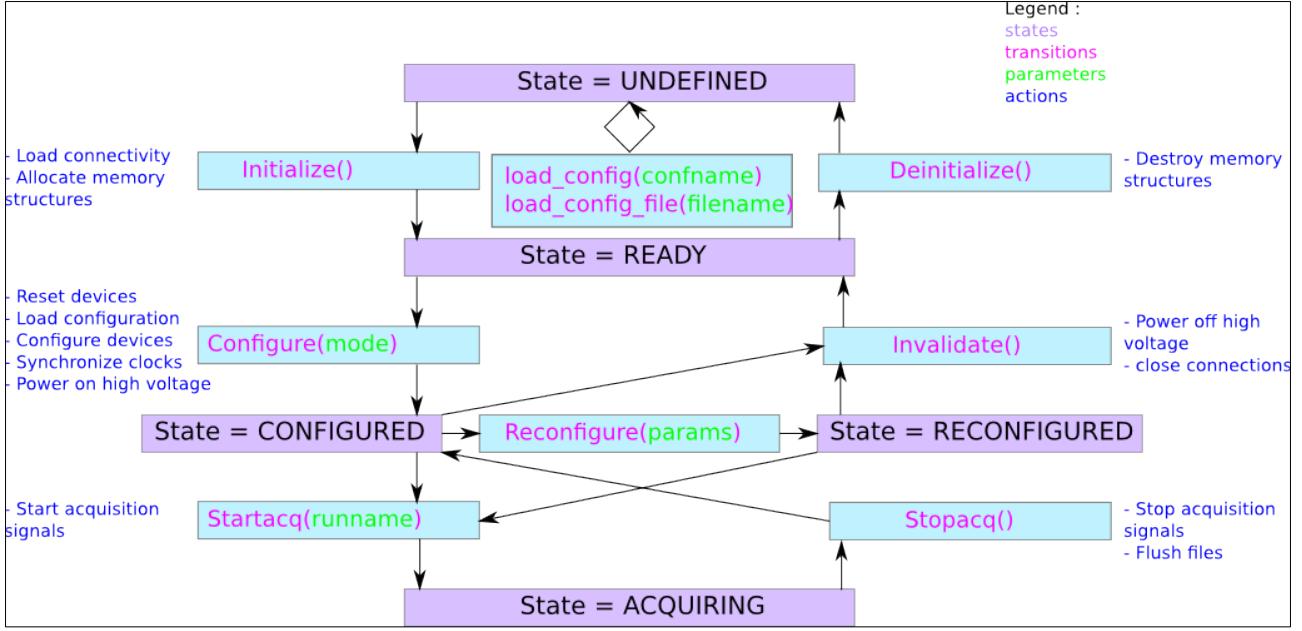


Figure 2.1: Pyrame State Machine

Another role that cmod plays is to manage the transitions between different states through the `transition_cmod` function. Again, usually this transition is managed by shell scripts or directly by the GUI so cmod is quite transparent to the user.

### 2.1.3 The Variables Module

The varmod is a centralized service that allows all the modules to share variables for collecting statistics or sharing global information. You can store a value associated with a name (a variable) and then get it back. You can also make some basic operations on the variables. For more information refer to the Pyrame documentation.

### 2.1.4 The Acquisition-Chain Module

In my opinion, understanding completely the Pyrame Acquisition-Chain is the most daunting task, if one really wants to know how the Pyrame software works internally. Since I am far from an expert in that field, I will only gather here what I have understood so far.

From the perspective of the user, reading the [Pyrame documentation](#), the comments in the configuration file and the comments in the `cmd_acqpc.py` file should be enough to get things going in almost every real-world scenario.

For the developers and more experienced users let's start by taking a look at the graph 2.2: I will only focus on the WAGASCI case. All the configuration regarding the acquisition chain is done under the `pcacq_X` field in the configuration file. If you take a pre-existing configuration file written by myself you will find many descriptive comments in that section that hopefully will guide you through a painless configuration.

## Media acquisition

The “media acquisition” phase is controlled by the acquisition plugins. What these plugins do is to “open” an acquisition media and get the data from there. The acquisition media can basically be of two types: the Ethernet port (with TCP or UDP protocols) or a raw file that was previously acquired (for testing or debugging purposes). For a more in-depth description of every plugin refer to [this page](#) and its sub-pages. After the data has been acquired, it is not ready to be analyzed yet because it contains the IP/TCP caps and it is in binary form.

You can take a look at the source code and related comments in the `anpan/cmd_acqpc/cmd_acqpc.py` file and in particular in the `init_fin_acqpc` function where every acquisition plugin is listed and explained.

## Data uncapping

In this phase, the acquired data is uncapped. This means that, for example in the TCP case, the IP/TCP headers are stripped, the data integrity checked (lost or corrupted packages) and the data content filtered (data packets

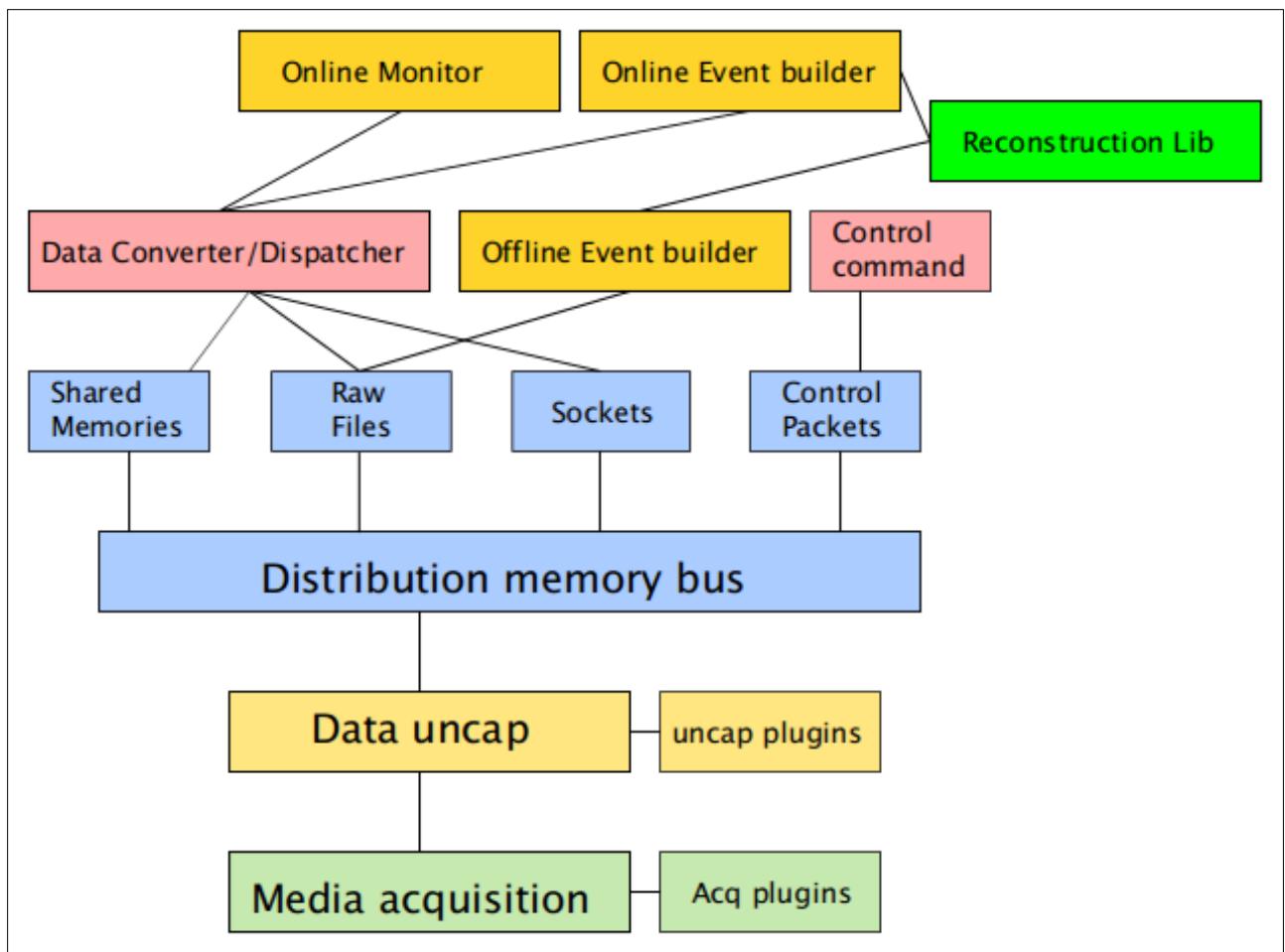


Figure 2.2: Pyrame State Machine

or control packets). Honestly I don't know much more than this about these plugins.

## Data Converter/Dispatcher

Here the decoder library is called. For example in the case of SPIROC2D the library path is `spiroc2d_decoder.so`. This library takes the uncapped raw data coming from the acquisition source (for example the GDCC) and convert it into a stream of blocks of events. These blocks are then fed to the Online Monitor and Online Event Builder programs. The decoder library needs to know the channel mapping (the geometry) of the detector to associate to every channel its position in space.

However, maybe because of some bugs or lack of time, the SPIROC2D converter for the WAGASCI prototype commissioning run was disabled. In the `cmd_dif.py` file the line needed to start the conversion was commented out and function's result was always set to positive, thus completely disabling Online Monitor capabilities of Pyrame. I am going to enable it again and debug it (TO-DO)

Moreover, in the current Pyrame implementation only the Shared Memory and Raw file distribution buses are implemented (check the `cmd_acq.c` file).

## Online Monitor

The WAGASCI online monitor code is contained in the `anpan/guis/om_wagasci` folder. It is a program external to Pyrame and WAGASCI that needs to be started by hand. It however interacts with Pyrame under the hood. It receives the Blocks and Events created by the "Data Converter/Dispatcher" face. More info about the More info can be found in section TO-DO.

In Pyrame the raw data is organized in events and blocks of events. If you want to understand in depth the WAGASCI Online Monitor it is almost mandatory that you read this article [19], otherwise the following subsections won't make any sense.

## Events

On a raw level, an event is just a set of hits. Each hit is roughly described by the time, space and charge released. Other parameters may also refer to an event. An event is described by a struct like this:

---

```
struct event {
    char **time;           // List of string containing hit times
    int time_size;         // Number of hit times
    char **space;          // List of string containing hit points
    int space_size;        // Number of hit points
    char **data;           // List of string containing various
                           // event properties.
    int data_size;         // Number of elements in data
    char used;             // ?
    struct event *next;    // Pointer to the next event
    struct block *block;   // Pointer to the block
};
```

---

Up to now the `data` list contains the following elements:

- “spill”: spill number.
- “spill\_flag”: the spill flag is set when the event happens inside of the beam spill time-frame. This means that the particle detected is most probably the result of a neutrino interaction from the neutrino beam inside or outside the detector.
- “roc”: ROC chip number.
- “rocchan”: ROC chip channel number.
- “bcid”: BCID (Bunch Crossing IDentification). Despite the misleading name, the BCID indicates just the number of rise time of the slow since the `startacq` signal. More on this on TO-DO.
- “sca”: Switched Capacitor Array number.
- “plane”: Plane number.
- “channel”: Channel number.
- “en”: Energy (charge). This field correspond to the ADC count.

- TDC time: “time”
- “hit”: The hit flag is set if the event contains a hit. The hit may be due to a particle passage inside of the detector or simply by dark or electric noise.
- “gain”: The gain flag is set when we want to measure the gain. In that case we compare the pedestal and first hit position to calculate the gain.

## Blocks

A block is a set of events bundled together according to a certain criterion (usually the spill number or bunch number). It is described by a struct like this:

---

```
struct block {
    int id;           // Block identifier
    char finalized; // ?
    int nb_events;   // Number of events in the block
    char **props;    // List of strings that can contain
                     // various block properties as the
                     // spill number, spill flag, etc...
    int props_size;  // Number of props strings
    char *bstr;      // ?
    struct evtstruct *es; // This struct is used to navigate
                         // through the various props strings
    struct event *events; // List of events
    struct block *next; // Pointer to the next block
};
```

---

Up to now the **props** list contains the following elements:

- spill number: “spill”
- spill flag: “spill\_flag”

The “spill\_flag” is “1” during the spill time frame and “0” otherwise. This flag is useful to discriminate between cosmic muons and particles coming from accelerated neutrino interactions. For further info about the WAGASCI spill timings refer to section TO-DO.

### 2.1.5 The Run Control Module

TO-DO

### 2.1.6 The Storage Module

Storage is a Pyrame module that manages data created by other modules and its inclusion on the RunDB. For further info about the RunDB module refer to the Pyrame documentation.

Storage allows to use a mount point and a path to transparently store data elsewhere in the network. It also automates common tasks such as saving the calxml configuration and a log with optional varmod variables (e.g.: stats) at the same place as data files.

As far as I know there are only two kinds of storage that the Storage Module can handle. One is the standard file on a file system and another one is a NFS share on the network.

In the case of a standard file this is the format for the mp variable:

---

```
<param name="storage_brg_mp">file:///</param>
```

---

In the case of a NFS share this is the format for the mp variable:

---

```
<param name="storage_brg_mp">nfs://server/nfs</param>
```

---

## 2.1.7 Run and interact with Pyrame

There are several ways to interact with Pyrame and its modules. When the AnpanInstaller script installs Pyrame, it makes sure that Pyrame is automatically started at every reboot. On system with `systemctl`<sup>1</sup> Pyrame is started as a system daemon and its execution is controller with the `systemctl` or `service` shell commands. To enable Pyrame at startup use

---

```
sudo systemctl enable pyrame
```

---

To disable it

---

```
sudo systemctl disable pyrame
```

---

To start a single time Pyrame

---

```
sudo systemctl start pyrame
```

---

To stop it

---

```
sudo systemctl stop pyrame
```

---

Moreover there are several ways to interact with Pyrame modules. If a module is already running the simplest way to interact with it is to open a terminal and use the command:

---

```
chkpyr2.py hostname port function parameters
```

---

where `hostname` is the IP address of the module, for example `localhost`, `port` is the port number of the module, `function` is the particular function that you want the module to perform and `parameters` are the parameters to be passed to that function. All the public functions of the Pyrame and Anpan/Calicoes modules are listed on the online documentations.

You can also interact with Pyrame modules from external python scripts ... TO-DO

## 2.1.8 Pyrame Configuration

TO-DO

## 2.2 Anpan

- WAGASCI\_PORT=12000
- ACQPC\_PORT=12001
- GDCC\_PORT=12003
- CCC\_PORT=12004
- DIF\_PORT=12005
- ASU\_PORT=12006

---

<sup>1</sup>Both Ubuntu 18.04 and CentOS 7 use `systemctl`

# Chapter 3

# ROOT 6

ROOT is a modular scientific software framework. It provides all the functionalities needed to deal with big data processing, statistical analysis, visualization and storage. It is mainly written in C++ but integrated with other languages such as Python and R<sup>1</sup>.

This chapter is very short because much of the content that should go here is already covered in section 4.3.3. The only real difference is that here we are compiling the latest version of ROOT with the latest compilers (at the time of writing: November 3, 2018).

I am assuming here that you have never installed ROOT before. That is why we are cloning the git repository. If you already have another installation of ROOT and you just want to upgrade or reinstall, please make the proper modification to these steps. I am assuming that you are capable of that.

This chapter is tailored towards Ubuntu 18.04 but it should be the same for any other Linux distribution with some minor modifications.

## 3.1 Preliminaries

First install all the packages listed in section 2. Again, some of them may not be strictly needed but cherry-picking which package is needed and which not could be very time consuming. If you have very strict space limitations and you are somewhat forced to install the least amount of software, I really feel sorry for you but don't expect me to willingly help you...

Then make you sure you are using the last version of the compilers by issuing

---

```
gcc --version
```

---

The output should be

---

```
gcc (Ubuntu 7.3.0-16ubuntu3) 7.3.0
[...]
```

---

## 3.2 ROOT Installation

Then follow the step 4 of section 4.3.3. Instead of step 5, type

---

```
mkdir -p $HOME/Code/ROOT/{sources,6-14-02,6-14-02-build}
cd $HOME/Code/ROOT
```

---

Instead of step 6, type

---

```
git clone http://github.com/root-project/root.git sources
cd sources
git checkout -b v6-14-02 v6-14-02
cd ../6-14-02-build
```

---

Instead of step 7, type

---

<sup>1</sup>Text taken from ROOT homepage

---

```
cmake -Dbuiltin_xrootd=ON  
-DCMAKE_INSTALL_PREFIX=$HOME/Code/ROOT/6-14-02 ./sources
```

---

The following steps are the very same of section 4.3.3 where in place of **5-34-00-patches** you just have to type **6-14-02**.

# Chapter 4

## NEUT 5.4.0

NEUT is a neutrino interaction simulation program library that has been developed for the studies of the atmospheric neutrino and the accelerator neutrinos. In this guide it is described how to compile NEUT on a Linux machine. **NEUT 5.4.0** is the latest NEUT version at the time of writing (November 3, 2018). The author of this guide is not the NEUT author, but it is a member of the T2K collaboration. NEUT is not an open-source software and its sources **can be accessed only by T2K collaborators** in possession of a username and password for the **T2K intranet**.

A fresh Linux installation is assumed, so, before compiling NEUT, I am showing how to install or compile all NEUT dependencies. If you already have some of the dependencies installed, you can safely skip the relative paragraphs. Regarding the style of this guide, I wanted to stay on the safe side so I have explained every step in detail. This means that the guide could appear boring and over-repetitive to a reader well-versed in Linux software compilation and programming and I preventively apologize for that.

### 4.1 List of NEUT dependencies

To compile NEUT, three main dependencies are needed:

- **CERNLIB 2005**

CERNLIB 2005 is a quite old version of **cernlib**, its latest release being dated 2012. Unsurprisingly it is quite tricky to compile. This is why I have included in the present guide a complete and detailed walk-through to the compilation of CERNLIB 2005. There are several patches that need to be applied to both CERNLIB and NEUT source codes. Some of them were written by me, some others by Hayato-san.

- **ROOT 5**

NEUT is compatible **only with ROOT 5** and not ROOT 6 yet. There are plans to upgrade NEUT so to make it compatible with ROOT 6, but I don't know if there is any progress in that direction nor if there is any ETA. This plan was announced in the last T2K meeting in May. The latest and most updated version of ROOT 5 is **ROOT 5.34.00-patches** (it is based on ROOT 5.34.39 and the latest commit on the **GitHub branch** is March 2018) and I have chosen that for my personal installation.

- **Fortran compilers (`gfortran` and `g77`)**

The old fortran compiler **g77** is not maintained anymore and, for example, it is not present in the Ubuntu repositories since the Ubuntu 8.04 Hardy Heron (my first Ubuntu, what nostalgia). Anyway, it is still possible to install it in a somewhat nonstandard way as it will be shown in the following.

### 4.2 NEUT compatibility

NEUT officially supports only CentOS 7, but this doesn't mean that it cannot be run without issues on other Linux OSes. I will introduce here 3 OSes that I have successfully compiled NEUT onto: CentOS 7, Ubuntu 17.10 and Ubuntu 18.04. I will show how to compile NEUT only on Ubuntu 18.04 64 bit because that is the most delicate case and because Ubuntu 18.04 is the latest Ubuntu LST (Long Term Support) release.

#### 4.2.1 CentOS 7 - TO DO

In May 2018 Hayato Yoshinari-san, who I think is also the main developer of **NEUT**, organized a small course about NEUT. Since the course was in Japanese and I am not fluent yet, I could understand very little. Anyway,

in that occasion Hayato-san provided us with a CentOS 7 virtual machine containing all the needed dependencies to compile and run NEUT. The virtual machine can be downloaded from [here](#). I am not sure that this link is of public dominion, but as far as it is circulated inside the T2K collaboration there shouldn't be any problem. The CentOS virtual machine can be run through Oracle [VM VirtualBox](#). The CentOS virtual machine comes without any desktop manager. But it is possible to install it if needed (I have installed GNOME). The user-name is **neut** and the password is **neut5.4.0**. Here some relevant details about it:

OS	CentOS 7
kernel	3.10.0-693.21.1.el7.x86_64
gcc/g++/gfortran	4.8.5 20150623 (Red Hat 4.8.5-16)
ROOT	5.28.00h+
CERNLIB	2005 (dated 2016/12/12)
NEUT	5.4.0

## 4.2.2 Ubuntu 18.04

With the aim of writing the present guide, I wanted to test the NEUT compilation on a fresh install. I have chosen Ubuntu 18.04 because it is the latest Ubuntu LTS version, so in the following I will assume an Ubuntu 18.04 64 bit fresh install. Every other Linux distribution should be compatible after minimal modification to some of the shell commands, particularly regarding the package manager and the package names. Notice that I haven't used the default Ubuntu 18.04 compiler (version 7.3.0) but I have installed the old 4.8 version of **gcc**, **g++** and **gfortran** as shown in subsection 4.3.2.

OS	Ubuntu 18.04
kernel	4.15.0-23-generic
gcc/g++/gfortran	(Ubuntu 4.8.5-4ubuntu8) 4.8.5
ROOT	5.34.00-patches (patched version of ROOT 5.34.39)
CERNLIB	2005 (dated 2016/12/12)
NEUT	5.4.0

## 4.3 Compile NEUT and its dependencies

### 4.3.1 Preliminaries

1. Open a terminal or a shell. Ideally you should never close this shell from now until last step. If you need to close it nothing bad will happen but you might need to set up CERNLIB and ROOT environments again and/or come back to the right folder depending at which step you want to resume compilation.
2. First, we need to install some packages from the Ubuntu repositories. Some of them may be optional but I didn't have the patience to cherry-pick them. I have included them all just to be on the safe side. Do notice that it may take some time and space (~500MB).

---

```
sudo apt-get update && sudo apt-get upgrade
sudo apt-get install build-essential git dpkg-dev cmake xutils-dev \
g++ gcc gfortran binutils libx11-dev libxpms-dev libxft-dev libxext-dev \
libssl-dev libpcre3-dev libglu1-mesa-dev libglew-dev \
libmysqlclient-dev libfftw3-dev libcfitsio-dev libgraphviz-dev \
libavahi-compat-libdnssd-dev libldap2-dev python-dev libxml2-dev \
libkrb5-dev libgsl-dev libqt4-dev libmotif-dev libmotif-common \
libblas-dev liblapack-dev csh tcsh gcc-4.8 g++-4.8 gfortran-4.8
```

---

### 4.3.2 Compilers

If you try to compile NEUT with compilers newer than 4.8, everything will still compile fine but you will get the following error on NEUT execution time:

---

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
LOCB/LOCF: address 0x7f68e28cd740 exceeds the 32 bit address space
or is not in the data segments
This may result in program crash or incorrect results
```

---

```
Therefore we will stop here
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
```

---

This error is ultimately connected to the **FFKEY** call in fortran. **FFKEY** allows something which is read in to be an integer or float at a later time in the run. This should not be allowed (and isn't) by modern standards, but is still in CERNLIB and NEUT.

To switch between compilers, create and make executable the following shell script

```
#!/bin/sh

if [ -z "$1" ]; then
    echo "usage: $0 version" 1>&2
    exit 1
fi

if [ ! -f "/usr/bin/gcc-$1" ] || [ ! -f "/usr/bin/g++-$1" ] || [ ! -f "/usr/bin/gfortran-$1" ];
then
    echo "no such version gcc/g++/gfortran installed" 1>&2
    exit 1
fi

sudo update-alternatives --set gcc "/usr/bin/gcc-$1"
sudo update-alternatives --set g++ "/usr/bin/g++-$1"
sudo update-alternatives --set gfortran "/usr/bin/gfortran-$1"
```

---

The script accepts as the only and compulsory parameter the version of the compiler to switch into. It is **4.8** for 4.8 compilers and **7** for the 7.x compilers.

Be sure to run the script and switch to the 4.8 compiler before starting the compilation of ROOT, CERNLIB and NEUT.

### 4.3.3 ROOT 5

As already pointed out in the introduction, at the time of writing, **NEUT 5.4.0 is only compatible with ROOT 5**. Please refer to section 4.1 for further details about this “issue”. Here I will explain how to compile ROOT 5.34.00-patches starting from the source code since it is the safest way to insure that everything is functioning correctly: if there is any incompatibility it is usually reported at compilation time, while, if you install ROOT through the pre-compiled binaries, any incompatibility will be uncovered only at run time, when the errors can be more obscure and misleading to interpret.

I recommend the use of the “three folders” procedure to install ROOT. One folder is for the source code, one for the compiling/building and one for the final installation. The sources folder is managed entirely using git and it contains a copy of the ROOT git repository. You can checkout the branch that you need to compile at any time. Another folder contains the temporary building files (such as the object files). It is effectively used only during compilation and can be removed afterwards. It is quite heavy (about 3GB) so I usually delete it as soon as I have installed ROOT. The last folder is the installation folder where a certain version of ROOT is installed. When running ROOT, in fact only this folder is needed.

I am showing now how to install ROOT. The following instructions are freely taken from the following pages:

- [Building root](#)
- [ROOT v5-34-00-patch release-notes](#)
- [Build prerequisites](#)

3. Open a terminal if not already opened.
4. Then install some fonts required by ROOT

---

```
sudo apt install xfstt xfsprogs t1-xfree86-nonfree ttf-xfree86-nonfree \
ttf-xfree86-nonfree-syriac xfonts-75dpi xfonts-100dpi
```

---

You can optionally install the following packets to have a more thorough ROOT installation. They are by no means mandatory. I honestly have no idea what many of these packages do. I just hate dealing with missing dependencies when I run software.

---

```
sudo apt install libgif-dev libtiff-dev libjpeg-dev liblz4-dev \
liblzma-dev libgl2ps-dev libpostgresql-ocaml-dev libsdlite3-dev \
libpythia8-dev davix-dev srm-ifce-dev libtbb-dev python-numpy
```

---

5. Create two directories that will contain the ROOT source code and the compiled code. You can change the paths as you wish, but then you have to remember to change all the references and links in the following accordingly. You may also need to modify the patches since they include these paths.

---

```
mkdir -p $HOME/Code/ROOT/{sources,5-34-00-patches,5-34-00-patches-build}
cd $HOME/Code/ROOT
```

---

6. Clone the ROOT source code into the sources directory using git and then:

---

```
git clone http://github.com/root-project/root.git sources
cd sources
git checkout -b v5-34-00-patches origin/v5-34-00-patches
cd ../5-34-00-patches-build
```

---

7. Execute the cmake command on the shell.

---

```
cmake -Dbuiltin_xrootd=ON -DCMAKE_INSTALL_PREFIX=$HOME/Code/ROOT/5-34-00-patches .../sources
```

---

CMake will detect your development environment, perform a series of test and generate the files required for building ROOT.

8. After CMake has finished running, start the build from the build directory:

---

```
cmake --build . --target install
```

---

On UNIX systems (with make or ninja) you can speedup the build with

---

```
cmake --build . --target install -- -jN
```

---

where N is the number of available cores.

9. Now you can safely remove the build directory if you need that space.

---

```
cd
rm -Rf $HOME/Code/ROOT/5-34-00-patches-build
```

---

10. Every time that you run ROOT, you need to setup the environment. There are at least three ways to do that, more ore less automated.

- One is to run the command

---

```
source $HOME/Code/ROOT/5-34-00-patches/bin/thisroot.sh
```

---

Every time it is needed.

- Another way is to define an alias in the `.bash_aliases` file or in the `.bashrc` file. For instance I have set

---

```
alias root5='source $HOME/Code/ROOT/5-34-00-patches/bin/thisroot.sh'
alias root6='source $HOME/Code/ROOT/6-12-06/bin/thisroot.sh'
```

---

To quickly switch between different ROOT versions.

- Alternatively one can set up the correct root environment at boot by inserting the following lines in the `.profile` file

---

```
# set the needed environment variables, PATH and LD_LIBRARY_PATH for root
if [ -f "${HOME}/Code/ROOT/5-34-00-patches/bin/thisroot.sh" ] ; then
```

---

```
source ${HOME}/Code/ROOT/5-34-00-patches/bin/thisroot.sh
fi
```

---

Then every shell will have that version of root as default.

11. Before using root, remember to reboot (to reload the fonts). Then you can ROOT with the command **root**.

#### 4.3.4 CERNLIB 2005

The following procedure was freely taken from the three reference web pages:

- NEUT installation
  - CERNLIB installation
  - g77 installation
12. Create a new folder for CERNLIB and move to it. In the following I am assuming that you won't leave that folder until the CERNLIB compilation is complete.

```
mkdir $HOME/Code/CERNLIB
cd $HOME/Code/CERNLIB
```

---

13. CERNLIB uses a different naming for make. I suppose this is due to small differences in naming conventions between UNIX and Linux (g-nu).

```
sudo ln -s /usr/bin/make /usr/bin/gmake
```

---

14. The following commands add the hardy repositories to the **sources.list** file and install **g77**. This step is needed to download and install the fortran **g77** compiler. It is advisable to comment or remove these lines after the **g77** installation.

```
sudo tee -a /etc/apt/sources.list << END
# Old hardy repository needed to install g77
# It is recommended to comment or remove following lines after g77 installation

deb [trusted=yes] http://old-releases.ubuntu.com/ubuntu/ hardy universe
deb-src [trusted=yes] http://old-releases.ubuntu.com/ubuntu/ hardy universe
deb [trusted=yes] http://old-releases.ubuntu.com/ubuntu/ hardy-updates universe
deb-src [trusted=yes] http://old-releases.ubuntu.com/ubuntu/ hardy-updates universe
END
sudo apt-get update && sudo apt-get install g77
```

---

15. Now it is time to solve a couple of bugs of the **g77** compiler. This is quite an old and now largely obsolete compiler. For this reason many adjustments are needed to get it working. You may need to modify the code below if your system libraries path or your gcc compiler version are different. For more information and troubleshooting please consider visiting [this page](#).

The following command will create a symbolic link of a library that **g77** would not able to find otherwise.

```
sudo ln -s /usr/lib/gcc/x86_64-linux-gnu/7/libgcc_s.so /usr/lib/x86_64-linux-gnu/
```

---

16. Then let us set up the **LIBRARY\_PATH** environment variable. This must be done every time you want to use the **g77** compiler.

```
export LIBRARY_PATH="/usr/lib/x86_64-linux-gnu:$LIBRARY_PATH"
```

---

If you want to avoid issuing the previous command every time, you can do the following:

---

```
tee -a $HOME/.profile << END

# set LIBRARY_PATH so it includes x86_64-linux-gnu if it exists
if [ -d "/usr/lib/x86_64-linux-gnu" ] ; then
LIBRARY_PATH="/usr/lib/x86_64-linux-gnu:$LIBRARY_PATH"
export LIBRARY_PATH
fi
END
```

---

17. Download all the CERNLIB 2005 source code archives:

---

```
wget http://www-zeuthen.desy.de/linear_collider/cernlib/new/cernlib-2005-all-new.tgz
```

---

---

```
wget http://www-zeuthen.desy.de/linear_collider/cernlib/new/cernlib.2005.corr.2014.04.17.tgz
```

---

---

```
wget http://www-zeuthen.desy.de/linear_collider/cernlib/new/cernlib.2005.install.2014.04.17.tgz
```

---

18. Extract the downloaded files:

---

```
tar -zxf cernlib-2005-all-new.tgz
cp cernlib.2005.corr.2014.04.17.tgz cernlib.2005.corr.tgz
tar -zxf cernlib.2005.install.2014.04.17.tgz
```

---

19. Download my patched version of the old **patchy 4** source code and substitute it with the non patched one. I am not sure if the **patchy** suite is really necessary for NEUT but it is better to have and not need it than need it and not have.

---

```
rm patchy4.tar.gz
wget https://www.dropbox.com/s/n7n1wiivn1noets/patchy4.tar.gz
```

---

20. Download and check the patch:

---

```
wget https://www.dropbox.com/s/loa53mkvotzsvj/jojo-CERNLIB.patch
cp Install_cernlib Install_cernlib.backup
patch -p0 --dry-run < jojo-CERNLIB.patch
```

---

21. If there isn't any error, patch the installation script

---

```
patch -p0 < jojo-CERNLIB.patch
```

---

22. Now you can compile CERNLIB.

---

```
./Install_cernlib
```

---

23. Check that the compilation process went well by looking for errors in the log files in **2005/build/log**

---

```
grep ./2005/build/log -R -i -e "error [0-9]" | grep -v ZDROP | grep -v ZBOOKN
```

---

24. Finally set up the CERNLIB environment for the next section.

---

```
source cernlib_env
```

---

## 4.4 NEUT 5.4.0

25. If you are continuing from the same terminal as the previous section, you don't need this step. Otherwise open the terminal and issue the following commands to setup the NEUT compilation environment:

---

```
source $HOME/Code/ROOT/5-34-00-patches/bin/thisroot.sh
source $HOME/Code/CERNLIB/cernlib_env
```

---

26. Create the NEUT folder and move to it.

---

```
mkdir $HOME/Code/NEUT
cd $HOME/Code/NEUT
export NEUT_ROOT=$HOME/Code/NEUT
```

---

27. Download the NEUT source code from [this link](#). To download the file it is necessary to insert one's own T2K credentials. Then move the downloaded archive in the NEUT folder. During the download don't close the terminal. Go back to the terminal and issue:

---

```
cd $HOME/Code/NEUT
mv neut_5.4.0_20180120.tar.gz neut_5.4.0_20180120.tar
tar -xvf neut_5.4.0_20180120.tar
mv neut_5.4.0/src ./src
rmdir neut_5.4.0
```

---

28. Download and check the patches

---

```
cd src/neutsmpl
wget https://www.dropbox.com/s/dxyjk5fzw0vuv85/jojo-NEUT-1.patch
wget https://www.dropbox.com/s/ew07kn9duzvhqtt/jojo-NEUT-2.patch
patch -p0 --dry-run < jojo-NEUT-1.patch
patch -p0 --dry-run < jojo-NEUT-2.patch
```

---

29. If there aren't errors, apply the patches:

---

```
patch -p0 < jojo-NEUT-1.patch
patch -p0 < jojo-NEUT-2.patch
```

---

30. Finally compile NEUT. All should be fine now.

---

```
./Makeneutsmpl.csh
```

---

# Chapter 5

# Appendix

## 5.1 GDCC cheat-sheet

This cheat-sheet is intended for debug only. Normal users (myself included) should not need to care about this. I have included it in the appendix just in case I would need it in future (but I really hope not). The GDCC is built on standard Ethernet. Communication to and from it is done via RAW Ethernet packets:

### 5.1.1 Fast command packet format

Fast commands are generated via two mechanisms. The first is in direct response to various hardware inputs from the CCC into the GDCC and the second is manually via the GDCC → COMPUTER link.

When done via this method the packet used is smaller than the normal GDCC packet format, and is processed separately.

Dst MAC	Src MAC	Ethernet Type	Command_Word	DIF Link	Comma	Data	parity	PAD	CRC32
6 Bytes	6 Bytes	2 bytes	2 Bytes	2 Bytes	1 Byte	1 Byte	2 Bytes	Pad to min Ethernet size	4 bytes

Table 5.1: Fast command packet format

- Ethernet type: set to 0x0809 for Fast Command. These are generally not used in the real world, so we chose them at random. Packets with a different Ethernet Type will be ignored.
- Command word: set to a constant 0xFA57. Future operations may use a different Command\_Word for other usage cases.
- DIF Link: A mask which defines which port the command is for. A value of 0xFFFF would be used as a broadcast to all currently active DIF links.
- Comma: comma character to use
- Data: defines which byte to send as the data byte.
- Parity: is a simple check, the bits of this are defined as follows

Bit	Data Used
0	Lower 8 bits of Command_Word
1	Upper 8 bits of Command_Word
2	Lower 8 bits of DIF_Link
3	Upper 8 bits of DIF_Link
4	Comma
5	Data

We use an Even Parity scheme, and the reason for using this is that the command is recognised almost as soon as the parity is validated, rather than waiting for the full CRC32 to be verified.

## 5.1.2 GDCC packet format

Dst MAC	Src MAC	Ethernet Type	GDCC_type	GDCC_modifier	GDCC_pktID	GDCC_dataLength	GDCC_Data	PAD CRC32
6 Bytes	6 Bytes	2 bytes	2 Bytes	2 Bytes	2 Byte	2 Byte	Variable Pad to min Ethernet size	4 bytes

Table 5.2: GDCC packet format

- Ethernet type:
  - 0x0810: GDCC data pkt
  - 0x0811: DIF data pkt
- GDCC type: split in 2 bytes (not all are defined)
  - Upper (Sub System Encoding):
    - \* 0x00: GDCC registers (Both directions)
    - \* 0x01: DIF transport (Both directions)
    - \* 0x02: Diagnose memory (Both directions)
    - \* 0xFF: GDCC pkt generator (GDCC →PC)
  - Lower (Operation Encoding):
    - \* 0x00: GDCC PktGenData (GDCC →PC)
    - \* 0x01: Write from PC to GDCC (PC →GDCC)
    - \* 0x02: Read from PC to GDCC (PC →GDCC)
    - \* 0x03: Write ACK (GDCC →PC)
    - \* 0x04: Read reply from GDCC to PC (GDCC →PC)
    - \* 0x05: Read NACK (GDCC →PC)
    - \* 0x08: pkt\_DIF from PC to DIF (PC →DIF)
    - \* 0x09: pkt\_DIF from DIF to PC (DIF →PC)
    - \* 0xFF: GDCC Saw Bad Packet (GDCC →PC)
- GDCC modifier: is used to indicate things such as which DIF link a DIF packet should be sent down, that is currently the only use of it. 0xFFFF indicates a broadcast down all DIF links that are currently operational.
- GDCC pktID : This is used to track Replies to things. For example a any GDCC register operations will result in a reply, these replies will have the same PktID in them.
- GDCC data length: is a measure of how many objects there will be in the GDCC Data array. For Register operations on the GDCC it is the total number of GDCC Register Packets that follow. For DIF operations it is the Number of DIF Packets that follow. For DIF Event data it is the Number of Event Packets that follow.
- CRC32: This is not really a user accessible data field. Normally the MAC layer on the Ethernet card will add this, and will strip it on received packets. However, depending on the operation it may or may not be visible and so is included in this definition for completeness. Any packets that fail the CRC32 check on RX at the GDCC will be silently dropped.

## 5.1.3 GDCC Register Packet Format

Access to GDCC registers is done via the following sub packet type. Address is 16 bits, and in general fill all

Address	Data
2 Bytes	4 Bytes

unused data with 0x0 Data is 32 bits, even though most registers are 16. This is to allow future expansion.

Lower bits of data are used first, so a register that returns < 32 bits will return it in the lower bits of the data space, the same for writes.

When performing a Read the packet must still include the space for the data, even though it will be overwritten by the GDCC internal processing. This just makes things more symmetric for both reads and writes.

You can pack as many register sub packets as you want into an GDCC\_packet. However, they will all be of the same type, eg, all READ or all WRITE, you cannot currently mix them.

### **5.1.4 GDCC DIF Event Packet Format**

When Events come into the GDCC from the DIF they are wrapped in an GDCC packet before being sent onward to the COMPUTER. They are dropped verbatim into the GDCC\_Data block of the packet. The GDCC-DIF Link CRC is retained, so that software can check it if needed.

- GDCC\_Type: Will have the upper portion set to DIF Transport and the lower 8 bits set to show which DIF Link it came from.
- GDCC\_PktID: Will be the serial number of the packet, which will increase each time, allowing some way to see if there are missing ones.
- GDCC\_DataLength: Will show the number of encapsulated DIF Packets

Future enhancement might be the addition of a flag to say if the packet from the DIF passed the CRC or not.

### **5.1.5 GDCC Memory Map**

Memory access's to registers inside the GDCC are done using a 16bit address. This address is then subdivided into a Block and Register range. The upper 4 bits define the Block. The lower 12 define the Register.

<b>Block</b>	<b>Address</b>	<b>Register</b>
0x1	0x000	DIF_LINK_TX_EN
	0x002	DIF_LINK_RX_EN
	0x004	DIF_LINK_RTT
	0x006	DIF_LINK_AUTONEG_PAUSE
	0x008	DIF_LINK_RESTART
	0x00A	DIF_LINK_STATUS1
	0x00B	DIF_LINK_STATUS2
	0x00E	DIF_LINK_NO_SIGNAL
	0x010	DIF_LINK_DELAY1
	0x011	DIF_LINK_DELAY2
	0x012	DIF_LINK_DELAY3
	0x013	DIF_LINK_DELAY4
	0x018	DIF_LINK_RTT1
	0x019	DIF_LINK_RTT2
	0x01A	DIF_LINK_RTT3
	0x01B	DIF_LINK_RTT4
	0x020	DIF_LINK_RTT_DONE
	0x022	DIF_LINK_LOCKED
	0x024	DIF_LINK_DCM
0x2		
0x3		
0x4	0x000	GDCC_ENABLES
	0x001	GDCC_TX_MUX_COUNT
	0x006	GDCC_DIF_DATA_MAC_L
0x007 g	0x007	GDCC_DIF_DATA_MAC_M
	0x008	GDCC_DIF_DATA_MAC_H
	0x00E	GDCC_REVISION
	0x00F	GDCC_VERSION
	0x010	GDCC_PKTGEN_CONTROL
	0x011	GDCC_PKTGEN_SIZE
	0x012	GDCC_PKTGEN_COUNT
	0x013	GDCC_PKTGEN_DELAY
	0x014	GDCC_PKTGEN_SEED
	0x015	GDCC_PKTGEN_TXCOUNT
	0x016	GDCC_PKTGEN_MAC_L
	0x017	GDCC_PKTGEN_MAC_M
	0x018	GDCC_PKTGEN_MAC_H

# Bibliography

- [1] Ortec Ametek. *Preamplifier Introduction*. English. 6 pp. URL: <https://www.ortec-online.com/-/media/ametekortec/other/preamplifier-introduction.pdf?la=en>.
- [2] G Bertuccio, A Pullia, and G De Geronimo. “Criteria of choice of the front-end transistor for low-noise preamplification of detector signals at sub-microsecond shaping times for X- and  $\gamma$ -ray spectroscopy”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 380.1 (1996). Proceedings of the 9th International Workshop on Room Temperature Semiconductor X- and  $\gamma$ -Ray Detectors, Associated Electronics and Applications, pp. 301–307. ISSN: 0168-9002. DOI: [https://doi.org/10.1016/S0168-9002\(96\)00474-3](https://doi.org/10.1016/S0168-9002(96)00474-3). URL: <http://www.sciencedirect.com/science/article/pii/S0168900296004743>.
- [3] S. Callier et al. “Silicon Photomultiplier integrated readout chip (SPIROC) for the ILC: Measurements and possible further development”. In: *2009 IEEE Nuclear Science Symposium Conference Record (NSS/MIC)*. Oct. 2009, pp. 42–46. DOI: <10.1109/NSSMIC.2009.5401891>.
- [4] Stéphane Callier et al. *SPIROC : Silicon PM Readout ASIC*. [https://indico.cern.ch/event/21985/contributions/1522359/attachments/356872/496914/J\\_Fleury-TWEPP08-SPIROC.pdf](https://indico.cern.ch/event/21985/contributions/1522359/attachments/356872/496914/J_Fleury-TWEPP08-SPIROC.pdf). [Online; accessed 2-November-2018]. 2008.
- [5] R Cornat, F Gastaldi, and F Magniette. “Acquisition and control command system for power pulsed detectors”. In: *Journal of Instrumentation* 9.01 (2014), p. C01030. URL: <http://stacks.iop.org/1748-0221/9/i=01/a=C01030>.
- [6] R. Fabbri, B. Lutz, and W. Shen. “Overview of Studies on the SPIROC Chip Characterisation”. In: *ArXiv e-prints* (Nov. 2009). arXiv: [0911.1566 \[physics.ins-det\]](0911.1566 [physics.ins-det).
- [7] Franck Gastaldi and Remi Cornat. *Gigabit Data Concentrator Card*. [https://agenda.linearcollider.org/event/5484/contributions/24339/attachments/19955/31599/GDCC\\_caliceweek\\_japan.pdf](https://agenda.linearcollider.org/event/5484/contributions/24339/attachments/19955/31599/GDCC_caliceweek_japan.pdf). [Online; accessed 2-November-2018]. 2012.
- [8] Franck Gastaldi et al. “A scalable gigabit data acquisition system for calorimeters for linear collider”. In: *PoS TIPP2014* (2014), p. 193. DOI: <10.22323/1.213.0193>.
- [9] OMEGA microelectronics group. *SPIROC2D Datasheet*. [Online; accessed 2-November-2018]. URL: <https://drive.google.com/file/d/11w0rbJyYjQiRsMrM-DFjh6vm0zg8Q5Z/view?usp=sharing>.
- [10] OMEGA microelectronics group. *SPIROC2D Pin List*. [Online; accessed 2-November-2018]. URL: <https://drive.google.com/file/d/1nFgbT5x7ScryKyfCsVz8-4NdzL8xVRds/view?usp=sharing>.
- [11] Texas Instruments. *CDCLVC11xx 3.3-V and 2.5-V LVCMOS High-Performance Clock Buffer Family*. English. Version SCAS895B – MAY 2010 – REVISED FEBRUARY 2017. 2017. 32 pp. URL: <http://www.ti.com/lit/ds/symlink/cdclvc1104.pdf>.
- [12] Hamamatsu Photonics K.K. *Characteristics and use of Charge amplifier*. English. Version TECHNICAL INFORMATION SD-37. 2001. 10 pp. URL: [https://www.hamamatsu.com/resources/pdf/ssd/charge\\_amp\\_kacc9001e.pdf](https://www.hamamatsu.com/resources/pdf/ssd/charge_amp_kacc9001e.pdf). November 16, 2011.
- [13] G.F. Knoll. *Radiation Detection and Measurement*. John Wiley & Sons, 2010. ISBN: 9780470131480. URL: <https://books.google.co.jp/books?id=4vTJ7UDel5IC>.
- [14] G. Lioliou and A.M. Barnett. “Electronic noise in charge sensitive preamplifiers for X-ray spectroscopy and the benefits of a SiC input JFET”. In: *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 801 (2015), pp. 63–72. ISSN: 0168-9002. DOI: <https://doi.org/10.1016/j.nima.2015.08.042>. URL: <http://www.sciencedirect.com/science/article/pii/S0168900215009766>.
- [15] S Conforti Di Lorenzo et al. “SPIROC: design and performances of a dedicated very front-end electronics for an ILC Analog Hadronic CALorimeter (AHCAL) prototype with SiPM read-out”. In: *Journal of Instrumentation* 8.01 (2013), p. C01027. URL: <http://stacks.iop.org/1748-0221/8/i=01/a=C01027>.

- [16] Frédéric Magniette, Miguel Rubio-Roy, and Floris Thiant. “Pyrame, a rapid-prototyping framework for online systems”. In: *Journal of Physics: Conference Series* 664.8 (2015), p. 082028. URL: <http://stacks.iop.org/1742-6596/664/i=8/a=082028>.
- [17] OMEGA - CENTRE DE MICROÉLECTRONIQUE. *SPIROC*. URL: <https://portail.polytechnique.edu/omega/en/products/products-presentation/spiroc>.
- [18] Stéphane Callier on behalf of OMEGA microelectronics group. *SPIROC2D characterization*. [https://agenda.linearcollider.org/event/6931/contributions/34075/attachments/28104/42545/SC\\_SPIROC2D\\_measurements\\_20151210\\_pdf.pdf](https://agenda.linearcollider.org/event/6931/contributions/34075/attachments/28104/42545/SC_SPIROC2D_measurements_20151210_pdf.pdf). [Online; accessed 2-November-2018]. 2015.
- [19] Miguel Rubio-Roy, Floris Thiant, and Frédéric Magniette. “Flexible online monitoring for high-energy physics with Pyrame”. In: *Journal of Physics: Conference Series* 898.3 (2017), p. 032009. URL: <http://stacks.iop.org/1742-6596/898/i=3/a=032009>.