APC, AstroParticule et Cosmologie, Université Paris Diderot, CNRS/IN2P3, CEA/Irfu, Observatoire de Paris, Sorbonne Paris Cité, 10, rue Alice Domon et Léonie Duquet, 75205 Paris Cedex 13, France

WFEE ASIC BUS User Guide

	Name & Society	Date	Signature
Prepared by	**Bernard Courty APC**	**2021**	
Approved by	**Damien PRELE APC**	**27/07/2023**	
Contributors		**../../..**	

INDEXATION NOTE

KEYWORDS : I2C, RS485, WFEE, BUS
TITLE : WFEE ASIC BUS User Guide
AUTHORS : **Bernard Courty**
SUMMARY : description of the WFEE ASIC I2C/RS485 bus, this document deals only with the digital interface between the WFEE ASICs and the RTU, it will not describe analog signals (see WFEE_INTERFACE_DOC) neither power supply characteristics and requirements.   ce document s'applique à la V1.1 depuis 2021 (un exemplaire V1.0 reste à l'APC)   .

MODIFICATION CHANGES

Issue	Date	Modifications
1	23/04/2019	Document creation
2	25/04/2019	Corrected and approved
3	25/11/2021	New schematic and photograph of the PCB in the box
4	27/09/2023	Pinout subD9 ATHENA Pic and new information about front panel box

INDEX

1 PURPOSE OF THE DOCUMENT 5
2 ACRONYMS 5
3 CONTEXT 5
4 I2C BUS AND RS485 6
4.1 LOGICAL CONSIDERATIONS 6
4.1 ELECTRICAL CONSIDERATIONS 6
5 TEST SETUP EXAMPLE 8
5.1 HARDWARE EXAMPLE 8
5.2 SOFTWARE EXAMPLE 12
6 BOX AND PINOUT CONNECTOR 14

1 PURPOSE OF THE DOCUMENT

The purpose of this document is to describe how to use the WFEE ASIC bus (connecting with RTU) in order to set the SQUID bias/offset of the FPA.
This document deals only with the digital interfacing with the ASICs, it will not describe the analog signals (see WFEE_INTERFACE_DOC) neither power supply characteristics and requirements.

2 ACRONYMS

Abbr.	Signification
ASIC	Application-Specific Integrated Circuit
ATHENA	Advanced Telescope for High-ENergy Astrophysics
FPA	Focal Plane Assembly
I^2C	Inter-Integrated Circuit
ICU	Instrument Control Unit
RTU	Remote Terminal Unit
SCL	Serial Clock Line
SDA	Serial Data Line
SQUID	Superconducting QUantum Interference Device
TIA	Telecommunications Industry Association
WFEE	Warm Front-End Electronics
X-IFU	X-ray Integral Field Unit

3 CONTEXT

The WFEE biases SQUIDs of the FPA with a digital control from the ICU through a RTU. This is achieved by the mean of ultra-stable current sources, which are digitally adjustable on 8 bits (as a slow-DAC).

To avoid multiplying cables on the system, we use an I2C bus to control tens of current sources.

The current WFEE design is composed of 3 boxes (at 120deg from each other). In each box will be 4 ASICs and two buses. Each bus will allow controlling 2 ASICs.

As those 2 ASICs are identical, the differentiation will be done by setting some switches (for the less, one) on the electronic boards: one of the I2C address bit will be set to “1” on one board, and set to “0” for the other.

This digital BUS is used as a Slow Control to setup the ASIC current sources, and is not used during the observation mode and could be switched off (by the RTU during this mode).

4 I2C BUS AND RS485

4.1 LOGICAL CONSIDERATIONS

The ATHENA WFEE I2C bus lets a Master (RTU-ICU) read and write 8bits data from several devices (up to 128, identified by a 7bits address), using only 2-octets frames.

A WFEE ASIC includes tens of I2C devices, each one has a unique 7bits address (one of these bits is set on the board) and controls one 8bits register (i.e. one DAC).

The purpose of I2C is to transfer octet (8bits) of data in a serial way, using two signal lines:

SCL: unidirectional clock from the master to the devices, up to 400kHz is supported,

(a bidirectional behaviour has also been studied inside the WFEE ASIC).

SDA: for data and address, synchronised with the SCL line.

A negative edge of **SDA** during **SCL** "high" means a START of frame.

A positive edge of **SDA** during **SCL** "high" means a STOP of frame.

During the rest of the frame, SDA **must** change **only** when **SCL** is "low".

An I2C frame is a succession of [8bits data + 1bit acknowledge]. The first in every I2C frame is composed by the **7bits ID** (MSB first) of the I2C device that must be addressed, followed by **R/W** ("1" means Read,"0" means Write) and the one bit **Ack**nowledge from the device.

For the Athena WFEE I2C devices, the second octet of the I2C frame will simply be the 8bits value (MSB first) that you want to read from or write in its register.

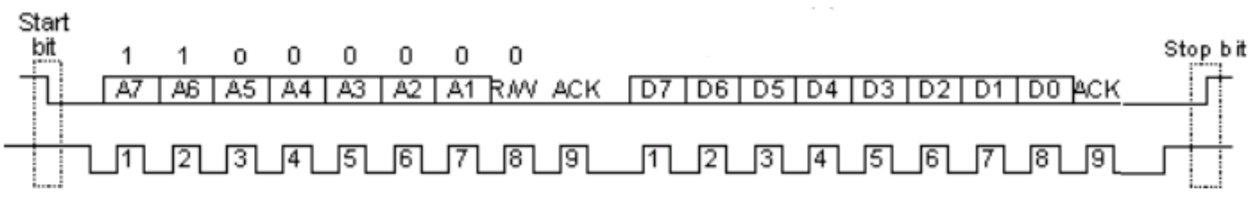


Figure 1 : WFEE I2C Frame

4.1 ELECTRICAL CONSIDERATIONS

On usual I2C bus, **SCL** and **SDA** are non-differential, 0 to Vdd signals. They are open-collector so no care of signal direction is needed.

ATTENTION: In the WFEE ASIC, to improve both power considerations and EMC efficiency, **SCL** and **SDA** are converted into differential low voltage signals, RS485-like. Then the direction of **SDA** must be switched during the I2C frame.

From the master point of view, **SDA** is bidirectional and must be set in the **WRITE** direction during **A7-A1**, **R/W** and during **D7-D0** if **R/W**=0

READ direction must be used during the transfer of **D7-D0** if **R/W**=1 and during some **Ack**

As the behaviour of the "**Ack**nowledge" inside the ASIC is still subject to change, I would recommend ignoring it and setting the **SDA** in the **READ** direction (i.e. Output disabled) from the master point of view.

As the **SDA** signal must not toggle when **SCL** is high, I recommend switching the direction immediately after the **SCL** going low.

The electrical communication between the WFEE ASICs and the RTU will be made by those RS485-like signals and should follow some TIA-485 recommendations:

Ideally, the two ends of the cable will have a termination resistor connected across the two wires. Without termination resistors, signal reflections can cause data corruption. The value of each termination resistor should be equal to the cable characteristic impedance (typically, 120 ohms for twisted pairs). As SCL is currently always unidirectional the resistor could be omitted on the Master side.
The termination also includes pull-up and pull-down resistors to establish fail-safe bias for each data wire for the case when the lines are not being driven by any device. These resistors are present on the WFEE ASIC board.

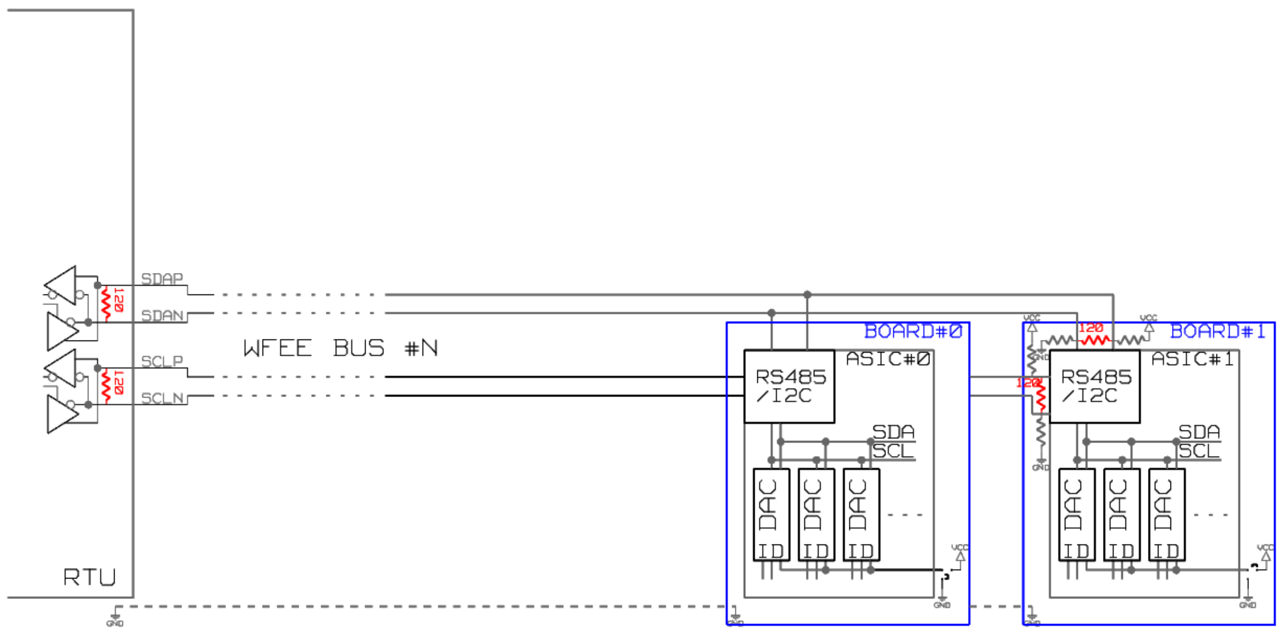


Figure 2 : WFEE BUS connection

I recommend using 3.3V-powered differential device or driver in order not to overload the WFEE ASIC IOs protection (3.3V). I also recommend using a slew-rate driver to optimize the signal transmission in the line between

RTU and WFEE ASIC as for EMC consideration. As the length of the differential line will not be above few meters, a Vout of 1.5Vdiff is not mandatory, instead power could be saved as it's done in the WFEE ASIC using Vout around 300mVdiff.
The TIA standard (ANSI/TIA/EIA-485-A, page 15, A.4.1) requires the presence of a common return path between all circuit grounds along the balanced line for proper operation. This might be ignored if low power and slew rate limited signals are used, reducing EMI. A Voltage reference connection is still required between the electronic boards and performed by the mechanical structure due to "mGnd".

5 TEST SETUP EXAMPLE

5.1 HARDWARE EXAMPLE

The WFEE ASIC I2C+RS485 BUS has been successfully tested using a 8bit microcontroller from Microchip: PIC18F46J50 together with MAX13432 RS485 drivers.

This test device allows sending a read or a write frame to the WFEE ASIC board, using the two (SDA and SCL) differential lines.
It is powered and controlled by a Desktop Computer via a USB2.0 interface.
The WFEE ASIC board must be powered separately and before plugging or powering the USB test device.

The real-time firmware running on the PIC18F46J50 can send a Read or a Write frame to any I2C device address at 400kHz. It has been developed as bare-metal programming.

The firmware can be reprogrammed into the PIC18F46J50 via a PICKIT3 or via the USB (using HIDbootloader). When the test device is powered-on with its push-button pressed, it enters the USB-reprogrammable mode. Otherwise it stays in normal operation mode.

Any Computer can send orders to the device as a USB bulk frame, using for example libusb.

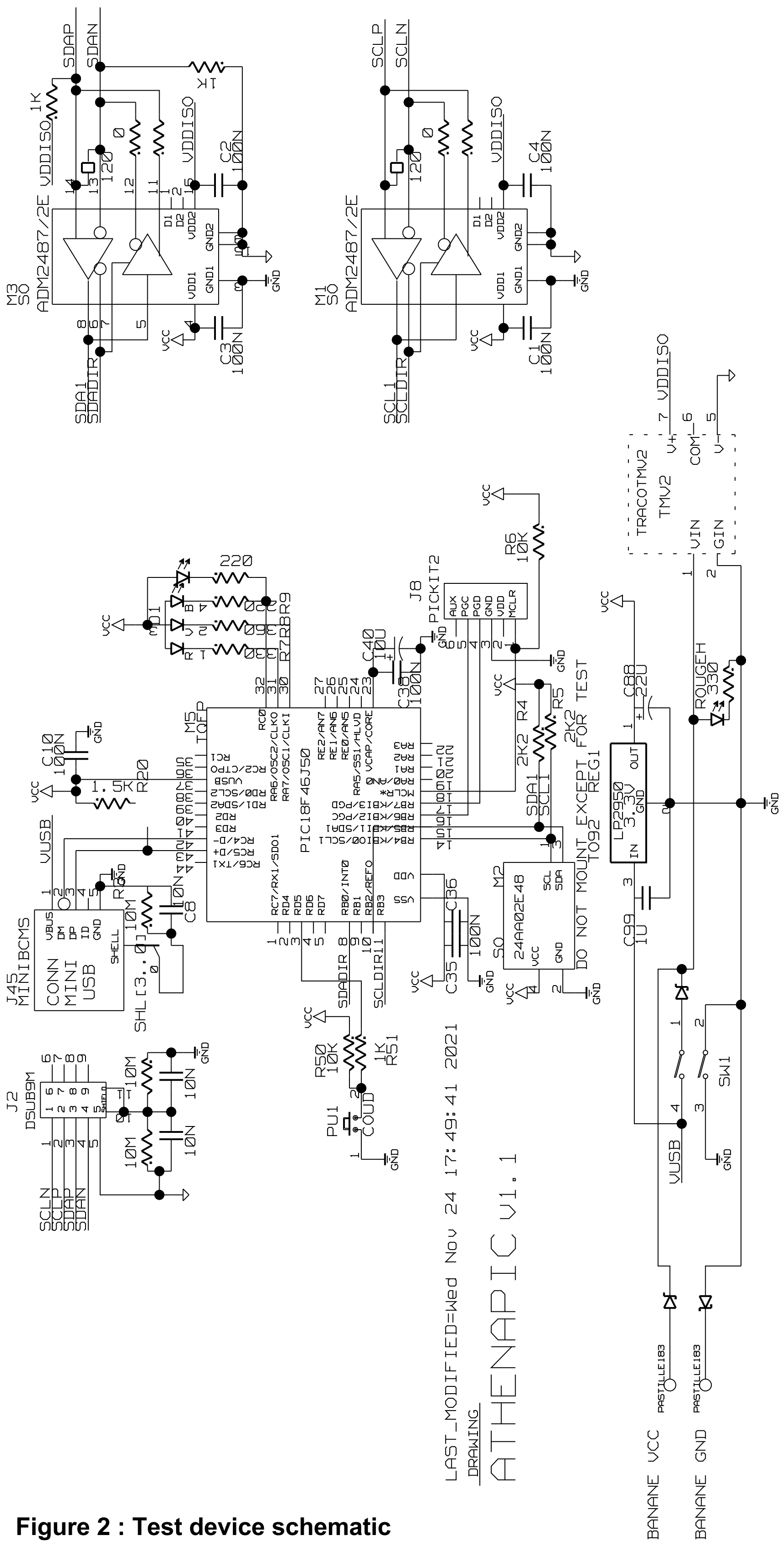


Figure 2 : Test device schematic

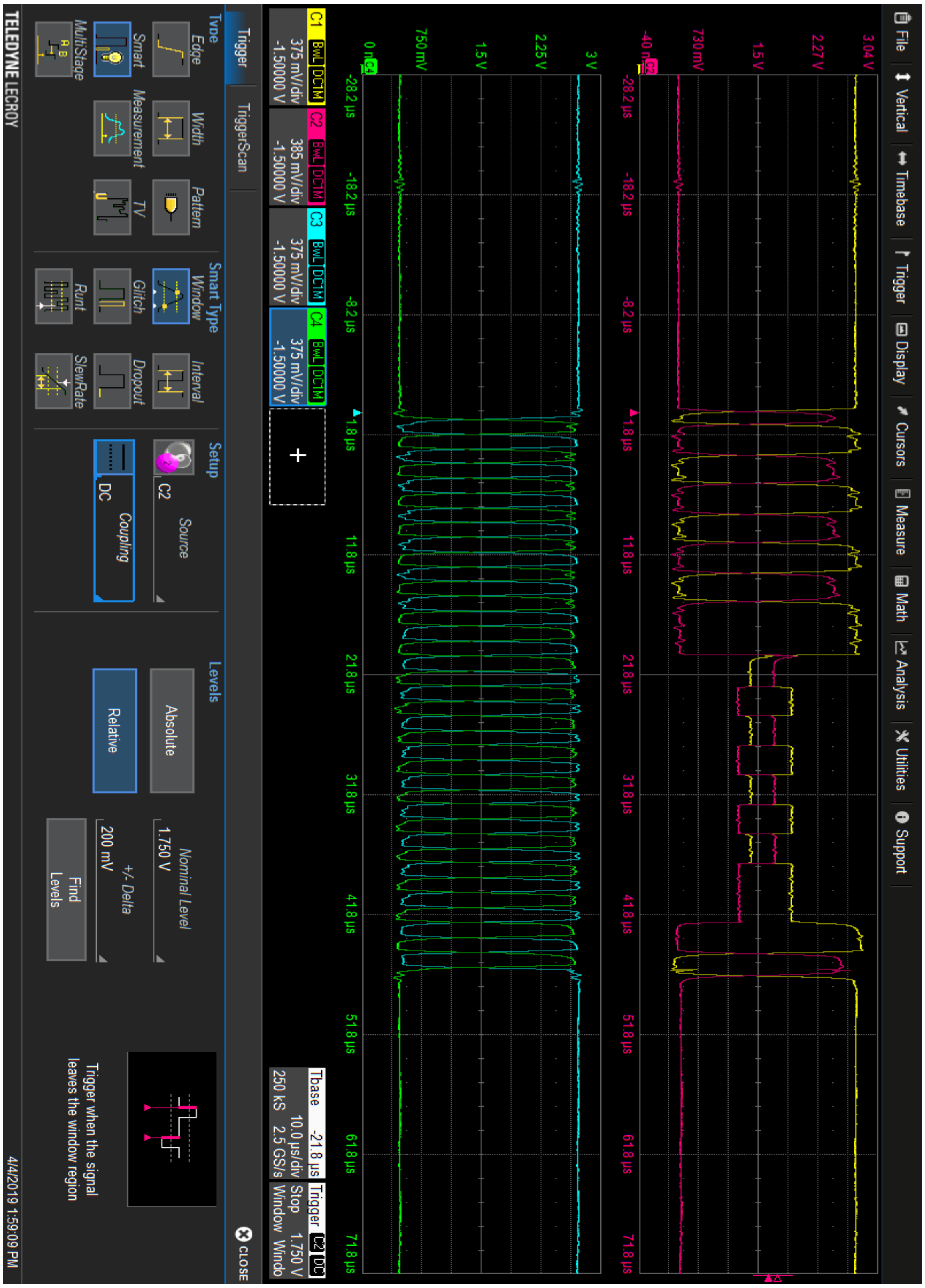
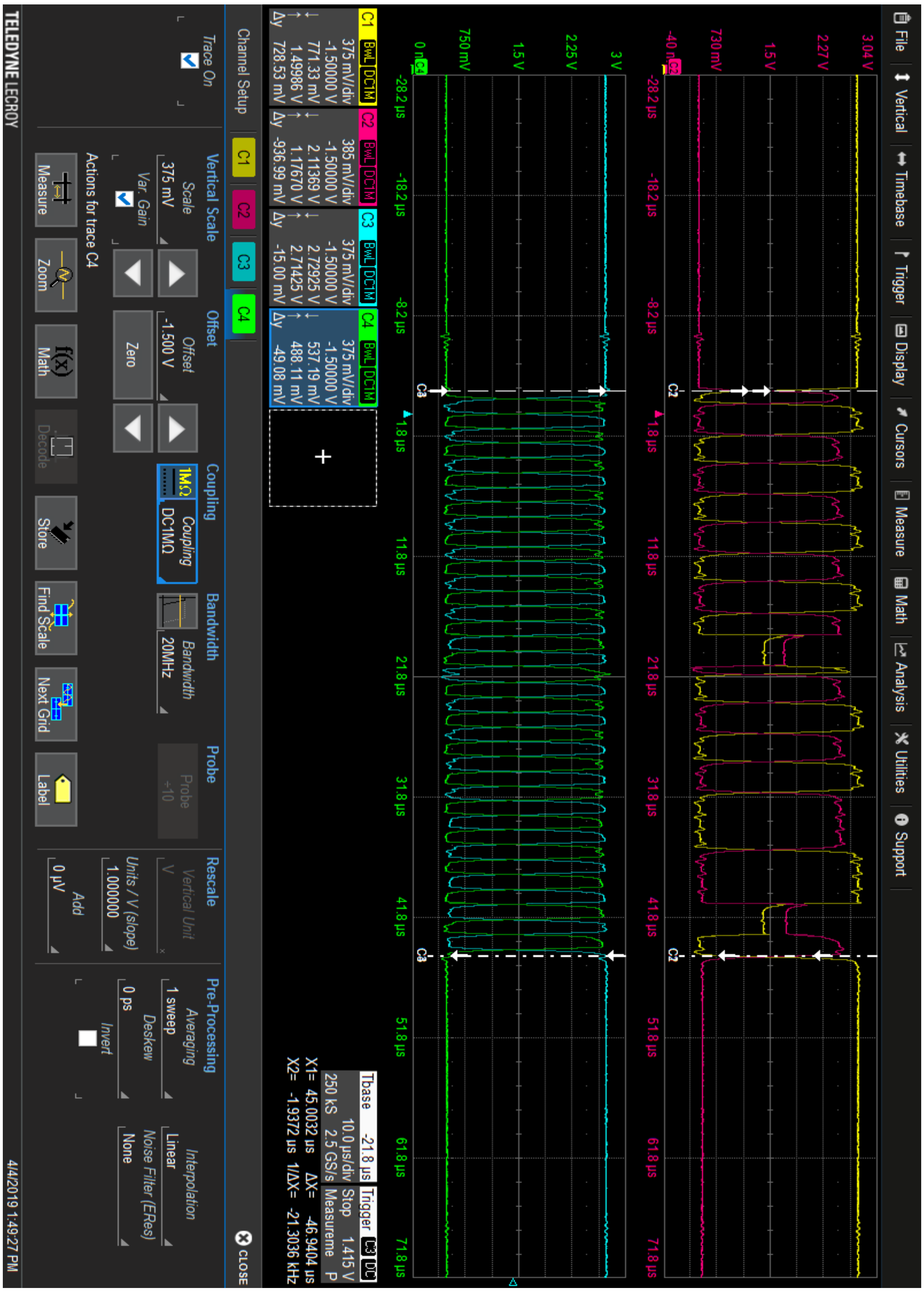

C1 BwL DC1M 375 mV/div -1.50000 V
C2 BwL DC1M 385 mV/div -1.50000 V
C3 BwL DC1M 375 mV/div -1.50000 V
C4 BwL DC1M 375 mV/div -1.50000 V
Tbase -21.8 µs 10.0 µs/div 250 kS 2.5 GS/s
Trigger C2 DC Stop 1.750 V Window Window
Source C2
Coupling DC
Nominal Level 1.750 V
+/- Delta 200 mV
Trigger when the signal leaves the window region
4/4/2019 1:59:09 PM

Figure 4 : Read Frame



```
C1 BwL DC1M      C2 BwL DC1M      C3 BwL DC1M      C4 BwL DC1M
   375 mV/div       385 mV/div       375 mV/div       375 mV/div
   -1.50000 V       -1.50000 V       -1.50000 V       -1.50000 V
↓  771.33 mV     ↓  2.11369 V     ↓  2.72925 V     ↓  537.19 mV
↑  1.49986 V     ↑  1.17670 V     ↑  2.71425 V     ↑  488.11 mV
Δy 728.53 mV     Δy -936.99 mV    Δy -15.00 mV     Δy -49.08 mV

Tbase  -21.8 µs   Trigger C3 DC
10.0 µs/div       Stop    1.415 V
250 kS  2.5 GS/s  Measureme  P
X1= 45.0032 µs    ΔX= -46.9404 µs
X2= -1.9372 µs    1/ΔX= -21.3036 kHz
```

Figure 5 : Write Frame

5.2 SOFTWARE EXAMPLE

We have successfully used **LIBUSB0.1** under **Linux** and **LIBUSB1.0** under **Windows7**/**MinGW**

Here are software examples and explanations for **Windows7** only:

First you will need to install/select the correct windows driver for the USB device:

I highly recommend using Zadig e.g. zadig_2.2.exe from https://zadig.akeo.ie/downloads/

After plugging-in the USB device, run Zadig:

Options-> List All Devices

If needed, select: "**My Libusb Device**" (USB ID = **04D8 4568**)

Select the driver **libusbK** (should work also with **WinUSB**)

Then push **Reinstall Driver**

We have used Libusb1.0 e.g. **libusb-1.0.22.7z** from https://libusb.info/

to compile the software under MinGW

N.B.: libusb0.1 has a slightly different API

Example of code for USB device init (and release) in C for MinGW libusb1.0 Windows7

```
#include<stdio.h>
#include<libusb.h>

#define MY_VID 0x04D8
#define MY_PID 0x4568

#define EP_IN 0x81
#define EP_OUT 0x01

int main()
{
  libusb_context *myctx = NULL;
  libusb_device_handle *MyLibusbDeviceHandle = NULL;
  struct libusb_device **devs;
  struct libusb_device *dev;
  struct libusb_device_descriptor desc;

  int nb,err,i;

  err = libusb_init(&myctx);
  if(err) {printf("usb_init %s\n",libusb_strerror(err));exit(-1);}

  nb = libusb_get_device_list(myctx, &devs);
  dev = NULL;
  for(i=0; i<nb; i++)
  {
    err = libusb_get_device_descriptor(devs[i],&desc);
    if(err) {printf("usb_get_dev_desc %s\n",libusb_strerror(err));exit(-1);}
    if(desc.idVendor==MY_VID && desc.idProduct==MY_PID)
     {dev=devs[i];break;}
  }
  if (dev==NULL) {printf("No device found\n");exit(-1);}

  err = libusb_open(dev,&MyLibusbDeviceHandle);
  if(err) {printf("usb_open %s\n",libusb_strerror(err));exit(-1);}

  if (err=libusb_set_configuration(MyLibusbDeviceHandle, 1))
  {
    printf("usb_set_config %s\n",libusb_strerror(err));
    libusb_close(MyLibusbDeviceHandle);
    return(-1);
  }
  if((err=libusb_claim_interface(MyLibusbDeviceHandle, 0)))
  {
    printf("usb_claim_int %s\n",libusb_strerror(err));
    libusb_close(MyLibusbDeviceHandle);
    return(-1); ;
  }



  Sleep(1000); // here is
  Sleep(1000); // your arbitrary code
  Sleep(1000); //   ...




  libusb_release_interface(MyLibusbDeviceHandle, 0);
  libusb_close(MyLibusbDeviceHandle);

  libusb_free_device_list(devs, 1);

  libusb_exit(myctx);
  return 0;
}
```

Here are some samples of code used for the WFEE ASIC setup/read/etc :

```
  unsigned char OutPak[64], InPak[64];
  int err,ll;
```

// You can toggle the LED just to check you can reach the USB device:

```
  OutPak[0] = 0x86; // command to toggle the LED on the UDB device
  if (err=libusb_bulk_transfer(MyLibusbDeviceHandle, EP_OUT, &OutPak[0], 64, &ll, 2000))
   {printf("bulk_write %s\n",libusb_strerror(err)); Sleep(500);}
  if (ll!=64) {printf("incomplete USB write %i\n",ll);}

  if (err=libusb_bulk_transfer(MyLibusbDeviceHandle, EP_IN, &InPak[0], 64, &ll, 5000))
   {printf("bulk_read %s\n",libusb_strerror(err)); Sleep(500);}
```

// Setting the I2C device (DAC) #40 with value 0x23 :

```
  OutPak[0] = 0x84;  /* command to write I2C */
  OutPak[1] = 40; /* 0 to 127 , 7-bits significant I2C dev addr */
  OutPak[2] = 0x23; /* 8-bits value */
  if (err=libusb_bulk_transfer(MyLibusbDeviceHandle, EP_OUT, &OutPak[0], 64, &ll, 5000))
   { printf("bulk_write %s\n",libusb_strerror(err)); Sleep(500);}
  if (ll!=64) {printf("incomplete USB write %i\n",ll);}
  if (err=libusb_bulk_transfer(MyLibusbDeviceHandle, EP_IN, &InPak[0], 64, &ll, 5000))
   {printf("bulk_read %s\n",libusb_strerror(err)); Sleep(500);}
// printf("%02x %02x %02x \n",InPak[0],InPak[1],InPak[2]);
// this last line can show you the frame returned by the device
// you can see that the I2C address and value are swapped.
```

// Getting the value of I2C device #99 (DAC) :

```
  OutPak[0] = 0x82;  /* I2C read */
  OutPak[1] = 99;
  if (err=libusb_bulk_transfer(MyLibusbDeviceHandle, EP_OUT, &OutPak[0], 64, &ll, 5000))
   {printf("bulk_write %s\n",libusb_strerror(err)); Sleep(500);}
  if (ll!=64) {printf("incomplete USB write %i\n",ll);}

  if (err=libusb_bulk_transfer(MyLibusbDeviceHandle, EP_IN, &InPak[0], 64, &ll, 5000))
   {printf("bulk_read %s\n",libusb_strerror(err)); Sleep(500);}
  if (ll!=64) {printf("incomplete USB read %i\n",ll);}
  printf("%02x %02x %02x \n",InPak[0],InPak[1],InPak[2]);
// you can see the value returned in InPak[1]
```

Note several USB devices could be built with modified USB ID = **04D8 4569 …**
In order to be able to control several WFEE BUSes at the same time.

6 BOX AND PINOUT CONNECTOR

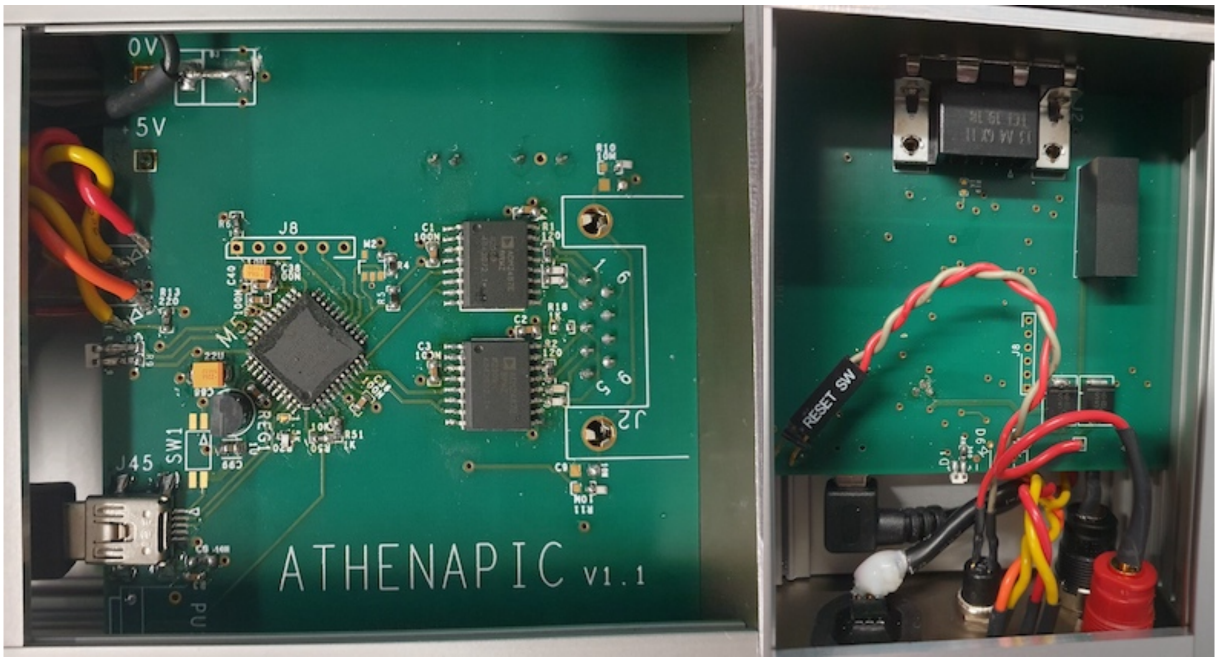


Figure 3 : ATHENAPIC send to IRAP in nov. 2021

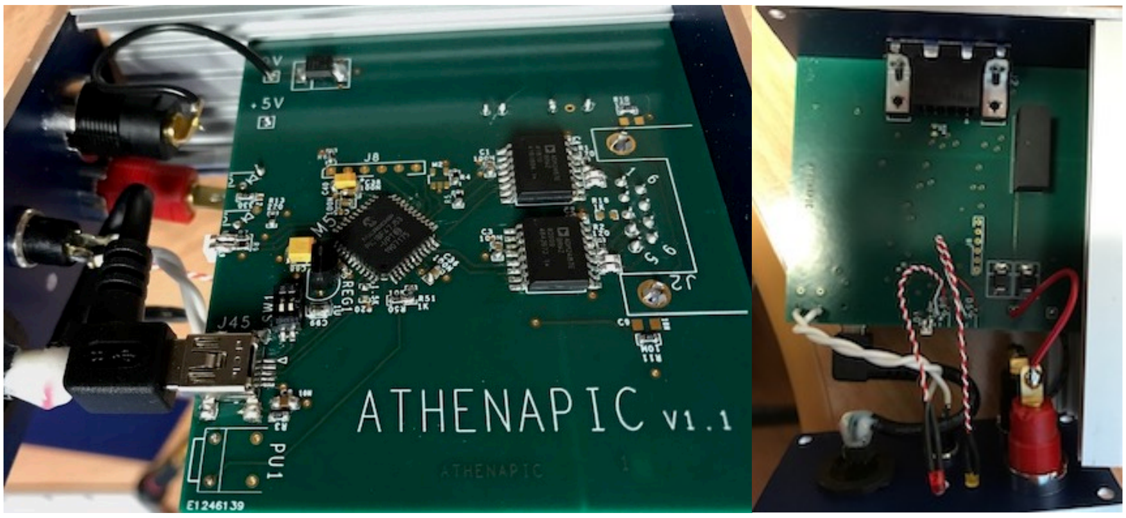
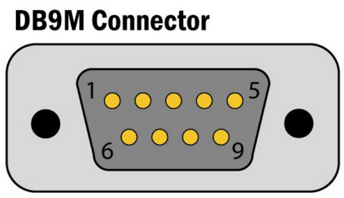


Figure 4 : The ATHENAPIC @ APC



Pin #	Signal
1	SCL-
2	SCL +
3	SDA +
4	SDA -
5	NC
6	NC
7	NC
8	NC
9	NC

➔ pin5 est reliée au GND des drivers RS485, isolés galvaniquement, et peut etre utilisée si besoin.

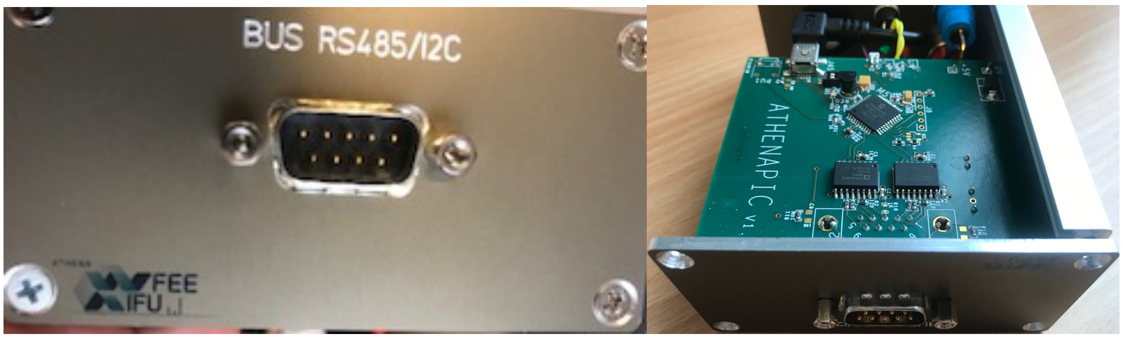


Figure 5 : pinout of the subD9 male connector of the ATHENApic Box (shell is connected to the ATHENApic box but remains floating from any potential)

The other side of the box is dedicated to the usb connection to a computer. A 5V external power to supply the dc/dc converter with galvanic isolation which operates the RS485 drivers"

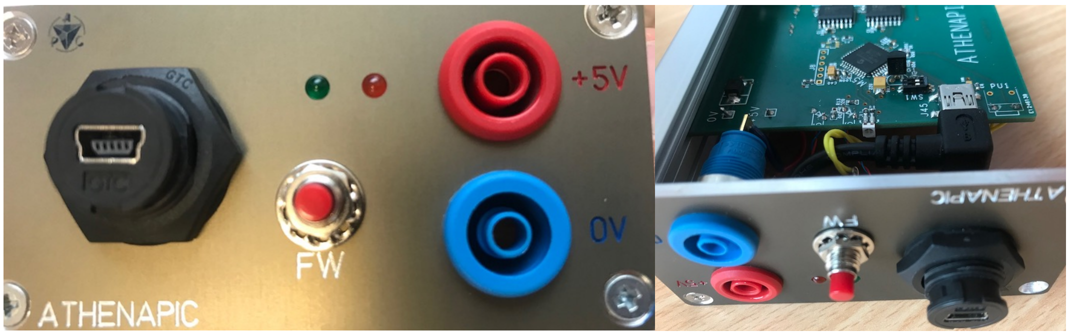


Figure 6: Face of the ATHENApic showing the USB connection and an optional 5V connection to bias the RS485 drivers (ADM2487) alternatively from the usb power