VMEDAQ

发布 *20211226*

Hongyi Wu(吴鸿毅)

2022 年 08 月 24 日

content:

感谢您选择 PKU VMEDAQ。更多开源程序，请访问我们的 Github 和 吴鸿毅的 Github

CHAPTER 1

简介

本程序为 **北京大学实验核物理组**当前使用的模拟获取系统，简称为 VMEDAQ。

该获取基于 RIKEN 的获取发展而来。我们已经对原本程序进行较大的修改。如果使用本程序，请严格使用本程序包内程序，请勿随意升级/替换程序包内部程序/固件。

如有使用需求，请直接联系吴鸿毅。

1.1 版本

最新版本 2022.08.16

程序下载请访问: VMEDAQ

网页版说明书请访问: 说明书

1.2 关于

本程序历史维护:

- 李智焕
- 李晶
- 臧宏亮
- 吴鸿毅 (wuhongyi@qq.com / wuhongyi@pku.edu.cn)

说明书贡献者:

- 王东玺（中国原子能科学研究院）
- 孙立杰（中国原子能科学研究院）
- 吴鸿毅

1.3 性能介绍

- 本获取经过 Scientific Linux 6/7 系统测试。即将进行 Ubuntu 20.04 测试。
- 程序默认支持控制器 CAEN V1718/V2718，CAEN V830 Scaler，CAEN V7xx 系列 ADC/QDC/TDC，CAENV1190/V1290 TDC，MESYTEC MADC32
- 支持多个机箱同步获取。将插件分散在多个机箱，可大大减少数据传输的死时间。
- 本获取分软件 busy 跟硬件 busy 两种模式。
- **对软件 busy 模式**
  - 该模式下，一个事件的死时间由 trigger 门宽，7 us 左右模数转换时间，20 us 数据传输中断请求及数据传输时间组成。其中除了数据传输时间，其它三个时间是固定的，大约为 30 us。
  - 限制该模式下计数率的因素为数据传输时间，数据越大，所需传输时间也就越长。
  - 以一个机箱，300-500 路左右输入为例，平均 10000 个触发能够记录 5000-6000 个事件，效率在 50-60%
  - 如果以两三个插件为例，则能够达到 70% 以上
- **对硬件 busy 模式**
  - 该模下式，一个事件的死时间由 trigger 门宽，7 us 左右模数转换时间两部分组成。
  - 意味着该模式下，一个事件的死时间大约在 11 us 左右。
  - 该模式模数转换及数据传输同步进行，因而数据高速传输产生的高频信号会对前放/主放的信号带来微小的影响。
  - 通过适当抬高阈值可消除该影响。
  - 该模式下获取效率极高，平均 10000 个触发能够记录 9000 多个事件，效率达到 90% 以上。

1.4 目录

文件夹内有以下文件/文件夹：

- analysis（一些用来辅助分析的代码）
- anaroot（底层库，用来将原始数据转为 ROOT 及在线统计）
- checkcnt（自动检查数据事件关联情况）
- cutpedo（自动拟合推荐合适 pedestal）
- DAQConfig（获取控制包）
- firmware（固件）
- httponline（基于网页的在线监视）
- online（在线监视能量，能谱）
- r2root（数据转换）
- source（babirl 源码，将会配置自动化安装脚本）
- statistics(时时监视每路信号的计数率，每 10 ns 更新一次)
- README.rst（本文件）
- docs(网页版说明书)
- README（rst 版说明书）

- README.pdf（pdf 说明书）

CHAPTER 2

软件安装

本页面安装软件放置在 source 文件夹内，里面包括 **获取驱动、依赖库等以及自动安装脚本**。

2.1 系统要求

本获取经过 Scientific Linux 6/7 系统测试。建议采用 CentOS 6/7 或者 Scientific Linux 6/7。

本获取要求 CERN ROOT 5/6，建议优先选择 ROOT 6。

如果没有合适的系统，可参考我们的获取系统安装 Install Scientific 7 。安装好系统之后，还需要对基础依赖工具做一些安装及升级，可以下载执行 自动化安装脚本 自动配置或者按照教程手动安装。

2.2 CAEN Lib

本程序依赖 CAENVMELib/CAENComm/CAENUpgrader 三个库文件。

其中 CAENVMELib/CAENComm 为获取运行必须的库。CAENUpgrader 用来更新固件。

进入 source 文件夹内，在 ROOT 权限下执行 setup.sh 脚本，将会自动安装以上三个依赖库。

```
# 在 source 文件夹内，ROOT 权限下执行以下命令
sh setup.sh    #需要ROOT权限
```

- **CAENVMELib-3.3.0-build20210806.tgz**
  - 该版本驱动存在问题，对 V1718/V2718 I/O 寄存器控制存在 bug
- **CAENVMELib-2.50.tgz**
  - 暂时推荐使用

2.3 检查 CAENVMELib 安装

进入 CheckRegisterToolByV2718 文件夹，make 编译里面程序，如果生成一个名为 pku 的可执行文件，则软件安装成功。

```
cd CheckRegisterToolByV2718
make
```

2.4 检查 CAENUpgrader 安装

- **CAENUpgrader-1.6.6**
  - 测试正常
- **CAENUpgrader-1.7.2**
  - Ubuntu20.04 异常

安装后在终端中输入

```
CAENUpgraderGUI
```

将会弹出 CAEN Upgrader GUI 的图形界面。

2.5 V1718

如果您使用 V1718，则需要安装 USB 驱动。

```
tar -xzvf CAENUSBdrvB-1.5.4.tgz
cd CAENUSBdrvB-1.5.4
make
make install      #需要ROOT权限
```

2.6 A2818 驱动

- **A2818Drv-1.20.tgz**
  - 测试正常
- **A2818Drv-1.21.tgz**
  - 测试正常
- **A2818Drv-1.22.tgz**
  - 在 Centos 7 下编译无法通过
- **A2818Drv-1.23.tgz**
  - CentOS7 测试正常，推荐使用
- **A2818Drv-1.24.tgz**
  - Ubuntu20 测试正常，推荐使用

如果您使用 A2818，则安装以下驱动。

```
# A2818Drv-1.20-build20161118.tgz
#将该文件夹复制到 /opt 并安装在该位置
tar -zxvf A2818Drv-1.20-build20161118.tgz
cp -r A2818Drv-1.20 /opt            #需要ROOT权限
cd /opt/A2818Drv-1.20               #需要ROOT权限
cp ./Makefile.2.6-3.x Makefile      #需要ROOT权限
make                                #需要ROOT权限

#设置开机自动执行该脚本
#在文件 /etc/rc.d/rc.local 中添加以下一行内容
/bin/sh /opt/A2818Drv-1.20/a2818_load
#或者在开启电脑之后执行以上命令
```

重启机箱后，在终端内输入 **dmesg|grep a2818** 将会看到以下的 A2818 驱动加载信息

```
a2818: CAEN A2818 CONET controller driver v1.20s
a2818:   Copyright 2004, CAEN SpA
pci 0000:05:02.0: enabling device (0000 -> 0003)
pci 0000:05:02.0: PCI INT A -> GSI 19 (level, low) -> IRQ 19
a2818: found A2818 adapter at iomem 0xf7800000 irq 0, PLX at 0xf7900000
a2818:   CAEN A2818 Loaded.
a2818:   CAEN A2818: 1 device(s) found.
```

Centos7

```
make -C /lib/modules/3.10.0-1160.el7.x86_64/build M=/opt/A2818Drv-1.23 LDDINCDIR=/
↪opt/A2818Drv-1.23/../include modules
make[1]: 进入目录“/usr/src/kernels/3.10.0-1160.el7.x86_64”
arch/x86/Makefile:166: *** CONFIG_RETPOLINE=y, but not supported by the compiler.␣
↪Compiler update recommended.。 停止。
make[1]: 离开目录“/usr/src/kernels/3.10.0-1160.el7.x86_64”
```

以上为错误发生时候的输出提示。

此时，用户可以修改 **/usr/src/kernels/3.10.0-1160.el7.x86_64/arch/x86/Makefile** 文件，通过注释以下代码来避免这个错误发生。

```
ifneq ($(RETPOLINE_CFLAGS),)
    KBUILD_CFLAGS += $(RETPOLINE_CFLAGS) -DRETPOLINE
else
    $(error CONFIG_RETPOLINE=y, but not supported by the compiler. Compiler update␣
↪recommended.)
endif
```

2.7 A3818 驱动

如果您使用 A3818，则安装以下驱动。安装该驱动时，电脑机箱必须插入 A3818 卡，否则将会报安装失败。

```
tar -zxvf A3818Drv-1.6.1.tgz
cd A3818Drv-1.6.1
make
make install        #需要ROOT权限
```

然后在终端内输入 dmesg 将会看到以下的 A3818 驱动加载信息

```
fuse init (API version 7.14)
CAEN A3818 PCI Express CONET2 controller driver v1.6.0s
  Copyright 2013, CAEN SpA
pci 0000:02:00.0: PCI INT A -> GSI 16 (level, low) -> IRQ 16
  alloc irq_desc for 33 on node -1
  alloc kstat_irqs on node -1
pci 0000:02:00.0: irq 33 for MSI/MSI-X
pci 0000:02:00.0: setting latency timer to 64
Found A3818 - Common BAR at iomem ffffc900067d4000 irq 0
Found A3818 with 1 link(s)
found A3818 Link 0 BAR at iomem ffffc900067d6000 irq 0
  CAEN A3818 Loaded.
  CAEN PCIe: 1 device(s) found.
```

2.8 RIKEN babirl

babirl 自动化安装方法

```
#在个人用户目录下安装理研 babirl 库
#在普通权限下执行以下脚本
sh autoinstallbabirl.sh
```

安装脚本会自动添加环境变量安装结束后查看.bashrc 文件，最后将多了三行如下内容

```
PATH=$PATH:/home/wuhongyi/babirl/bin/
export TARTSYS=/home/wuhongyi/VMEDAQ/anaroot
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$TARTSYS/lib:$TARTSYS/sources/Core
#其中 wuhongyi 为电脑当前用户名
```

```
#在ROOT权限下执行以下脚本
sh afterinstallbabirl.sh [user name]      #需要ROOT权限

#其中这里的 [user name] 换成你的帐号用户名，例如我的用户名为wuhongyi
# sh afterinstallbabirl.sh wuhongyi
```

2.9 初始化 babicon

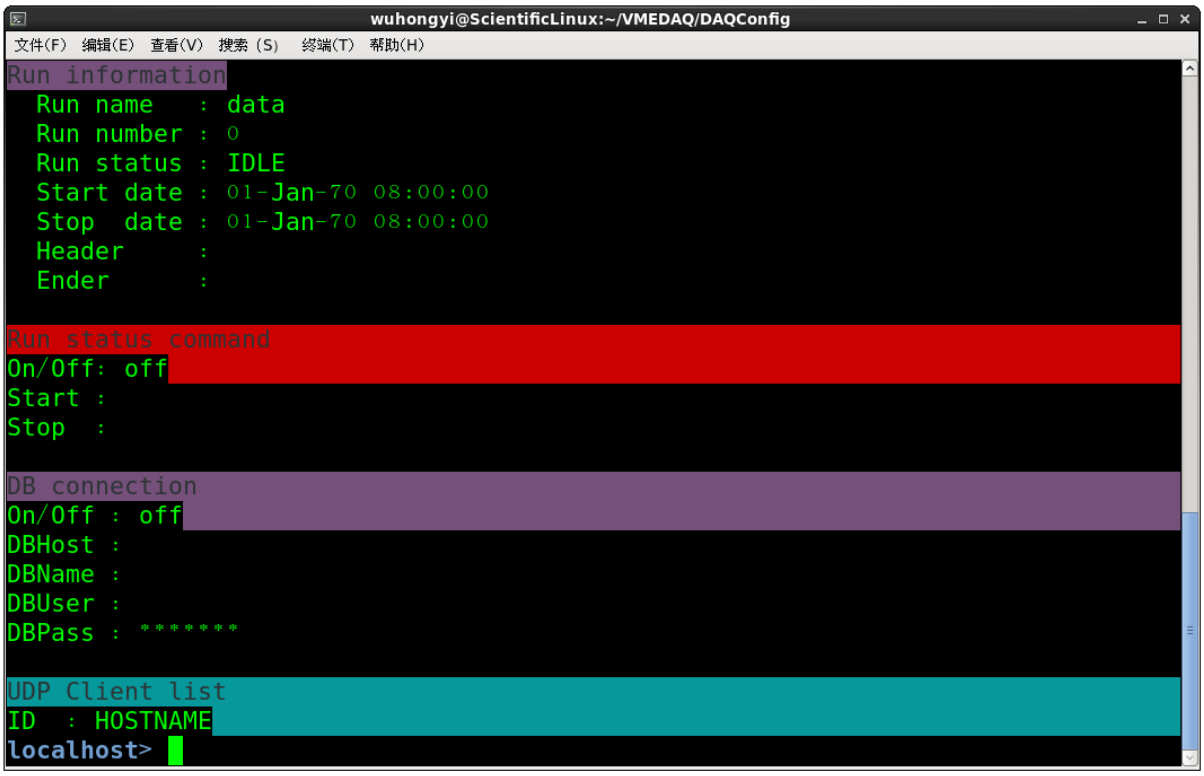
执行 DAQConfig 中的 StartDAQ.sh 开启进程

运行 babicon(安装后第一次需输入以下初始化)

新打开一个终端，然后输入

```
babicon
```

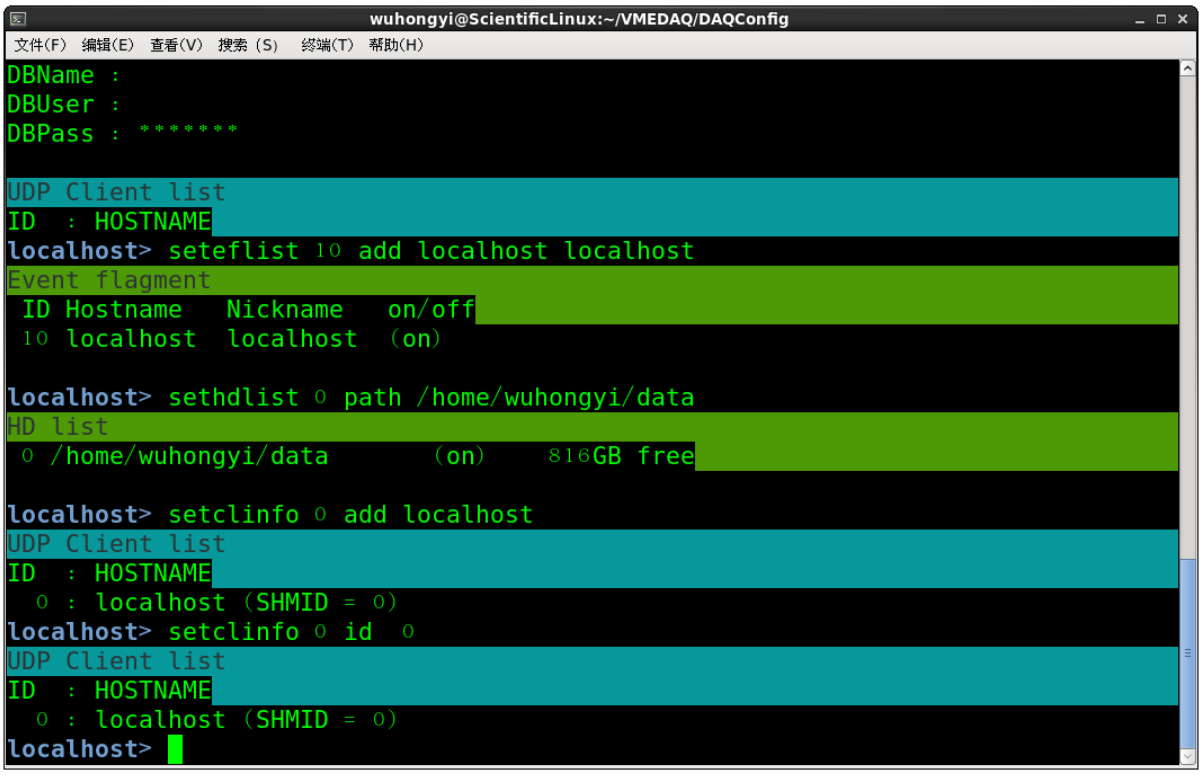
回车之后将看到以下界面



以下进行基本的变量设置

```
seteflist 10 add localhost localhost
sethdlist 0 path /home/wuhongyi/data     #这里为数据存储路径
setclinfo 0 add localhost  #localhost为本机器
setclinfo 0 id  0

#如果设置给远程电脑
setclinfo 0 add [ip]    #[ip] 为接收端电脑IP
setclinfo 0 id  0
```



2.10 防火墙设置

将共享数据发送到 Online 电脑，需要做以下设置或者关闭防火墙

对 Scientific Linux 6，终端 ROOT 权限下输入 **setup** ，选择 **防火墙配置**，去掉 **启用**。对 cientific Linux 7，ROOT 权限下终端输入以下信息关闭 firewall

```
systemctl stop firewalld.service #停止firewall
systemctl disable firewalld.service #禁止firewall开机启动
firewall-cmd --state #查看默认防火墙状态（关闭后显示notrunning，开启后显示running）
```

如果机器不联网，可以不需要开启以下 iptables 防火墙，反正不会被黑

```
#在 /etc/sysconfig/iptables␣
↪添加以下一行（不能放到最后一行，其中IP替换为发送DAQ电脑的IP）
-A INPUT -p udp -m state --state NEW -m udp --dport 17500:17510 -s 222.29.111.201 -
↪j ACCEPT
```

之后在 ROOT 权限下执行以下命令

```
systemctl restart iptables.service #最后重启防火墙使配置生效
systemctl enable iptables.service #设置防火墙开机启动
```

CHAPTER 3

固件要求

注意

- 请确保所使用的所有插件固件版本与以下一致。
- 我们尽可能及时更新保证采用较新/最新的固件。
- 由于新版本软件/固件我们需要经过大量评估测试，用户请不要随意升级我们未推荐的版本。

3.1 当前固件版本

```
V1718   2.18
V2718 FW CONET2 Compliant    2.18_1.5
  V2718 2.18
  A2719 1.5

A2818  新版的CONET2 1.0  旧版的CONET1 0.8
A3818   0.5
        0.6 未测试

v1190   1.1
MADC32  0224
```

3.2 查看固件版本

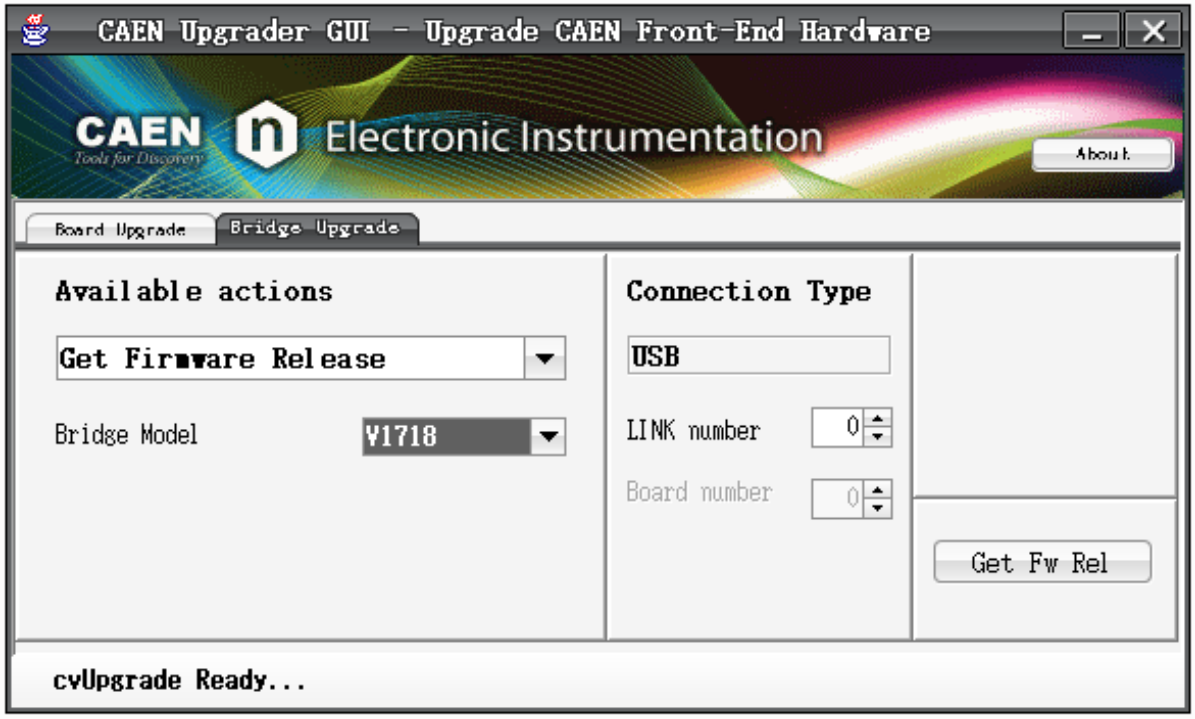
V1718/V2718/A2818/A3818 查看固件版本采用 CAENUpgraderGUI 程序，V1718/V2718/A2818/A3818/V1x90 升级固件版本同样采用 CAENUpgraderGUI 程序。即在终端中执行

```
CAENUpgraderGUI
```

升级固件时候，Browse 选择固件之后会弹出一个警告窗口，提示你 “You have chosen to use a raw binary file”，点击确认，然后点击右下角的 Upgrade。等待升级结束，将会有一个窗口提示你重启。

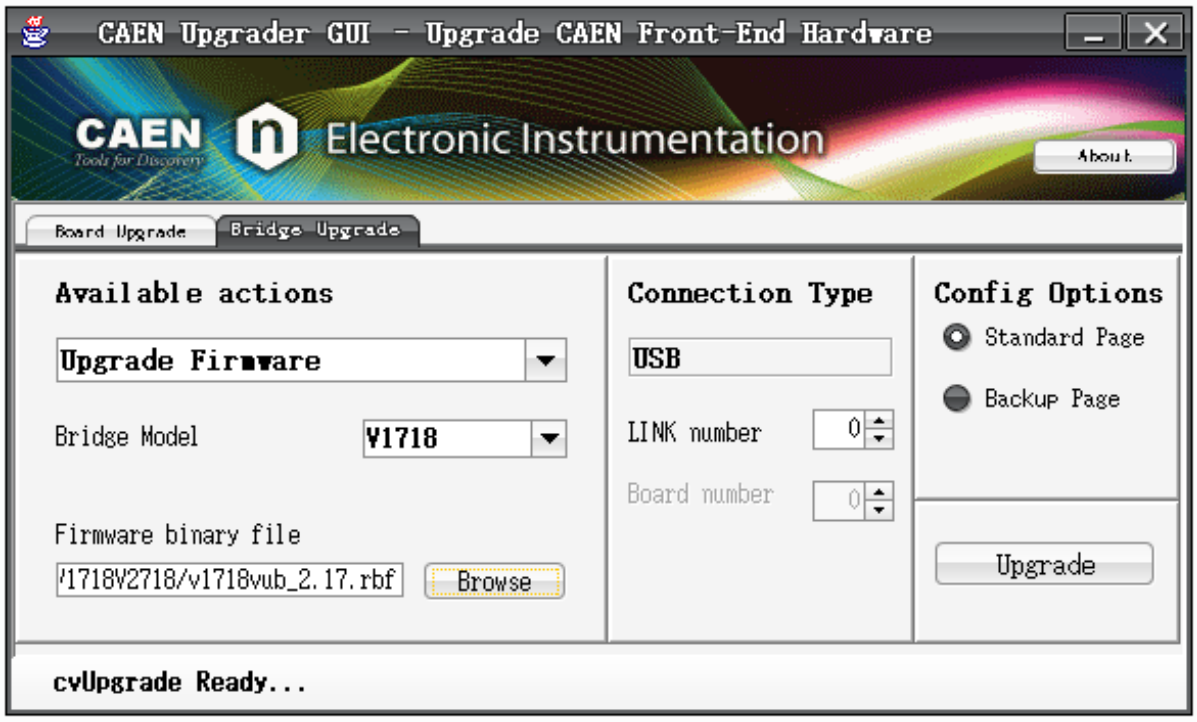
3.2.1 V1718

如下图，查看 V1718 的固件版本，点击 *Get Fw Rel* 按钮。



如果该固件版本不是 **当前固件版本**所列版本，则升级固件。

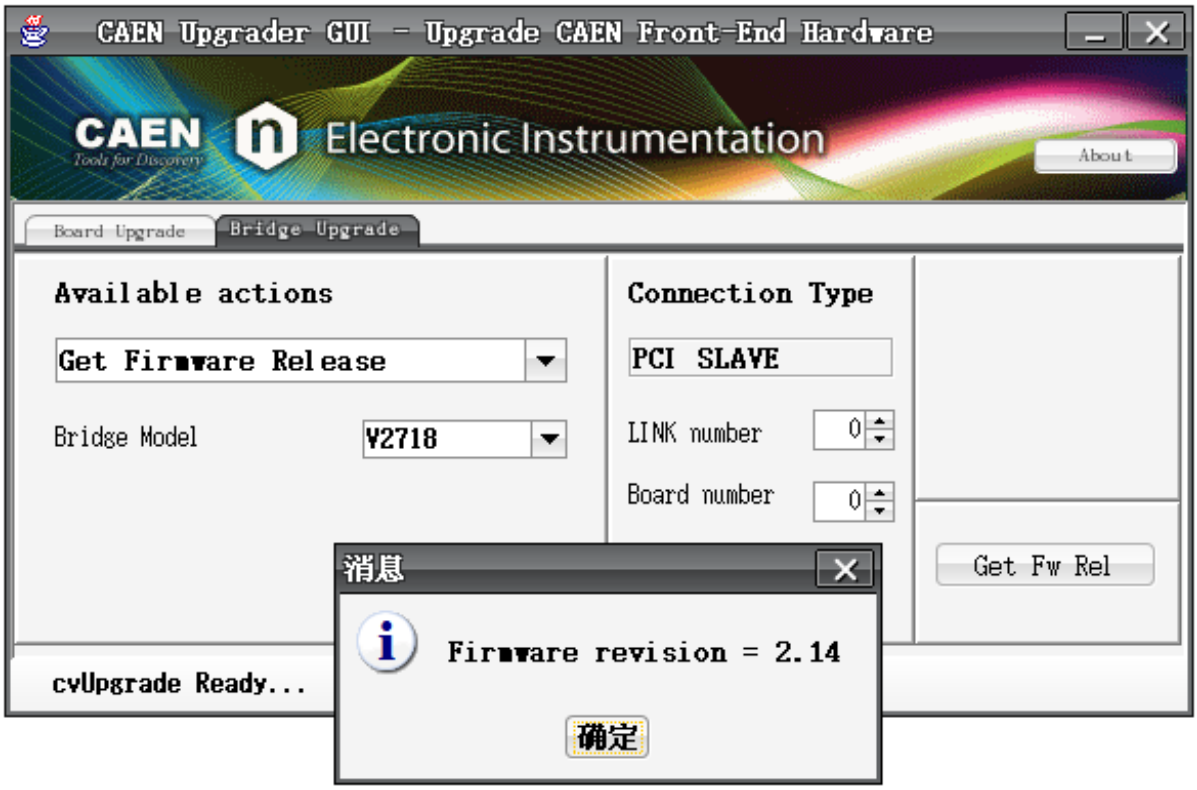
升级界面如下图所示：



3.2.2 V2718

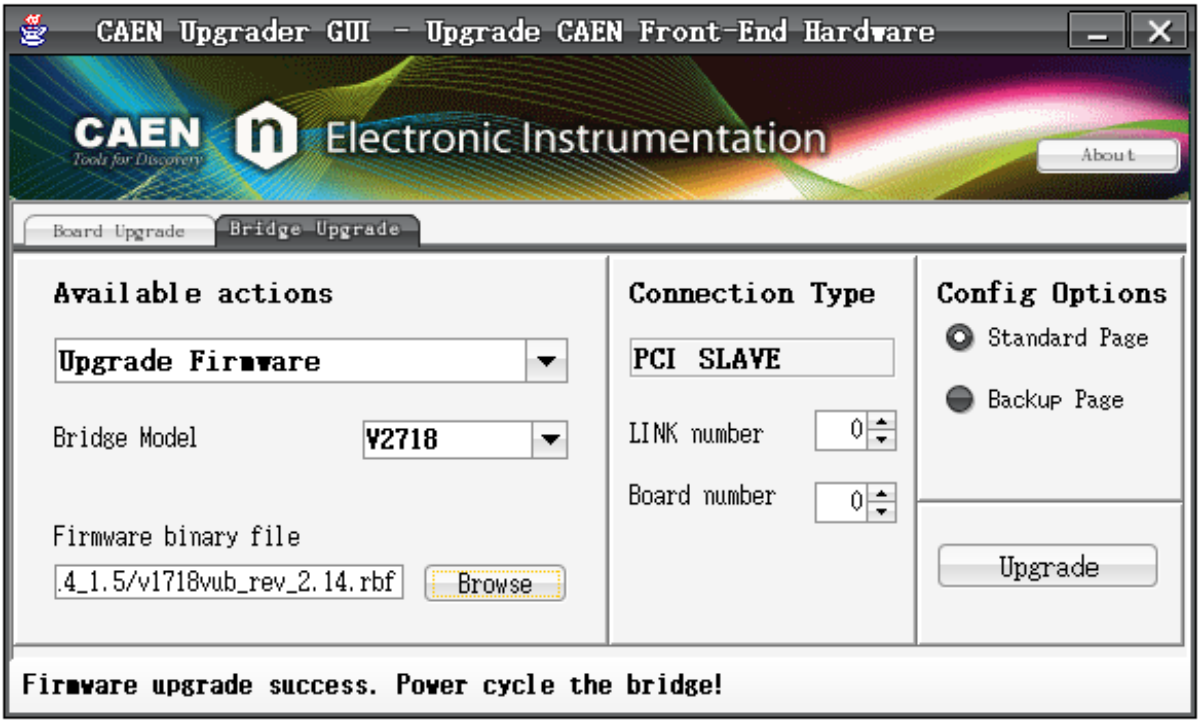
V2718 上固件包括主板 V2718 及子板上的 A2719。

如下图，查看 V2718 主板的固件版本，点击 *Get Fw Rel* 按钮。

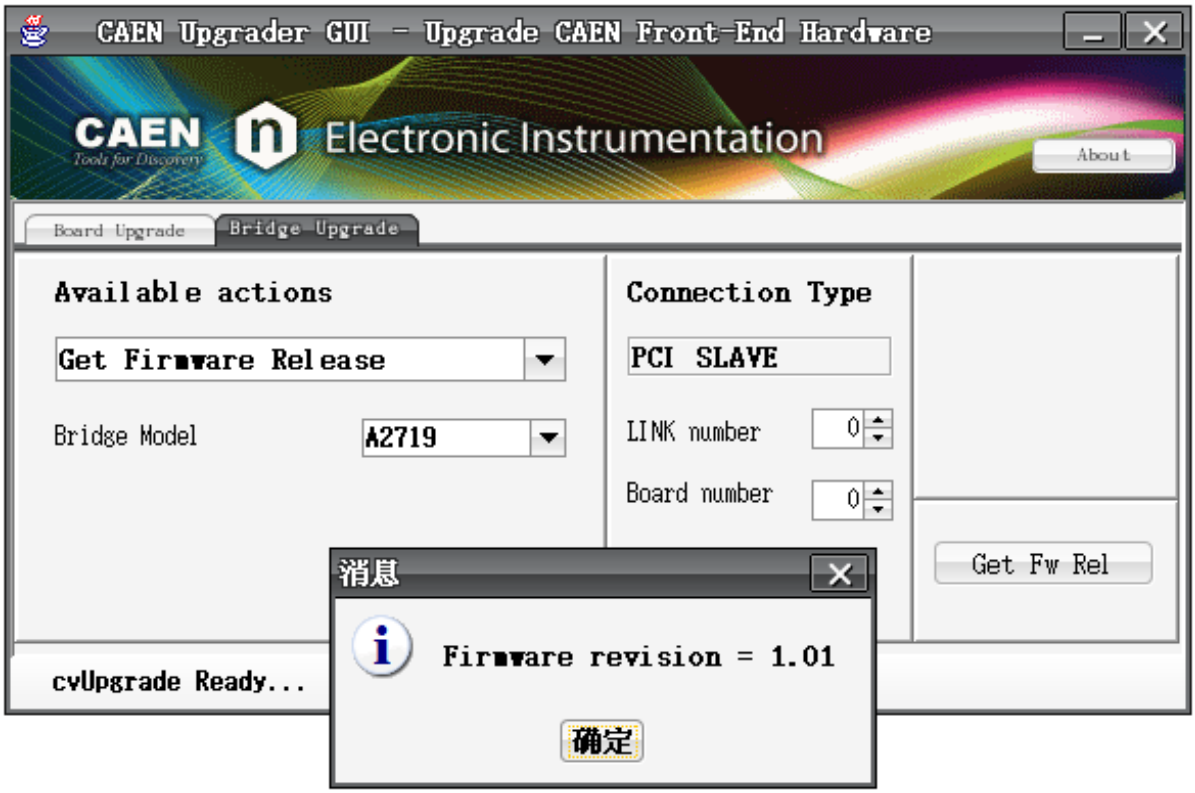


如果该固件版本不是 **当前固件版本**所列版本，则升级固件。

升级界面如下图所示：



如下图，查看子板 A2719 的固件版本，点击 *Get Fw Rel* 按钮。

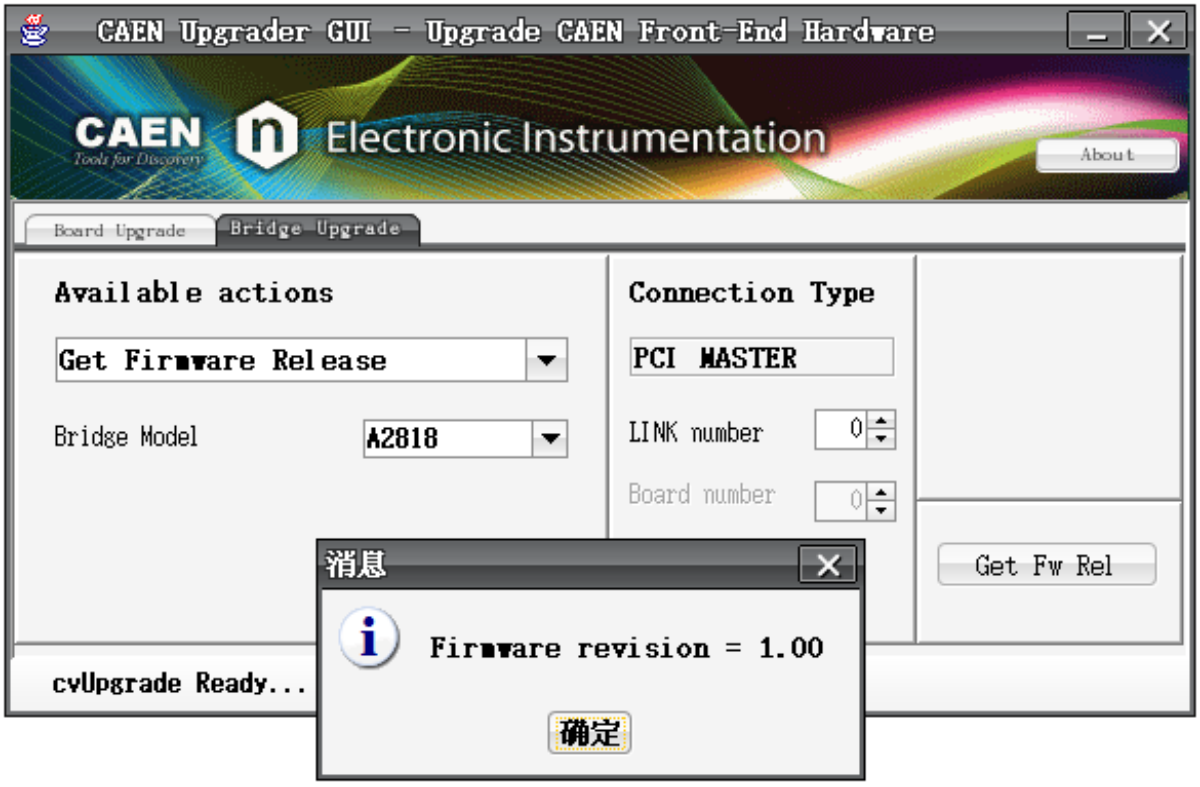


如果该固件版本不是 **当前固件版本**所列版本，则升级固件。

升级界面如下图所示：

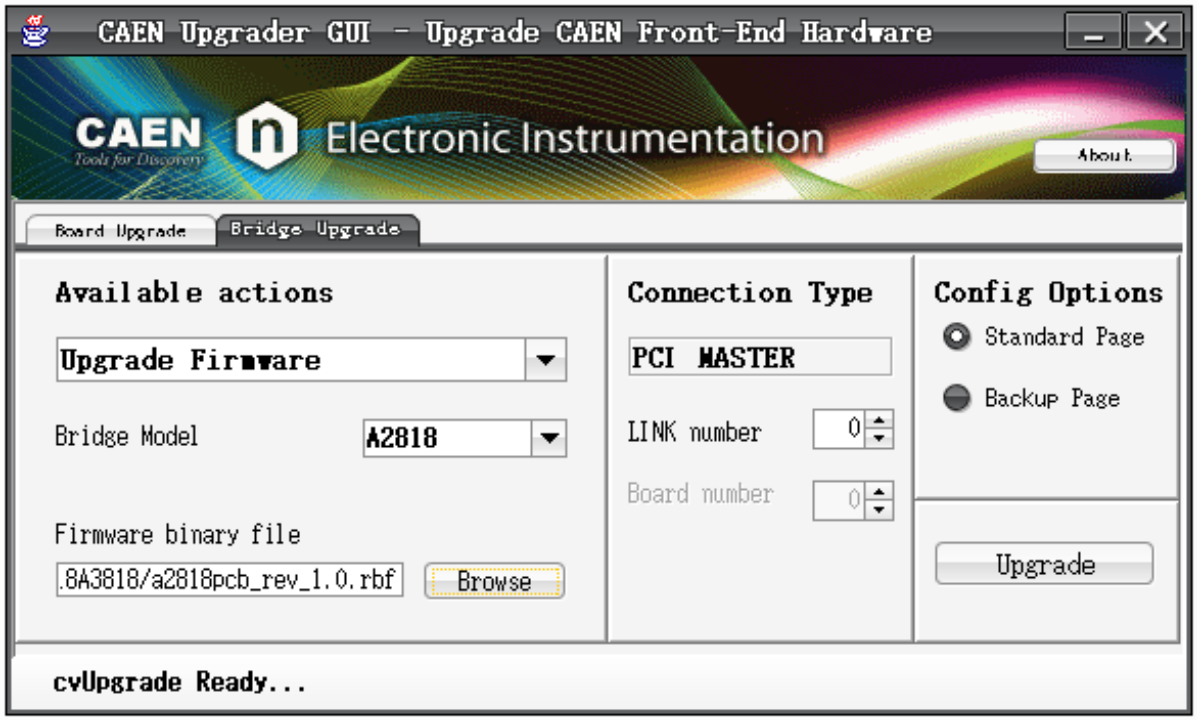
3.2.3 A2818

如下图，查看 A2818 的固件版本，点击 *Get Fw Rel* 按钮。



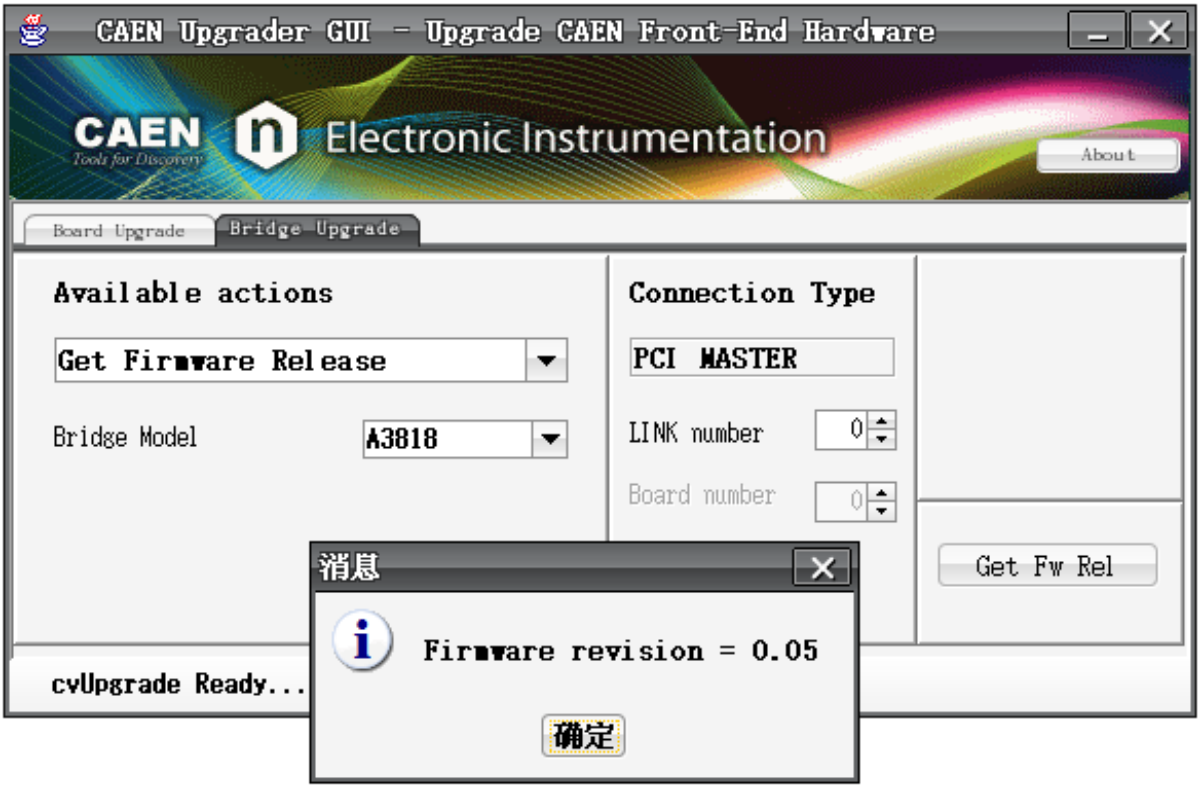
如果该固件版本不是 **当前固件版本**所列版本，则升级固件。

升级界面如下图所示：



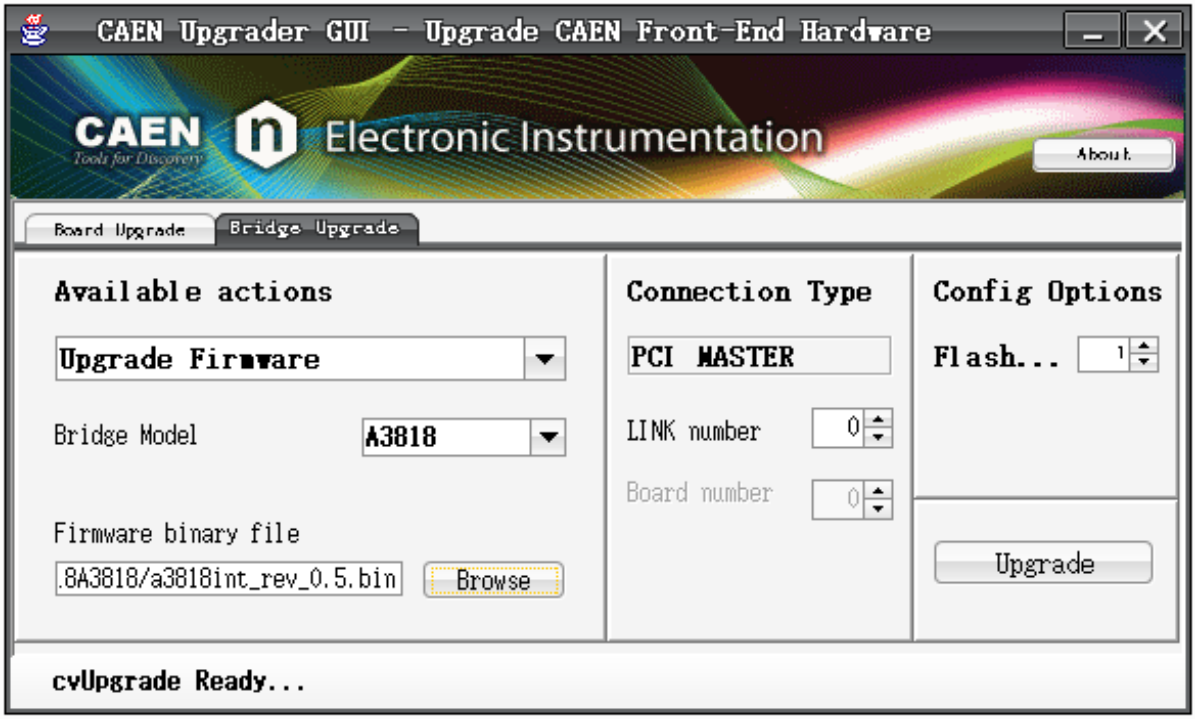
3.2.4 A3818

如下图，查看 A3818 的固件版本，点击 *Get Fw Rel* 按钮。



如果该固件版本不是 **当前固件版本**所列版本，则升级固件。

升级界面如下图所示：



3.2.5 V1x90

- **V1190/V1290**
  - Firmware Revision Register(Base Address + 0x1026, read only, D16)
  - This register contains the firmware revision number coded on 8 bit.

待补充

3.2.6 MADC32

- **madc32**
  - 0x600E firmware_revision

待补充

CHAPTER 4

获取配置

4.1 模块安放顺序

为了方便用户配置 DAQ，这里我们建议用户按照以下顺序依次插入采集模块（如果没有某些类型模块，则调过相应类型的模块）：

- 控制器 V1718/V2718
- 定标器 V830
- V7xx
- V1x90
- MADC32

4.2 程序修改建议顺序

- anaroot/CBLT.hh
- DAQConfig/StartDAQ.sh
- DAQConfig/StopDAQ.sh
- DAQConfig/bbcaenvme/babies/bbmodules.h
- DAQConfig/bbcaenvme/babies/start.c
- DAQConfig/bbcaenvme/babies/evt.c
- DAQConfig/bbcaenvme/babies/clear.c
- DAQConfig/bbcaenvme/babies/stop.c
- DAQConfig/bbcaenvme/init/daqinitrc.sh

修改程序，请先仔细阅读 DAQConfig 页面中的说明。

4.3 V1718/V2718

V1718/V2718 PCB 板上 DIP 开关：Prog: 0 off, 1 off, 2 off, 3 on, 4 off, I/O NIM

v1718/V2718 前面板 5 个 LEMO 输出口，分别为 0-4

通电时候输出口 0-3 处于高电平，输出口 4 处于低电平。因此软件 BUSY 模式时候采用输出口 4，硬件 BUSY 模式采用输出口 3。

4.4 门信号连接

待补充

4.5 软件 BUSY 模式

待补充

4.6 硬件 BUSY 模式

待补充

CHAPTER 5

analysis

存放辅助分析程序，当前只放置一个 MakeProcess 模板。

CHAPTER 6

anaroot

如果采用 CBLT 模式读取数据，则先修改 *CBLT.hh* 文件，不采用 CBLT 模式则不用修改。设置好之后，执行该目录下的自动编译、安装脚本 *autoPKU.sh* 即可

```
sh autoPKU.sh
```

修改 **CBLT.hh** 文件，其中设置应该与 CBLT 模式下的插件设置顺序一致。

当前 CBLT chain 支持 v830、v7xx、v1190、v1290、madc 五种类型的插件，如下所示：

```
#define v830m
#define v7xxm
#define v1190m
#define v1290m
#define madcm
```

获取中如果没有哪一种类型插件，则需注释掉该类型的定义！！！

以下 *xxxn* 为启用插件的数据顺序，从 0 开始编码，如果五种类型插件都有，则为以下设置：

```
#define v830n 0
#define v7xxn 1
#define v1190n 2
#define v1290n 3
#define madcn 4
```

如果只含有 v7xx、madc 两种类型的插件，则定义如下：

```
#define v7xxn 0
#define madcn 1
```

如果只含有 v830、v7xx、madc 三种类型的插件，则定义如下：

```
#define v830n 0
#define v7xxn 1
#define madcn 2
```

以下定义用来指定每种类型插件的个数

```
#define v830num
#define v7xxnum
#define v1190num
#define v1290num
#define madcnum
```

以下是 v830 的其它设置

```
#define v830chn 8 // 这里设置830开启路数
#define v830head 1 // 不要修改
#define v830geo 0 // 不要修改
```

CHAPTER 7

checkcnt

用来辅助检查文件中事件是否关联。执行程序之后将会在该文件夹内生成一个 pdf 文件，检查该文件内每张图数值是否有异常。

CHAPTER 8

cutpedestal

用来辅助设置 pedestal 数值。高斯拟合 pedestal，并给出三倍 sigma 的上限作为推荐数值，并生成初始文件夹 init 内脚本。

CHAPTER 9

DAQConfig

首先修改 **StartDAQ.sh/StopDAQ.sh** 两个文件，将文件内的 *wuhongyi* 替换成当前 LINUX 的用户名。然后修改 bbcaenvme 文件夹下 babies、init 文件夹内文件。

修改文件之前，我们需要先理解硬件地址与 GEO 编号。每个插件模块侧面都有四个拨盘，每个拨盘代表一个 16 进制位，例如，当四个拨盘从左到右分别为 1，2，3，4 时，表示其硬件地址为 0x1234。控制器与模块的通讯依靠该硬件地址来寻址，每个控制器下的模块地址应该具有唯一性。同时，我们可以通过软件对每个模块设置一个 GEO 编号，该模块输出数据中都会带有该 GEO 标记，方便我们对数据进行解码。GEO 编号范围为 0-31。

这里我们对硬件地址设置进行如下约定（当然以下约定不是强制要求，用户可以任意修改），

- v7xx 模块硬件地址从 0x1000 开始，然后 0x1001，有多少个模块就依次往后设置。
- MADC32 硬件地址从 0x2000 开始，然后 0x2001，有多少个模块就依次往后设置。
- v1x90 硬件地址从 0x4000 开始，然后 0x4001，有多少个模块就依次往后设置。
- v830 模块硬件地址从 0x5000 开始，然后 0x5001，有多少个模块就依次往后设置。

这里我们对 GEO 地址设置进行如下约定（当然以下约定不是强制要求，用户可以任意修改），

- 一般 v830 使用 1-2 个模块而已，因此 GEO 编号 30/31 我们预留给 v830。第一个 v830 的 GEO 为 30，第二个 v830 的 GEO 为 31。
- 实验中一般 v7xx 和 MADC32 模块使用较多，如果每种模块均不超过 10 个的话，我们默认 0-9 预留给 v7xx，10-19 预留给 MADC32。如果某种模块超过 10 个话，那么 v7xx 和 MADC32 的 GEO 按照 0-19 的编号依次往下进行分配。
- 实验中 v1x90 模块的使用数量一般不会超过 5 个，这里我们为 v1x90 预留 20-24，编号从 20 开始依次往后进行分配。
- 剩余的 GEO 编号 25-29 进行机动使用。

整个 DAQ 的程序配置中，需要在文件 babies/bbmodules.h 中进行硬件地址设置。然后还需对文件夹 init 内的文件进行硬件地址和 GEO 的设置。

9.1 babies/bbmodules.h

修改 ADCADDR、MADCADDR、V1x90ADDR、SCAADDR 使之与硬件地址匹配（可以多余设置，不可少设置）。其它不要修改。

这里我们按照之前的约定，V7xx 硬件地址从 0x1000 开始编号，用 ADC[x]ADDR 来表示不同模块。MADC32 硬件地址从 0x2000 开始编号，用 MADC[x]ADDR 来表示不同模块。V1x90 硬件地址从 0x4000 开始编号，用 V1x90ADDR[x] 来表示不同模块。V830 硬件地址从 0x5000 开始编号，用 SCAADDR[x] 来表示不同模块。（其中 [x] 代表不同的数字）

如果您依照我们的约定来设置，则不需要修改本文件。

如果您使用控制器 V1718，则需要注释以下代码。否则开启以下代码

```
//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......
// 以下部分用户需要修改

#define V2718   //如果使用 V1718 则注释本行

// 以上部分用户需要修改
//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......
```

如果您使用软件 busy 模式时，则开启以下代码行，如果您使用硬件 busy 模式时，则注释掉以下行代码。

```
//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......
// 以下部分用户需要修改

#define SOFTWAREBUSY

// 以上部分用户需要修改
//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......
```

如果您使用常规的 busy 模式，且需要多个机箱同步运行，则开启以下代码，否则注释掉

```
//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......
// 以下部分用户需要修改

#define SOFTWAREBUSYMULTICRATE

// 以上部分用户需要修改
//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......
```

选择合适的插件类型作为 INTERRUPT 源，以下 4 行代码仅能开启一个。选择的依据是系统中的模块在一个事件中，谁最后完成数据的 MEB 写入，这样能够避免有些模块已经开始读取数据，而一些模块数据还未转换结束。当有 v7xx 模块时，默认选择该类型模块，如果系统有 MADC32 时，需要考虑读取它之前是否完成数据的转换。

```
//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......
// 以下部分用户需要修改

#define V7XXINTERRUPT
// #define SCAINTERRUPT
// #define MADC32INTERRUPT
// #define V1X90INTERRUPT

// 以上部分用户需要修改
//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......
```

9.2 babies/start.c

根据文件内提示设置，有该类型插件则开启对应代码。其它不要修改。

V830

```
//....ooo0000ooo........ooo0000ooo........ooo0000ooo........ooo0000ooo......
// 以下部分用户需要修改

// 有 V830 插件
v830_clear_all(SCAADDR0);

// 以上部分用户需要修改
//....ooo0000ooo........ooo0000ooo........ooo0000ooo........ooo0000ooo......
```

用户需要修改以上代码段，如果您不使用 V830 模块，则注释掉以上区域的代码。

如果您使用一个 V830 模块，则添加代码：

```
v830_clear_all(SCAADDR0);
```

如果您使用两个 V830 模块，则添加代码：

```
v830_clear_all(SCAADDR0);
v830_clear_all(SCAADDR1);
```

V7xx

```
//....ooo0000ooo........ooo0000ooo........ooo0000ooo........ooo0000ooo......
// 以下部分用户需要修改

// 有 V7xx 插件
// 每个插件单独设置
v7xx_rst_counter(ADC0ADDR);
v7xx_rst_counter(ADC1ADDR);
v7xx_rst_counter(ADC2ADDR);
// v7xx_rst_counter(ADC3ADDR);
// v7xx_rst_counter(ADC4ADDR);
// v7xx_rst_counter(ADC5ADDR);
// v7xx_rst_counter(ADC6ADDR);
// v7xx_rst_counter(ADC7ADDR);

v7xx_clear(ADC0ADDR);
v7xx_clear(ADC1ADDR);
v7xx_clear(ADC2ADDR);
// v7xx_clear(ADC3ADDR);
// v7xx_clear(ADC4ADDR);
// v7xx_clear(ADC5ADDR);
// v7xx_clear(ADC6ADDR);
// v7xx_clear(ADC7ADDR);

// 以上部分用户需要修改
//....ooo0000ooo........ooo0000ooo........ooo0000ooo........ooo0000ooo......
```

用户需要修改以上代码段，如果您不使用 V7xx 模块，则注释掉以上区域的代码。

如果您使用一个 V7xx 模块，则添加代码：

```
v7xx_rst_counter(ADC0ADDR);
v7xx_clear(ADC0ADDR);
```

如果您使用两个 V7xx 模块，则添加代码：

```
v7xx_rst_counter(ADC0ADDR);
v7xx_rst_counter(ADC1ADDR);
v7xx_clear(ADC0ADDR);
v7xx_clear(ADC1ADDR);
```

使用更多 V7xx 则依次类推。

V1x90

```
//....ooo0000ooo........ooo0000ooo........ooo0000ooo........ooo0000ooo......
// 以下部分用户需要修改

// 有 V1190/V1290 插件
// 每个插件单独clear
// v1190_clear(V1x90ADDR0);
// v1290_clear(V1x90ADDR1);

v1190_clear(V1x90ADDR0);
v1190_clear(V1x90ADDR1);
// v1290_clear(V1x90ADDR0);
// v1290_clear(V1x90ADDR1);

// 以上部分用户需要修改
//....ooo0000ooo........ooo0000ooo........ooo0000ooo........ooo0000ooo......
```

用户需要修改以上代码段，如果您不使用 V1x90 模块，则注释掉以上区域的代码。

如果您只使用一个 V1190 模块，则添加代码：

```
v1190_clear(V1x90ADDR0);
```

如果您只使用两个 V1190 模块，则添加代码：

```
v1190_clear(V1x90ADDR0);
v1190_clear(V1x90ADDR1);
```

如果您使用一个 V1190，一个 V1290，则添加代码：

```
v1190_clear(V1x90ADDR0);
v1290_clear(V1x90ADDR1);
```

更多模块使用的组合，请以此类推。

MADC32

```
//....ooo0000ooo........ooo0000ooo........ooo0000ooo........ooo0000ooo......
// 以下部分用户需要修改

// 有 MADC32 插件
madc32_mclear(MSTMDCADDR);
madc32_mirq_level(MSTMDCADDR,0);
madc32_mreset_ctra_counters(MSTMDCADDR);
madc32_mfifo_reset(MSTMDCADDR);
madc32_mstart_acq(MSTMDCADDR);

// 以上部分用户需要修改
//....ooo0000ooo........ooo0000ooo........ooo0000ooo........ooo0000ooo......
```

用户需要修改以上代码段，如果您不使用 MADC32 模块，则注释掉以上区域的代码。如果您使用了 MADC32 模块，不管使用了多少个模块，只需要开启以上代码即可对所有的模块完成初始化。

9.3 babies/evt.c

根据文件内提示设置。其它不要修改。

```
//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......
// 以下部分用户需要修改

// 软件BUSY模式下6036->0x1不需要以下清除，6036->0x3需要以下清除，6036->
↪0x0需要以下清除
// 硬件BUSY模式下只能采用6036->0x3，需要以下清除

// 有 MADC32 插件
madc32_mclear(MSTMDCADDR);

// 以上部分用户需要修改
//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......
```

用户需要修改以上代码段，如果您不使用 MADC32 模块，则注释掉以上区域的代码。如果您使用了 MADC32 模块，不管使用了多少个模块，只需要开启以上代码即可对所有的模块完成清除。

当然，在软件 busy 模式下，对每个模块的寄存器进行相应的寄存器配置，可以不用以上清除指令自动进行清除，此时每个事件能够节约 20 us 左右的时间，该方案建议对 DAQ 比较熟悉的用户使用。

当系统中有两个及以上模块时。采用 CBLT 方式读取数据，这是最高效的数据读取方式。但是当系统中只有一个模块时，则注释掉 CBLT 的数据读取，开启对应插件的读取。

```
//....oooOOOOOooo........oooOOOOOooo........oooOOOOOooo........oooOOOOOooo......
// 以下部分用户需要修改

// Read data
babies_init_segment(MKSEGID(0, 0, PLAQ, CBLT));
#ifdef V7XXINTERRUPT
v7xx_dmasegdata(MSTTDCADDR, 4000);
#endif
#ifdef SCAINTERRUPT
usleep(10);
v830_dmasegdata(MSTTDCADDR, 4000);
#endif
#ifdef V1X90INTERRUPT
v1190_dmasegdata(MSTTDCADDR, 4000);
#endif
#ifdef MADC32INTERRUPT
madc32_dmasegdata(MSTTDCADDR, 4000);
#endif
babies_end_segment();


// 插件
// babies_init_segment(MKSEGID(0, 0, PLAQ, V785));
// v7xx_dmasegdata(ADC0ADDR, 34);
// babies_end_segment();

// babies_init_segment(MKSEGID(0, 0, PLAQ, V830));
// v830_dmasegdata(SCAADDR0, 34);
// babies_end_segment();

// babies_init_segment(MKSEGID(0, 0, PLAQ, V1190));//V1290
// v1190_dmasegdata(V1x90ADDR0, 1000);
// babies_end_segment();

// babies_init_segment(MKSEGID(0, 0, PLAQ, MADC));
// madc32_dmasegdata(MADC0ADDR, 34);
```

(下页继续)

(续上页)

```
// babies_end_segment();

// 以上部分用户需要修改
//....ooo0000ooo........ooo0000ooo........ooo0000ooo........ooo0000ooo......
```

9.4 babies/stop.c

根据文件内提示设置，有该类型插件则开启对应代码，开启对应类型 busy 代码。其它不要修改。

MADC32

```
//....ooo0000ooo........ooo0000ooo........ooo0000ooo........ooo0000ooo......
// 以下部分用户需要修改

// 有 MADC32 插件
madc32_mstop_acq(MSTMDCADDR);

// 以上部分用户需要修改
//....ooo0000ooo........ooo0000ooo........ooo0000ooo........ooo0000ooo......
```

用户需要修改以上代码段，如果您不使用 MADC32 模块，则注释掉以上区域的代码。如果您使用了 MADC32 模块，不管使用了多少个模块，只需要开启以上代码即可对所有的模块发送结束采集指令。

9.5 cmdvme/cmdvme.c

如果使用控制器 V1718，则需要修改文件中以下代码。将 *V2718* 改为 *V1718* 即可。

```
enum board bd = V2718;
```

如果编译中出现以下类似的错误，

```
cmdvme.c:29:8: error: initialization of flexible array member in a nested context
29 |   {WR, "-wr"},
   |        ^~~~
cmdvme.c:29:8: note: (near initialization for ‘chkmode[0]’)
```

则代码修改如下

```
typedef struct {
  enum mode md;
  const char mdchar[3];//指定数组长度
}stchkmode;
```

9.6 init/daqinitrc.sh

修改该文件内对应脚本，使之与获取插件对应，用来初始化插件。

重点是修改 cblt.hh 文件，对启用的插件设置 CBLT ADDR 为 0xbb，其中 MADC 还得设置 MCST ADDR 为 0xdd。还得设置每一个插件在 CBLT 中的顺序，first、mid、last。至少得两个插件才能组成 CBLT.

init/daqinitrc.sh 文件包含以下内容：

```
#!/bin/sh

/bin/sh ./v830.sh
/bin/sh ./v7xx_all.sh
# /bin/sh ./v7xx_thres.sh
/bin/sh ./v1190_0.sh
/bin/sh ./v1190_1.sh
# /bin/sh ./v1290.sh
/bin/sh ./madcall.sh
# /bin/sh ./madc_thres.sh
/bin/sh ./cblt.sh
```

如果您使用了 V830 则开启以下代码，否则注释掉以下代码：

```
/bin/sh ./v830.sh
```

待补充

CHAPTER 10

httponline

基于网页的在线监视。

仅仅需要修改插件定义即可，无需修改其它代码。

修改文件 **UserDefine.hh** ，按照提示修改即可。

```
// V7XX
#define V7XX_CRATE //没有该插件则注释本行
#define V7XX_CRATE_NUM 3  //插件数
const int V7XX_CRATE_GEO[V7XX_CRATE_NUM] = {0,1,2};
#define V7XX_HIST_BIN 4200
#define V7XX_HIST_MIN 0
#define V7XX_HIST_MAX 4200

// MADC
#define MADC_CRATE //没有该插件则注释本行
#define MADC_CRATE_NUM 2  //插件数
const int MADC_CRATE_GEO[MADC_CRATE_NUM] = {10,11};
#define MADC_HIST_BIN 2000
#define MADC_HIST_MIN 0
#define MADC_HIST_MAX 8000

// V1190
#define V1190_CRATE //没有该插件则注释本行
#define V1190_CRATE_NUM 1  //插件数
const int V1190_CRATE_GEO[V1190_CRATE_NUM] = {20};
#define V1190_HIST_BIN 5000
#define V1190_HIST_MIN 0
#define V1190_HIST_MAX 50000

// V1290
#define V1290_CRATE //没有该插件则注释本行
#define V1290_CRATE_NUM 1  //插件数
const int V1290_CRATE_GEO[V1290_CRATE_NUM] = {21};
#define V1290_HIST_BIN 5000
#define V1290_HIST_MIN 0
#define V1290_HIST_MAX 500000
```

修改文件 **UserDefine.hh** 之后，按照以下指令进行编译

```
make clean
make
# 编译成功会在目录中生成可执行文件 online

# 开启 http online 后台进程
./online
# 默认端口 8888，可通过 ip:8888 访问
# 访问网页需要用户名/密码，其中 用户名 admin 密码 admin，拥有最高权限；用户名
↪guest 没有密码。仅可以查看
```

CHAPTER 11

online

时时监视每路信号的能量信息。连续按两次“ctrl-C”即可退出该程序。
按照提示修改 Online.cc 文件

CHAPTER 12

r2root

仅仅需要修改插件定义即可，无需修改其它代码。

修改文件 **UserDefine.hh** ，按照提示修改即可。

```
#define ROOTFILEPATH "/home/wuhongyi/data/rootfile/"
#define RAWFILEPATH "/home/wuhongyi/data/"
#define RAWFILENAME "data"

// RAWFILEPATH 原始数据路径
// RAWFILENAME 原始数据文件名
// ROOTFILEPATH 生成的 ROOT 文件的存放位置
```

```
// scaler
#define v830_r2root //没有该插件则注释本行
#define v830num 1
const int v830geo[v830num] = {30};//元素个数必须与 v830num 一致


// adc
#define v785_r2root //没有该插件则注释本行
#define v785num 1
const int v785geo[v785num] = {0};//元素个数必须与 v785num 一致


// qdc
#define v792_r2root //没有该插件则注释本行
#define v792num 1
const int v792geo[v792num] = {1};//元素个数必须与 v792num 一致


// tdc
#define v775_r2root //没有该插件则注释本行
#define v775num 1
const int v775geo[v775num] = {2};//元素个数必须与 v775num 一致


// gdc
#define v1x90_r2root //没有该插件则注释本行
#define v1x90num 2
```

(下页继续)

(续上页)

```
#define v1x90multi 5 //gdc 数组第三维度
#define v1x90hitmax 3000
const int v1x90geo[v1x90num] = {20,21};//元素个数必须与 v1x90num 一致


// madc
#define madc32_r2root //没有该插件则注释本行
#define madc32num 2
const int madc32geo[madc32num] = {10,11};//元素个数必须与 madc32num 一致
```

修改文件 **UserDefine.hh** 之后，按照以下指令进行编译

```
make clean
make
# 编译成功会在目录中生成可执行文件 convert

# 数据转换指令
./convert [run number]
# 其中 [run number] 为要转换的文件编号，例如转换编号为 652 的文件命令如下
./convert 652
```

CHAPTER 13

statistics

用来监视每路的计数率。

CHAPTER 14

基于可编程逻辑模块的触发系统

本固件对硬件的要求为：使用三个 8 通道 LEMO 子板。所有通道配置为 NIM 信号输出。

V2748 的最后两个 LEMO 输出口连接到可编程逻辑模块上的 G(0) 和 G(1) 口。A 口的 32 输入通道用于触发信号的输入；B 口的 32 输入通道用于采集模块 busy 信号的输入。

前两个子板共 16 个通道输出为采集系统的 GATE。第三个子板的前 4 个通道输出延迟 GATE 来驱动 V1x90，后 4 个通道为监视输出，通过连接到示波器来观察信号（大多数示波器只有 4 个通道，因此我们的监视器设置默认设置为 4 个通道。通过修改控制寄存器来切换监视信号）。

CHAPTER 15

网页控制界面的安装

15.1 PLULib 驱动安装

PLULib 依赖 gcc 5.0 及以上的版本。CAEN 给我们提供了驱动源代码，因此使用我们的程序包则不需要安装以下 PLULib 驱动。

```
### 使用 吴鸿毅 提供的网页控制程序，请不要安装以下驱动！！！！！！
tar -zxvf CAEN_PLULib-1.1.0-Build20190924.tgz
cd CAEN_PLU-1.1
sh install_x64
cd ..
```

15.2 CentOS 7

15.2.1 apache

CentOS 7(Scientific Linux 7) 配置文件路径

```
/etc/httpd/conf/httpd.conf
```

selinux 关闭

```
getenforce # 查看selinux状态
setenforce 0 # 临时关闭
修改/etc/selinux/config 文件，将SELINUX=enforcing改为SELINUX=disabled #永久关闭
```

修改 /etc/httpd/conf/httpd.conf

```
#
# "/var/www/cgi-bin" should be changed to whatever your ScriptAliased
# CGI directory exists, if you have that configured.
#
<Directory "/var/www/cgi-bin">
    AllowOverride None
    Options ExecCGI
```

(下页继续)

(续上页)

```
    Require all granted
</Directory>
```

```
#
# AddHandler allows you to map certain file extensions to "handlers":
# actions unrelated to filetype. These can be either built into the server
# or added with the Action directive (see below)
#
# To use CGI scripts outside of ScriptAliased directories:
# (You will also need to add "ExecCGI" to the "Options" directive.)
#
AddHandler cgi-script .cgi .py .sh
```

将文件夹 *cgi-bin* 内文件复制到 */var/www/cgi-bin*，将文件夹 *html* 内文件复制到 */var/www/html*。

linux 下 apache 启动、停止、重启命令

```
service httpd start  #启动
service httpd restart #重新启动
service httpd stop #停止服务
```

15.2.2 apache 无法启动

检查 log 文件 **/var/log/httpd/error_log**

如果有以下内容，则说明由于 SSL 证书过期导致无法正常启动

```
[Mon Feb 15 18:44:20.458085 2021] [:error] [pid 1482] SSL Library Error: -8181␣
↪Certificate has expired
[Mon Feb 15 18:44:20.458106 2021] [:error] [pid 1482] Unable to verify certificate
↪'Server-Cert'. Add "NSSEnforceValidCerts off" to nss.conf so the server can␣
↪start until the problem can be resolved.
```

也可通过以下命令来检查证书是否过期

```
certutil -d /etc/httpd/alias -L -n Server-Cert
```

简单的处理方法是先设置禁止检查证书，待更新证书后再取消此设置，操作方法：在 **/etc/httpd/conf.d/nss.conf** 中加入 **NSSEnforceValidCerts off** 此行设置

15.2.3 firewalld

```
# 取消firewalld的锁定
systemctl unmask firewalld
## 使用systemctl start firewalld命令开启防火墙的时候，却开不成功，出现Failed to␣
↪start firewalld.service: Unit is masked的错误，是firewalld服务被锁定了
```

```
# 查看firewalld状态
systemctl status firewalld
```

```
# 开启防火墙
systemctl start firewalld
##没有任何提示即开启成功
```

```
# 关闭防火墙
systemctl stop firewalld
```

```
# 重启防火墙
firewall-cmd --reload
```

```
# 查询TCP/UDP的80端口占用情况
firewall-cmd --query-port=80/tcp
firewall-cmd --query-port=80/udp
## 如果返回结果为“no”，则表示该端口尚未开放
```

```
# 永久开放TCP/UDP的端口
firewall-cmd --permanent --zone=public --add-port=3306/tcp
firewall-cmd --permanent --zone=public --add-port=80/tcp
firewall-cmd --permanent --zone=public --add-port=80/udp

## 提示FirewallD is not running，则防火墙没开启
```

```
# 启动 Apache
systemctl start httpd

# 重启http服务
systemctl restart httpd

##查询状态
systemctl status httpd

##  默认监听80端口 /var/www/html
```

15.3 Ubuntu 20

15.3.1 CGI 配置

修改 apache 的 **/etc/apache2/conf-enabled/serve-cgi-bin.conf** 文件

改为

```
<IfDefine ENABLE_USR_LIB_CGI_BIN>
        ScriptAlias /cgi-bin/ /var/www/cgi-bin/
        <Directory "/var/www/cgi-bin">
                AllowOverride All
                Options +ExecCGI -MultiViews +SymLinksIfOwnerMatch
                Require all granted
                AddHandler cgi-script .cgi .sh
        </Directory>
</IfDefine>
```

启用 cgi mod

```
sudo ln -s /etc/apache2/mods-available/cgid.conf /etc/apache2/mods-enabled/cgid.
↪conf
sudo ln -s /etc/apache2/mods-available/cgid.load /etc/apache2/mods-enabled/cgid.
↪load
sudo ln -s /etc/apache2/mods-available/cgi.load /etc/apache2/mods-enabled/cgi.load
```

重启 apache 服务

```
sudo /etc/init.d/apache2 restart
```

15.4 网页配置

以上 CentOS/Ubuntu，共用一套网页控制源码，将 *www* 中 **cgi-bin** 和 **html** 两个文件夹中的程序，复制到 */var/www* 中的相应文件夹中。

然后进入 */var/www/cgi-bin* 文件夹，使用 root 权限执行，make clean ，然后执行 make

CHAPTER 16

网页控制界面

16.1 控制界面

通过控制寄存器来改变实验触发模式，逻辑信号的延迟与展宽等。

按钮“Program FPGA”用于初始化系统配置，当可编程逻辑模块上电之后第一时间点击该按钮来完成系统的初始化。

可以保存 5 个实验设置参数，分别为“Experimental setup 1-5”。通过修改寄存器进行实验逻辑配置之后，可以点击按钮“Save”保存，将会把当前 FPGA 寄存器参数保存到选定的实验配置中。按钮“Load”用于将选择的实验配置加载到 FPGA 中。

示波器监视部分用于选择 4 个 LEMO 输出通道的输出信号，下表中列出了当前所有可供选择的选项。点击“Read”按钮即可读取当前的设置参数，按钮“Change”用于将当前输入框的参数写入 FPGA 中。

表 1: 4 通道 LEMO 输出

vaule	signal
00	ungated trigger
01	status
02	clear
03	gate adc
04	gate 1x90
05	GIN(0)
06	GIN(1)
07	reset
08	hardware busy
09	long trigger(20 us)
10	veto
11	保留
12	保留
13	保留
14	保留
15	保留
16	OR32_00
17	OR32_01
18	OR32_02
19	OR32_03
20	OR32_04
21	OR32_05
22	OR32_06
23	OR32_07
24	保留
25	保留
26	保留
27	保留
28	保留
29	保留
30	保留
31	保留
32	A(0) GDG
33	A(1) GDG
34	A(2) GDG
35	A(3) GDG
36	A(4) GDG
37	A(5) GDG
38	A(6) GDG
39	A(7) GDG
40	A(8) GDG
41	A(9) GDG
42	A(10) GDG
43	A(11) GDG
44	A(12) GDG
45	A(13) GDG

下页继续

表 1 – 续上页

vaule	signal
46	A(14) GDG
47	A(15) GDG
48	A(16) GDG
49	A(17) GDG
50	A(18) GDG
51	A(19) GDG
52	A(20) GDG
53	A(21) GDG
54	A(22) GDG
55	A(23) GDG
56	A(24) GDG
57	A(25) GDG
58	A(26) GDG
59	A(27) GDG
60	A(28) GDG
61	A(29) GDG
62	A(30) GDG
63	A(31) GDG

寄存器设置部分用于读取或者修改寄存器设置参数。读取寄存器时，需要输入要读取寄存器的地址，然后点击按钮“Read”；修改寄存器时，输入要修改寄存器的地址以及参数值，然后点击按钮“Write”。

表 2: control register

vaule	function
0x1900	OR32 A[31:0]
0x1904	OR32 A[31:0]
0x1908	OR32 A[31:0]
0x190C	OR32 A[31:0]
0x1910	OR32 A[31:0]
0x1914	OR32 A[31:0]
0x1918	OR32 A[31:0]
0x191C	OR32 A[31:0]
0x1920	AND32 A[31:0]
0x1924	AND32 A[31:0]
0x1928	AND32 A[31:0]
0x192C	AND32 A[31:0]
0x1930	AND32 A[31:0]
0x1934	AND32 A[31:0]
0x1938	AND32 A[31:0]
0x193C	AND32 A[31:0]
0x1968	trigger select(见下方关于触发源的选择)
0x1970	port B enable/disable in hard busy A[31:0]
0x1974	[25:16]ADC Gate [9:0]V1x90 Delay
0x0978	[0] software/hardware busy
0x19F8	[0] debug mode（test mode pars copy when 1）

表 3: 触发源的选择

vaule	function
00	A(0)
01	A(1)
02	A(2)
03	A(3)

下页继续

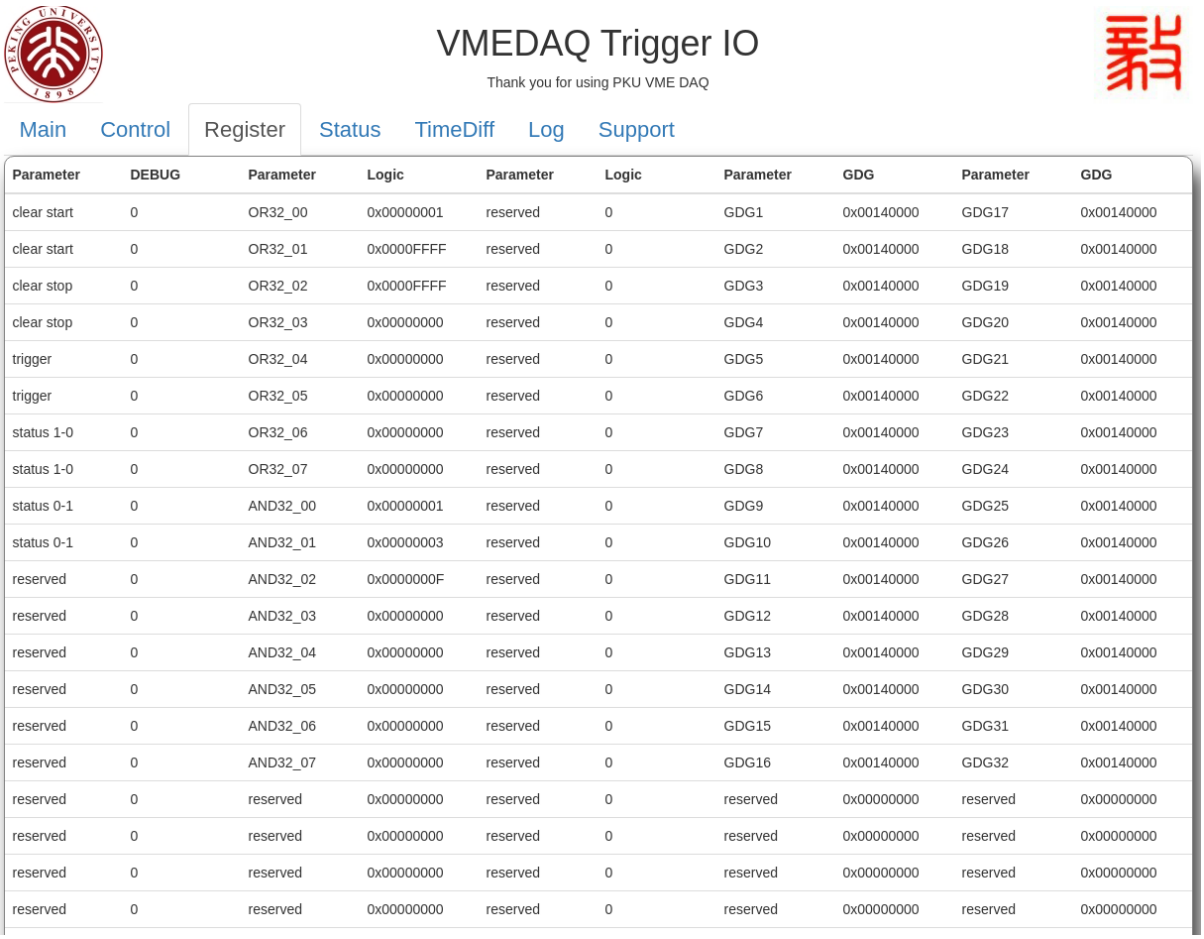
表 3 – 续上页

vaule	function
04	A(4)
05	A(5)
06	A(6)
07	A(7)
08	A(8)
09	A(9)
10	A(10)
11	A(11)
12	A(12)
13	A(13)
14	A(14)
15	A(15)
16	A(16)
17	A(17)
18	A(18)
19	A(19)
20	A(20)
21	A(21)
22	A(22)
23	A(23)
24	A(24)
25	A(25)
26	A(26)
27	A(27)
28	A(28)
29	A(29)
30	A(30)
31	A(31)
32	OR32_00
33	OR32_01
34	OR32_02
35	OR32_03
36	OR32_04
37	OR32_05
38	OR32_06
39	OR32_07
40	AND32_00
41	AND32_01
42	AND32_02
43	AND32_03
44	AND32_04
45	AND32_05
46	AND32_06
47	AND32_07

GDG 控制部分用于控制 32 通道信号的延迟和展宽。

读取信号的延迟和展宽时，需要输入要读取的通道，然后点击按钮“Read”；修改某通道的延迟和展宽时，输入要修改通道编号以及参数值，然后点击按钮“Write”。

16.2 寄存器界面



Parameter	DEBUG	Parameter	Logic	Parameter	Logic	Parameter	GDG	Parameter	GDG
clear start	0	OR32_00	0x00000001	reserved	0	GDG1	0x00140000	GDG17	0x00140000
clear start	0	OR32_01	0x0000FFFF	reserved	0	GDG2	0x00140000	GDG18	0x00140000
clear stop	0	OR32_02	0x0000FFFF	reserved	0	GDG3	0x00140000	GDG19	0x00140000
clear stop	0	OR32_03	0x00000000	reserved	0	GDG4	0x00140000	GDG20	0x00140000
trigger	0	OR32_04	0x00000000	reserved	0	GDG5	0x00140000	GDG21	0x00140000
trigger	0	OR32_05	0x00000000	reserved	0	GDG6	0x00140000	GDG22	0x00140000
status 1-0	0	OR32_06	0x00000000	reserved	0	GDG7	0x00140000	GDG23	0x00140000
status 1-0	0	OR32_07	0x00000000	reserved	0	GDG8	0x00140000	GDG24	0x00140000
status 0-1	0	AND32_00	0x00000001	reserved	0	GDG9	0x00140000	GDG25	0x00140000
status 0-1	0	AND32_01	0x00000003	reserved	0	GDG10	0x00140000	GDG26	0x00140000
reserved	0	AND32_02	0x0000000F	reserved	0	GDG11	0x00140000	GDG27	0x00140000
reserved	0	AND32_03	0x00000000	reserved	0	GDG12	0x00140000	GDG28	0x00140000
reserved	0	AND32_04	0x00000000	reserved	0	GDG13	0x00140000	GDG29	0x00140000
reserved	0	AND32_05	0x00000000	reserved	0	GDG14	0x00140000	GDG30	0x00140000
reserved	0	AND32_06	0x00000000	reserved	0	GDG15	0x00140000	GDG31	0x00140000
reserved	0	AND32_07	0x00000000	reserved	0	GDG16	0x00140000	GDG32	0x00140000
reserved	0	reserved	0x00000000	reserved	0	reserved	0x00000000	reserved	0x00000000
reserved	0	reserved	0x00000000	reserved	0	reserved	0x00000000	reserved	0x00000000
reserved	0	reserved	0x00000000	reserved	0	reserved	0x00000000	reserved	0x00000000
reserved	0	reserved	0x00000000	reserved	0	reserved	0x00000000	reserved	0x00000000

当您访问寄存器界面时，该界面刷新一次。该界面展示了可以设置的寄存器的参数，主要用来进行参数的核查。

16.3 状态监视界面

当您访问状态页面时，该页面将每 5 秒钟自动刷新一次。

VMEDAQ Trigger IO

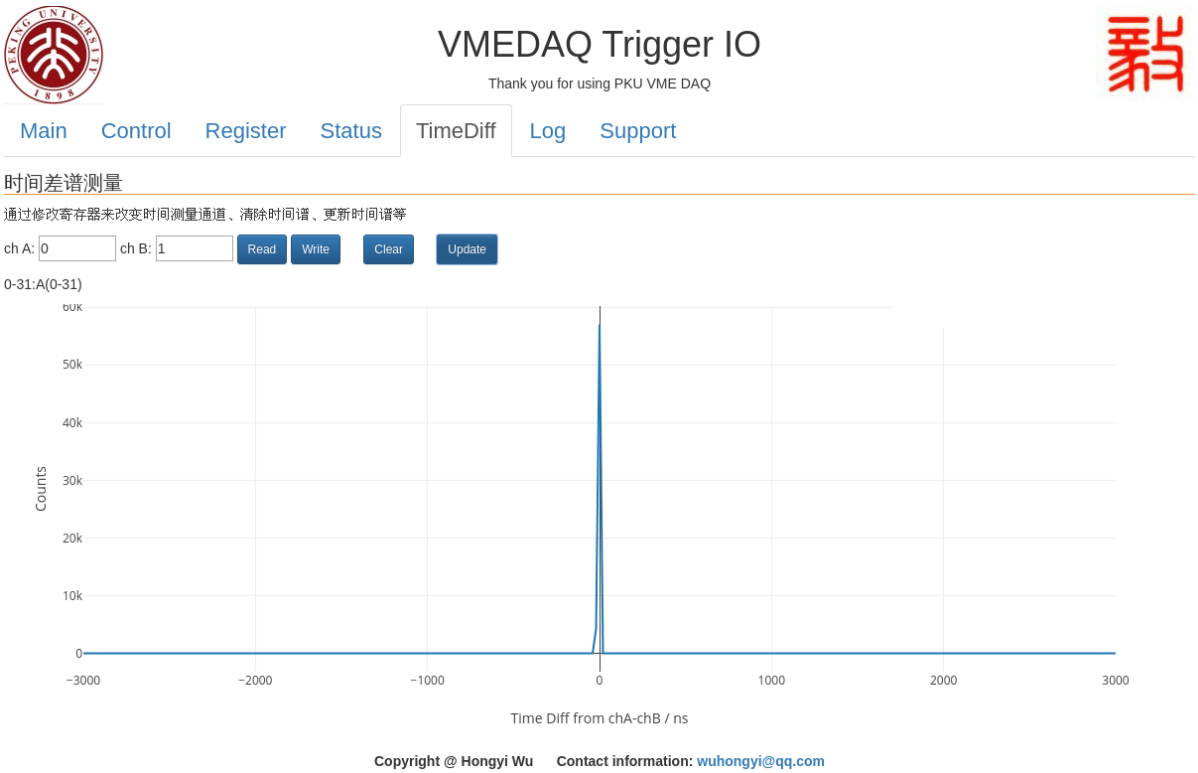
Thank you for using PKU VME DAQ

Main Control Register Status TimeDiff Log Support

Parameter	Status	Parameter	Scaler	Parameter	Scaler	Parameter	Scaler	Parameter	Scaler
FW_VERSION	0x20200701	OR32_00	19365	A(0)	19365	A(16)	0	ungate trigger	19365
SW_VERSION	0x20200512	OR32_01	25268	A(1)	22598	A(17)	0	gate trigger	4737
DateOfExpiry	0x20201231	OR32_02	25268	A(2)	0	A(18)	0	clear 1	4736
trigger select	0	OR32_03	0	A(3)	0	A(19)	0	clear 2	0
port B enable	0x000001FE	OR32_04	0	A(4)	0	A(20)	0	reserved	0
ADC GATE(ns)	4000	OR32_05	0	A(5)	0	A(21)	0	reserved	0
GATE Delay(ns)	3800	OR32_06	0	A(6)	0	A(22)	0	reserved	0
busy mode	0	OR32_07	0	A(7)	0	A(23)	0	reserved	0
reserved	0	AND32_00	19365	A(8)	0	A(24)	0	reserved	0
reserved	0	AND32_01	16725	A(9)	0	A(25)	0	reserved	0
reserved	0	AND32_02	0	A(10)	0	A(26)	0	reserved	0
reserved	0	AND32_03	0	A(11)	0	A(27)	0	reserved	0
reserved	0	AND32_04	0	A(12)	0	A(28)	0	reserved	0
reserved	0	AND32_05	0	A(13)	0	A(29)	0	reserved	0
reserved	0	AND32_06	0	A(14)	0	A(30)	0	reserved	0
reserved	0	AND32_07	0	A(15)	0	A(31)	0	reserved	0
Monitor 1	0	reserved	0	reserved	0	reserved	0x00000000	time of event	15516
Monitor 2	1	reserved	0	reserved	0	reserved	0x00000000	TOT of clear1	298
Monitor 3	2	reserved	0	reserved	0	reserved	0x00000000	TOT of clear2	0
Monitor 4	3	reserved	0	reserved	0	reserved	0x00000000	reserved	0

该页面主要用于实时的计数率监视。当前版本包含了 A 口 32 个输入通道的计数率，4 个 LEMO 输出通道的计数率，OR_00-07 的计数率，AND_00-07 的计数率，ungated trigger、gate trigger 等信号的计数率等。

16.4 在线时间差测量



该页面实现了任意两个逻辑信号的时间差谱测量（chA-chB，时间差大于 0 表示 chA 信号晚于 chB 信号）。按钮“Read”用于读取信号源参数；按钮“Write”用于更改信号源；按钮“Clear”用于清除 FPGA 中的时间差谱，当更改信号源后必须清除 FPGA 中的时间差谱。按钮“Update”可用于从 FPGA 中读取当前的时间差谱并显示在网页上。

表 4: time difference meaurement sources

vaule	signal
00	A(0) GDG
01	A(1) GDG
02	A(2) GDG
03	A(3) GDG
04	A(4) GDG
05	A(5) GDG
06	A(6) GDG
07	A(7) GDG
08	A(8) GDG
09	A(9) GDG
10	A(10) GDG
11	A(11) GDG
12	A(12) GDG
13	A(13) GDG
14	A(14) GDG
15	A(15) GDG
16	A(16) GDG
17	A(17) GDG
18	A(18) GDG
19	A(19) GDG
20	A(20) GDG

下页继续

表 4 – 续上页

vaule	signal
21	A(21) GDG
22	A(22) GDG
23	A(23) GDG
24	A(24) GDG
25	A(25) GDG
26	A(26) GDG
27	A(27) GDG
28	A(28) GDG
29	A(29) GDG
30	A(30) GDG
31	A(31) GDG
32	OR32_00
33	OR32_01
34	OR32_02
35	OR32_03
36	OR32_04
37	OR32_05
38	OR32_06
39	OR32_07
40	AND32_00
41	AND32_01
42	AND32_02
43	AND32_03
44	AND32_04
45	AND32_05
46	AND32_06
47	AND32_07