ALPIDE Software - User manual

Markus Keil, Felix Reidt

rev. 2, May 4, 2017

This manual describes the software used to perform standard tests of the ALPIDE chip with different readout boards and in different setups (single chip with DAQ board, HICs or staves with MOSAIC board...). The manual is intended as a quick start guide to get the ALPIDE software installed and perform the most important tests via the command line interface. For a description on the software structure and how to implement your own test applications, please refer to the developer's manual .

1 Installation and First Steps

1.1 Prerequisites

The software is currently supported to run only on the three following operating systems installing the listed packages. This reference installations are used to test software after every commit to the repository.

- **Ubuntu 16.04 LTS:**

```
yum install -y gcc gcc-c++ make cmake tar zlib wget subversion \
  git libusb1-devel tinyxml-devel
  wget -O /opt/root.tar.gz \
    https://root.cern.ch/download/root_v6.08.06.Linux-ubuntu16-
        x86_64-gcc5.4.tar.gz
  cd /opt/ ; tar xzfv root.tar.gz
```

  ROOT has to be loaded using

```
source /opt/root/bin/thisroot.sh .
```

- **CentOS CERN 7:**

```
yum install -y gcc gcc-c++ make cmake tar zlib wget subversion \
  git libusb1-devel tinyxml-devel
  wget -O /opt/root.tar.gz \
    https://root.cern.ch/download/root_v6.08.06.Linux-centos7-
        x86_64-gcc4.8.tar.gz
  cd /opt/ ; tar xzfv root.tar.bz
```

  ROOT has to be loaded using

```
source /opt/root/bin/thisroot.sh .
```

- **Scientific Linux CERN 6 *DEPRECATED*:**

```
yum install -y libX11-devel libXft-devel libXpm-devel \
  libXext-devel libusb1-devel wget tinyxml-devel
wget -O /etc/yum.repos.d/slc6-devtoolset.repo \
  http://linuxsoft.cern.ch/cern/devtoolset/slc6-devtoolset.repo
yum install -y devtoolset-2
export PATH=/opt/rh/devtoolset-2/root/usr/bin:$PATH
# Python
wget https://www.python.org/ftp/python/2.7.13/Python-2.7.13.tar.
    xz
tar xfv Python-2.7.13.tar.xz
cd Python*
./configure --prefix=/usr/local/
make install
cd ../
rm -rf Python-*
# CMake
wget https://cmake.org/files/v3.8/cmake-3.8.1.tar.gz
tar xfv cmake-3.8.1.tar.gz
cd cmake-*
./configure --prefix=/usr/local
make install
cd ../
rm -rf cmake-*
export PATH=/usr/local/bin:$PATH
wget -O /opt/root.tar.gz \
  https://root.cern.ch/download/root_v6.08.06.source.tar.gz
cd /opt
tar xzfv root.tar.gz
mv /opt/root-* /opt/root
cd /opt/root
./configure
make
```

The `devtoolset-2` is necessary on SLC6 and the `$PATH` has to be modified using the following commands:

```
source /opt/rh/devtoolset-2/enable .
export PATH=/opt/rh/devtoolset-2/root/usr/bin:/usr/local/bin:
    $PATH
source /opt/root/bin/thisroot.sh
```

This command can either be executed before compiling or running the software or added to the `.bashrc`.

1.2 Installation

The software is available in a gitlab repository:

`https://gitlab.cern.ch/alice-its-alpide-software/new-alpide-software`

To check it out for the first time use

```
git clone https://username@gitlab.cern.ch/alice-its-alpide-software/new-
    alpide-software
```

with your user name. Note that your account needs to be added to a list of users in order to being able to access the repository. Versions are updated regularly, make sure you have checked out the latest version. After the clone you will find the software in a directory `new-alpide-software`, an additional subdirectory `analysis` contains standard macros to analyse the data.

Executing Tests
After compiling the software package you will find a certain number of standard tests, described below, in the form of individual executables. Each executable performs a specific test once and, if applicable, writes the output data in a text file in the subdirectory `Data`. The settings of the chip(s), readout board(s) and the scan itself can be modified in the config file `Config.cfg`. Some of the available parameters are described below, for a more detailed description of the syntax and the available parameters please refer to the comments in the config file itself.

Executing Tests with the DAQ Board
To execute tests with the DAQ board setup the FX3 chip has to be configured. The procedure and the necessary files are identical to the old pALPIDEfs-software:

```
./download_fx3 -t RAM -i SlaveFifoSync.img
```

In case you have not performed tests with the DAQ board before you can download the necessary files from the directory `fx3` in the repository https://gitlab.cern.ch/alice-its-alpide-software/pALPIDEfs-software. You can then compile the tool to configure the FX3 chip (execute `build_mac.sh` or `build_linux.sh`, resp., in the directory `fx3`).

2 Setup Definition

The standard tests of the ALPIDE software, provided in command line applications, work in the same way for different types of setups. There are several predefined setups that can be selected via the config file. Each of these predefined setup types implements a readout board (DAQ or MOSAIC), a certain number of chips with predefined chip IDs and mapping of chips to the data lines and command interfaces of the MOSAIC board.

Several config-file switches exist to modify these pre-defined setups, e.g. to assign different chip IDs, command interfaces or data receivers to individual chips.

The currently available setups are the following:

1. Single chip: a setup with a single chip connected to the DAQ board is selected by the config-file switch:

   ```
   DEVICE CHIP
   ```

   In this case the chip has the chip-ID 16 (OB Master), which is the standard configuration for single chip tests. If needed this can be changed by adding a switch `CHIPID_0 x` (Since there is only one chip in the setup the index `_0` can also be omitted.)

2. Single chip with MOSAIC: to test a single chip with the MOSAIC board the switch

   ```
   DEVICE CHIPMOSAIC
   ```

   has to be used. In this case the software assumes a chip mounted on a carrier with SAMTEC connector, which is connected through an IB Firefly-Eyespeed adapter to the MOSAIC. The default chip ID is 16. In case a different setup is used, mappings of control and data lines need to be changed using the switches `CONTROLINTERFACE_0 x` and `RECEIVER_0 y`, substituting the appropriate values for x and y.

3. Inner Barrel HIC: test of an inner barrel HIC is done using the switch

   ```
   DEVICE IBHIC
   ```

   This assumes an inner barrel HIC with 9 chips with chip IDs ranging from 0 to 8. The HIC is supposed to be connected to the MOSAIC via a standard IB Firefly-Eyespeed adapter. Again all basic settings (mapping and chip IDs) can be changed.

4. Outer Barrel HIC: test of an outer barrel HIC is done using the switch

   ```
   DEVICE OBHIC
   ```

This assumes an outer barrel HIC with 14 chips with chip IDs ranging from 16 to 22 and from 24 to 30. The The HIC is supposed to be connected to the MOSAIC via a standard OB Firefly-Eyespeed adapter. Again all basic settings (mapping and chip IDs) can be changed. When changing individual chip settings, note that the index of the config-file switch refers to the running index and not to the chip ID. These are identical for the IB, but not for the OB HIC. The correspondence between chip index and (default) chip ID is shown in the following table:

Index	0	1	2	...	6	7	8	...	13
Chip ID	16	17	18	...	22	24	25	...	30

This numbering scheme corresponds to a module ID of 1. In a future implementation it will be possible to change this via config file, currently this can already be done by changing the individual chip IDs.

5. Outer Barrel Half-Stave: The structure for an outer barrel half-stave is prepared with the device `HALFSTAVE`, however this device type requires in addition the number of modules, e.g. for a fully mounter outer layer half-stave:

   ```
   DEVICE HALFSTAVE
   NMODULES 7
   ```

   The prepared setup then contains `NMODULES` modules with 14 chips each and ascending module IDs 0 ... `NMODULES + 1`. If the module IDs are to be changed, this currently has to be done chipwise, however modulewise switches are under preparation.

Chip Enabling / Disabling
By default all defined chips are enabled for configuration and scans. For all multi-chip devices single chips can be disabled in the config file by adding a line `ENABLED_n 0` to disable chip `n`. In addition the software checks before execution of the actual test whether an enabled chip responds to the command interface. If this is not the case the chip is auto-disabled. In case of the OB module the daisy chain is established after this procedure, skipping all disabled chips [1].

3 Tests

This section describes the standalone test programs provided with the ALPIDE software. Each test is executed by running an individual command line executable. Configurations of test parameters (like number of mask stages, number of injections etc.) can be set in the config file `Config.cfg`. If the test generates output data this is written to the subdirectory `Data`

[1] Note that it is currently not possible to read out a daisy chain with the corresponding master disabled

3.1 FIFO Test

The FIFO test is a quick test to check the communication with the chips' control interfaces. It writes three different bit patterns (`0x0000, 0xffff` and `0x5555`) into each cell of the end-of-column FIFOs, reads them back and checks the correctness of the readback value. The test is executed by running

```
./test_fifo
```

The test will print on screen the numbers of errors for each chip (seperately for each pattern) and at the end the sum of errors for all tested chips. (If for debugging purposes output of the individual errors is needed the program has to be recompiled setting the variable `bool Verbose` in `main_fifo.cpp` to `true`).

3.2 On-chip DAC Scan (DAQ board only)

The output of the on-chip DACs can be connected to monitoring pins of the ALPIDE chip and measured by ADCs on the DAQ board. The DAC Scan measures the characteristics of all on-chip DACs. The test is started by running

```
./test_dacscan
```

For each DAC it loops over the values from 0 to 255 and measures the output values. The measured values are written into one file for each DAC.

3.3 Digital Scan

The digital scan generates a digital pulse in a number of pixels and reads the hits out. It is started with the command

```
./test_digital
```

The number of injections and the number of injected pixels is configurable with the parameters `NINJ, PIXPERREGION, NMASKSTAGES` in the config file (see also the comments in the box below).

The output data is written into a file `DigitalScan.dat`, each line has the format

```
Doublecol Address NHits
```

with `Doublecol` ranging from 0 to 511, `Address` from 0 to 1023 (Address is the address as described in the ALPIDE manual, not the row number).

Analysis Macro
In the subdirectory `analysis` you can find the macro `Hitmap.C`, which can be used to visualize the output data of the digital scan. If the macro is invoked in the form

```
.x Hitmap.C (<filename>, <number of injections>)
```

it draws a hitmap of the given chip and returns the numbers of pixels that have registered 0 hits or more / less hits than injections. If the second parameter is omitted, only the hitmap is drawn.

General remarks on scans:

- Mask stages: All injection-based scans (digital, analogue and threshold) work on a certain number of pixels at a time. This number of pixels is configurable by the parameter `PIXPERREGION` in the config file, which gives the number of pixels per region (!) that is injected simultaneously. (A value of 1 corresponds to 32 pixels per chip. The maximum value is 32, which corresponds to a complete row.) After the required number of injections has been done the scan moves to the next set of pixels. The number of such mask stages that is performed is configurable with the parameter `NMASKSTAGES`. In order to scan the entire chip the mask has to be staged 16384 / `PIXPERREGION` times, a lower number accordingly leads to a percentage $< 100\%$ of scanned pixels.
- Output files: due to the large amount of data in particular for threshold scans, output data is written only for pixels with > 0 hits. In case of multichip devices, one output file per chip is created.

3.4 Threshold Scan

The threshold scan performs analogue injections, looping over the charge. For each charge point 50 injections are performed. The command is

```
./test_threshold
```

The output file `ThresholdScan.dat` contains the raw data, i.e. the number of hits for each charge point, in the format

```
Doublecol Address Charge NHits
```

In addition to the parameters present for the digital scan, for the threshold scan also the charge range can be configured in the config file; the default setting ranges from 0 to 50 DAC units.

Analysis Macro
In the subdirectory `analysis` you can find the macro `FitThresholds.C`, which is used to fit the s-curves for the scanned pixels. For speed reasons it should be used in the compiled form

```
.x FitThresholds.C+ (<filename>)
```

The fitted values are written in an output file `FitValues... .dat` in the format `doublecol address thresh noise chisq`, which can be visualized in form of a threshold map with the macro `ThresholdMap.C` in the same directory.

3.5 Noise Occupancy

The scan gives a selectable number of random triggers and returns the number of hits. The command is

```
./test_noiseocc
```

The output file format is identical to the digital scan.

Analysis Macro
The macro `NoiseOccupancyRawToHisto` in the subdirectory `analysis` can be used to prepare a histogram with the noise occupancy as a function of the number of masked pixels. The path to the file, which is to be analysed, has to be given as a parameter. Note that the macro assumes a certain number of triggers, which is read from a config file in case of the DAQ board. If the MOSAIC is used this config file does not exist and the macro has to be adjusted accordingly (i.e. remove the reading of the config file and set the variable `n_trg` manually to the correct number of triggers). This will be adjusted in a future version.

3.6 Source Scan

A dedicated source scan is currently not implemented, however by adjusting the strobe length the noise occupancy scan can be used to perform a simple source scan (without single event information). The hitmap macro can then be used to show the hit data.