pALPIDEfs software - Installation and command line interface

Markus Keil

rev. 1, August 3, 2015

This manual is intended as a quick start guide to get the pALPIDE software installed and perform the most important tests via the command line interface. It does not (yet) contain a detailed overview of the software structure or instructions for more specialised tests that require code changes. Nonetheless the tests described here should be sufficient to perform a complete test of the chip functionality.

1 Installation

The software is available on a git repository:

`https://git.cern.ch/web/pALPIDEfs-software.git`

To check it out for the first time use

`git clone https://username@git.cern.ch/reps/pALPIDEfs-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. The repository contains four subdirectories:

- pALPIDEfs-software: The main software package with the low-level driver as well as test routines.
- crystalball: a lean standalone tool to configure the chip and read events.
- FPGA: a script to download the firmware to the FPGA.
- FX3: the configuration file for the FX3 chip as well as a tool to download it to the chip.

In order to compile the software you need to have libusb installed. If not yet installed, download and install version 1.0 from `http://www.libusb.org`. Once this is done you should be able to compile both the software (execute `make` in the directory `pALPIDEfs-software`) and the tool to configure the FX3 chip (execute `build_mac.sh` or `build_linux.sh`, resp., in the directory `fx3`).

If compilation fails this is most likely due to the installation path of the libusb package. In that case you need to locate the path of the header and library file on your system and modify the makefile / scripts accordingly.

After compilation, on Linux systems, the udev rules have to be set in order to access the new USB device. In order to do this add a rules file containing the following three lines in the directory `/etc/udev/rules.d/`:

```
BUS=="usb", SYSFS{idVendor}=="04b4", SYSFS{idProduct}=="00f3", MODE="0666"
BUS=="usb", SYSFS{idVendor}=="04b4", SYSFS{idProduct}=="00f1", MODE="0666"
BUS=="usb", SYSFS{idVendor}=="04b4", SYSFS{idProduct}=="00f0", MODE="0666"
```

(This is the syntax for scientific Linux and might need to be adapted for other installations, e.g. for Ubuntu `SYSFS` has to be substituted with `ATTRS` and `BUS` with `SUBSYSTEM`.)

2 Getting Started

After installation of the software you should be able to start testing with two simple steps:

1. Configure FX3 chip: This needs to be done after each power-cycle of the board and each reconfiguration of the FPGA. In order to download the configuration file to the FX3 chip you should go to the directory `fx3` and execute

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

   Upon successful execution you should see the message

   ```
   FX3 firmware programming to RAM completed
   ```

2. Start the program: The test program is started by executing `runTest` in the directory `pALPIDEfs-software`. The executed test is determined by the command line parameters passed to the program (see below).

3 Tests

The set of tests described in the following are a very first attempt to provide a comprehensive set of tests to qualify the pALPIDEfs-chip without having to dig deep into the software. The set of tests and even input parameters and output data format are therefore bound to change in the following weeks. Therefore please check for a newest version of the manual and the software frequently.

A list of the available tests and their syntax can be obtained from the command line by executing the program without any parameters (`./runTest`).

Each test is preceded by a powering-on and configuration of the chip. After each of the two steps the current consumptions (and the NTC temperature) are measured and printed on screen. After each test the chip is powered down (Note that this is not the case if the program crashes / is interrupted).

For tests that write output data you have to create a subdirectory `Data` under the working directory. All data files will be written there. The file name contains date and time of the scan start as a suffix. In addition some scans write a .cfg file, containing chip and scan settings used.

3.1 FIFO Test

The FIFO test is a quick test to check the JTAG communication with the chip. 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 started by passing the parameter `FIFO` to the program:

```
./runTest FIFO
```

3.2 On-chip DAC Test

The output of the on-chip DACs can be connected to monitoring pins of the pALPIDE chip and measured by ADCs on the DAQ board. The `READDAC` test loops over all chip DACs, measures their output once and prints the measured values to screen:

```
./runTest READDACS
```

In order to measure the full DAC characteristics the `SCANDACS` test can be used:

```
./runTest SCANDACS [PAR1]
```

For each DAC it loops over the values from 0 to 255 and measures the output values. The measured values are written into a file for each DAC. `PAR1` is an optional parameter for a step width ¿ 1, e.g. `./runTest SCANDACS 2` will loop over the DAC settings 0, 2, 4...

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 two parameters

```
./runTest SCANDIGITAL PAR1 PAR2
```

where `PAR1` is the number of injections per pixel, `PAR2` the number of mask stages. E.g. `./runTest DIGITAL 50 160` will test 1% of the pixels (cf. box), doing 50 digital injections into each.

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 pALPIDEfs manual, not the row number).

General remarks on scans:

- Mask stages: All injection-based scans (digital, analogue and threshold) work on a certain number of pixels at a time. In the current implementation this is one pixel in each of the 32 regions, starting from address 0 in the first double column of each region. After the required number of injections has been done the scan moves to the next set of pixels. In order to scan the entire chip the mask has to be staged 1024 * 16 times, 164 mask stages correspond to approximately 1% of the chip.
- Output files: due to the large amount of data in particular for threshold scans, output data is written only for pixels with > 0 hits.

3.4 Analogue Scan

The analogue scan works similar to the digital scan, however instead of generating a digital pulse after the discriminator, a programmable charge is injected into the preamplifier. The scan therefore requires an additional parameter:

```
./runTest SCANANALOGUE PAR1 PAR2 PAR3
```

with `PAR1` being the charge in DAC units[1], `PAR2` the number of injections per pixel and `PAR3` the number of mask stages. The output file format is identical to the one of the digital scan (filename `AnalogueScan.dat`).

3.5 Threshold Scan

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

[1] Preliminary calibration: 7 electrons / DAC unit

```
./runTest THRESHOLD PAR1 PAR2 PAR3 [PAR4 PAR5]
```

with the number of mask stages `PAR1` and the charge loop ranging from `PAR2` to `PAR3` (both in DAC units). 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
```

Parameters 4 and 5 are optional parameters to perform the test at a specific setting of VCASN and ITH different than the default setting. If used, both parameters have to be given in the order `VCASN ITH`.

3.6 Noise Occupancy

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

```
./runTest NOISEOCC PAR1 [PAR2 PAR3 PAR4]
```

with one two four parameters, the first parameter always being the number of events. If no other parameter is given, the scan is performed on all pixels with default settings. In case of three parameters, the second parameter is used as setting of VCASN, the third as setting of ITH. Four parameters can be given to measure the noise occupancy of a single pixel only. In that case `PAR2` is the region number, `PAR3` the double column and `PAR4` the address. If only parameters 1 to 3 are given, these are interpreted as `PAR2 = VCASN`, `PAR3 = ITH`.

The syntax given above does the noise occupancy measurement in readout mode B, using the same settings as for data taking (defined in `TTestsetup.h`). To perform a noise occupancy measurement in readout mode A the scan can be started with the command

```
./runTest NOISEOCC_A PAR1 [PAR2 PAR3 PAR4]
```

The meaning of the parameters stays the same. The hitmap is written into a file `NoiseOccupancy_....dat` in a format identical to digital and analogue scan.

3.6.1 Noise Occupancy Scan

A scan over a range of VCASN and ITH value can be performed, measuring the number of noise hits for each point:

```
./runTest NOISEOCCSCAN PAR1 ... PAR5 [PAR6]
```

The meaning of the parameters is (in order): number of events per point, VCASN range low and high, ITH range low and high and an optional mask file. The lines of the output files are in the format

```
VCASN ITH HitsSector0 HitsSector1 HitsSector2 HitsSector3
```

3.7 Source Scan

The source scan option does the same as the noise occupancy measurement, but with a longer STROBEB to increase the probability to see source hits with random trigger. The command is

```
./runTest SOURCE PAR1 [PAR2]
```

The first parameter is the number of events, the second an optional mask file name. Hits are written into a file `SourceScan.dat` (same format as above).

Also for the source scan a version in readout mode A is available:

```
./runTest SOURCE_A PAR1 [PAR2]
```

3.8 Noise Mask

This scan prepares a noise mask that can be used in the source scan. The scan is started by the command

```
./runTest NOISEMASK PAR1 PAR2
```

The scan will issue `PAR1` triggers and write all pixels that had one or more hits into an output file with name `PAR2`. This file can directly be used as `PAR2` of the source scan.

4 Scan Configuration

4.1 Configuration Settings

Chip, board and scan configurations can be given in three different ways (in decreasing order of precedence):

- As a command line parameter to the scan. E.g. `./runTest THRESHOLD 160 0 50 57 51 64` will run the threshold scan with settings of 57, 51 and 64 for VCASN, ITHR and IDB, respectively.
- In the config file Config.cfg. This file is read at the beginning of the program and contains switches for the most important settings of chip and DAQ board. All possible settings are explained in the comments in the file itself.
- Default values for all settings are defined in the source file TConfig.h (with the exception of the on-chip DAC settings, which are defined in TPalpidefs.cpp). Changing settings here and recompiling should be the last option if none of the other two options is available. In that case it has to be made sure that no changed settings are committed to git.

The precedence means that: If for a given setting no other value is given, the default value from TConfig.h (TPalpidefs.cpp) is used. This is overruled by a setting in Config.cfg, which in turn is overruled by a potential command line parameter.

4.2 Configuration Data Files

Several scans write a file containing configuration information and conditions data in addition to the scan data file. The name of the file is ScanConfig_date_time.dat, where date and time are identical to the one used in the filename of the scan data. It contains the most important configuration parameters for chip and DAQ board as well as:

- Software and firmware versions used.
- The temperature at the beginning and the end of the scan.
- The currents before and after initialisation of the chip.
- A list of stuck on pixels (pixels whose address was found twice in the same event).

5 Test Procedures

5.1 Functional Test (10 min)

This test program serves as a quick test program to test that the chip is fully functional. It does not give a complete characterisation.

1. FIFO Test:

   The FIFO test checks the basic communication with the chip and (as it is the first test) also gives a first measurement of the current consumption.

   Command: `./runTest FIFO`

   Items to check:

   - Current consumption before and after configuration.
   - FIFO test passed?
   - Note that chips with overcurrent will cause the FIFO test to fail, because the power supply is switched off before the FIFO test starts. This will show up as a printout on the screen, which will give the exact information on which current(s) had an overcurrent. This should then be registered as overcurrent and not as FIFO failure.

2. Test of the on-chip DACs:

   Command: `./runTest SCANDACS`

   Analysis macros: VoltageDACs.C, CurrentDACs.C

   Items to check:

   - DAC linearity for current and voltage DACs.
   - Voltage DAC range: all voltage DACs should show the same slope and reach 1.8 V. Note that some voltage DACs start at 0.3 - 0.4 V (instead of 0V) and therefore reach the 1.8 V earlier.

3. 1% Threshold scan:

   The 1% threshold scan gives a quick measure of the analogue performance, based on 1% (160×32) of the pixels. The pixels are distributed over the full matrix (as opposed to taking the first 160 rows), in order to avoid a bias of the measurement.

   Command: `./runTest THRESHOLD 160 0 50`

   Analysis macro: FitThresholds.C

   Items to check:

   - Number of pixels found by the fit macro close to 5120 (number of pixels scanned).
   - No large number of pixels without starting point
   - Average threshold, threshold RMS and noise values for all 4 sectors.

5.2 Complete Characterisation ($>$ 5 hrs)

The complete characterisation of the chip currently takes 5 - 10 hrs. Time is dominated by the threshold scan, it is therefore best to run this scan overnight.

1. DAC scan

   Command: `./runTest SCANDACS`

2. Digital Scan

   Command: `./runTest SCANDIGITAL 10 16384`

3. Full Threshold Scan

   Command: `./runTest THRESHOLD 16384 0 50`

4. Source Measurement

   Command: `./runTest SOURCE 1000000`

5. Noise Occupancy Measurement

   Command: `./runTest NOISEOCC 1000000`