**Simple and Inexpensive FPGA-based Analog Pulse Characterization Board**

Technical Description, Installation and Operation Manual

**Abstract**

This document describes how to build and operate a Analog Pulse Characterization Board statistical processing of analog pulses, such as these generated by number resolving photon counting detectors. This Board connects to a standard PC via USB-2 cable and can be used in popular data acquisition programs such as LabView.

**Introduction**

The growing interest in research and applications of photon-counting technology is evident. More and more research laboratories use single photon technologies for various applications, such as quantum communication and computing, single-molecule monitoring, precision measurements, etc. Meeting the demand for engineers and researchers with experience in single photon detection and statistical methods requires including simple photon-counting experiments in undergraduate laboratory courses. Development of new single photon detectors involves monitoring an analog output of the device and turning it into digital record for processing. In particular, we would like to separate detection signal from noise and retain its characteristics such as pulse height, length and integral. This design was inspired by the recent breakthrough in SPDs technology based on superconducting bolometery, which has an advantage of nearly 100% detection efficiency and photon-number-resolution, paired with the absence of afterpulsing and deadtime. Of course, our system can be used for any other analog input. Similar pulse characterization is required for many physics, electrical engineering and chemistry experiments. The board described here works best when rate, peak amplitude, length and integral of pulses generated by the underlying process is unknown a-priori, while the shape of pulses is expected to be similar to one another. Possible applications include characterization of single dust particles or scattering centers in gases or liquids. It could be used to monitor spikes on power or communication lines to detect anomalies and failures. This acquisition board can be paired with high-energy particle detectors. The device is also useful for characterizing periodic and quasi-periodic processes, such as monitoring heartbeat. We designed the acquisition instrument that samples this detector at a ~100 MHz rate, matching its typical bandwidth of this output. Other sampling rates (10 Hz-100 MHz) are possible with a firmware modification.

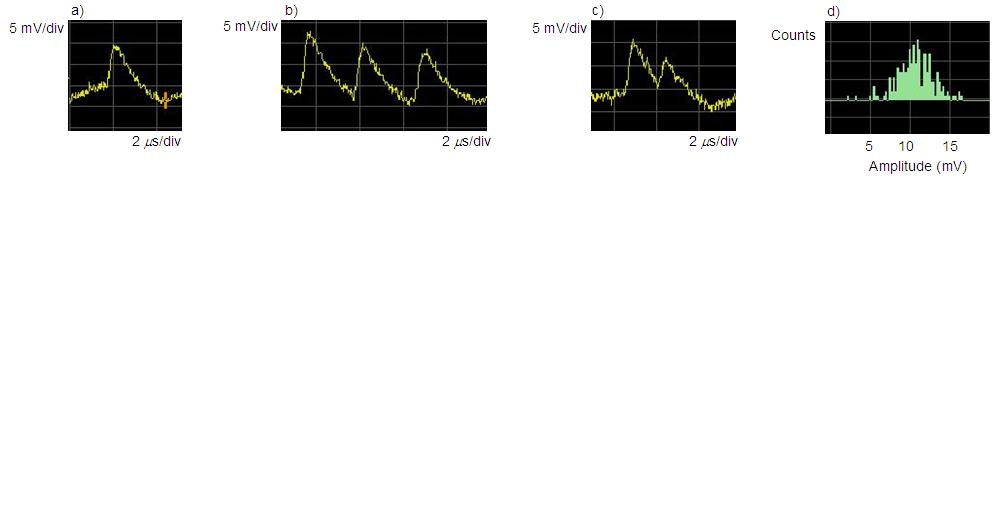
The main principles of development of this board are:

1. Easy assembly, installation and operation
2. Extremely low cost
3. Open Source software and FPGA firmware
4. Immediate connectivity to LabView

These principles provide a fast learning curve with an immediately useful device for simple photon counting (or any pulse counting) applications. At the same time, it allows more advanced users to accommodate unique applications, without spending too much time developing the board-to-PC interface.

**Description**

The board continuously monitors an analog output and detects voltage “pulses”. Leading and trailing edges of a pulse are defined as crossing a pre-set threshold voltage. Threshold voltage is adjustable via a command from the computer at the time of use, i.e. no modifications to hardware or software are needed. For each pulse, the following data is collected: peak voltage (amplitude) of the pulse, length of the pulse in clock cycles and the area (voltage integrated over the duration of the pulse), Fig. 1. In its current design, the monitoring occurs at 100 MHz, but can be easily changed via firmware to fit the needs of the monitoring task. Timestamps of leading edges of the pulse are not collected in this design to save communication bandwidth with the computer, but this data could also be collected after a straightforward firmware modification. The information about each detected pulse is transmitted via a USB-2 port to the computer. The peak transfer rate observed is >8 MB/s (corresponding to >1 million counts per second), but it is only limited by the speed of the PC used. The information then is processed and/or recorded to the hard drive in real time by the driver and is available (also in real time) to the end user.



amplitude

integral

length

Fig. 1. Principle of pulse characterization.

The device is build with two development boards. Flashy rev. K development board is an analog to digital converter (ADC) board measures the analog input. Xylo-EM FPGA development board with an Altera Cyclone II FPGA and a Cypress USB-2 chips applies data reduction algorithms and takes care of communicating data to a PC. Note here that one can use any ADC chip or an ADC development board of their choice, as long as it provides TTL output. There are two reasons behind our pick of an ADC development board. First, it powers up from a Xylo development board that takes power directly from a USB-2 port, so that no additional power modules are required. Second, most of development boards we explored have a high-bandpass filter for their analog input that is not desirable when pulse integration is performed.

Principle characteristics are presented in Table 1.

Table 1: Physical Characteristics

|  |  |
| --- | --- |
| **Parameter** | **Value** |
| Computer interface | USB-2 |
| Operating System | Windows 2000, XP |
| Computer Configuration | Any (Tests run on P4 2.8 Mhz, 1 GB ram) |
| Architecture | Open Source |
| Analog input channels | 1 |
| ADC width | 8 (or up to 16, after firmware modifications) |
| ADC voltage range | 0-700 mV |
| Native sampling rate | 100MHz |
| Sampling rate range | 100Hz … >150MHz |
| Board deadtime | None |
| Event threshold level | Adjustable |
| Digital Output | Amplitude, Length and Integral of each event |
| Digital Output data length | 8 bits per event |
| Highest USB data transfer rate, events | >1 106 Hz (PC dependent) |
| Estimated cost | <$400 |
| Estimated assembly and installation time | 8 h |

Table 2: Parts list

|  |  |
| --- | --- |
| **Part** | **Quantity** |
| Xylo-EM FPGA development board | 1 ea. |
| Flashy rev. K development board | 1 ea. |
| Generic Electronics Box 3”x2”x6” (min) | 1 ea. |
| USB-2 cable | 1 ea. |
| BNC connectors | 1 ea. |
| Plastic mounting screws, bolts | As necessary |

**Assembly and installation**

*Assembly*

This manual assumes that Xylo-EM FPGA and a Flashy rev. K development boards are used. For different board(s), changes in firmware and assembly may be necessary.

Step 1: Mount FPGA and Flashy boards to the box. Make sure that USB connector of FPGA board and BNC connector of Flashy board are accessible from the outside of the box. If desired, mount an external BNC jack on the box and solder it with the BNC cable to Flashy board.

Step 2: Solder FPGA pins to Flashy pins with short (<1”) wire pieces. Make sure that all pieces are of the same length. Refer to table 3.

Step 3: Connect analog input channel to a signal source with a BNC cable.

Step 4: Adjust potentiometers of Flashy so that the range of an ADC matches a signal source. IMPORTANT: Exercise extreme caution while adjusting potentiometers as they easily break.

Table 3: List of ADC-FPGA connections

|  |  |  |
| --- | --- | --- |
| **FPGA Pin** | **Flashy (ADC) Pin** | **Purpose** |
| 132 | 13 | Data 0 |
| 133 | 14 | Data 1 |
| 114 | 8 | Data 2 |
| 113 | 6 | Data 3 |
| 112 | 5 | Data 4 |
| 97 | 3 | Data 5 |
| 104 | 2 | Data 6 |
| 100 | 1 | Data 7 |
| 88 | 7 | ADC clock |
| Any pin marked ground | 4, 10 or 16 | Ground |
| Any pin marked +3.3V | 11, 12 | +3.3 V Power |

*Software installation*

Step 1: Driver installation. You will need “FX2\_USB.inf” and “FX2\_USB.sys” driver files from the board’s start-up kit.

Connect the board to a computers USB2 port and wait for windows to detect the board and start the Hardware Update Wizard (Fig. 2). Direct the wizard to the directory with “FX2\_USB.inf” and “FX2\_USB.sys” driver files and ignore a warning about windows certification of a driver (if issued). Refer to Fig. 2.

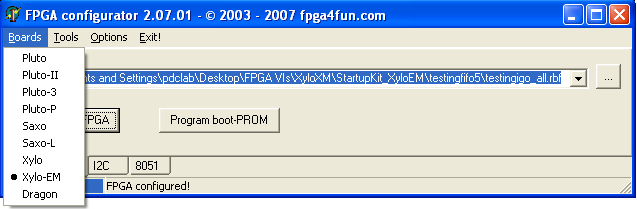


Fig. 2. Step 1 of software installation process.

Step 2: Configuring the board. Open FPGAconf.exe, available from your board’s start-up kit. Select Xylo-EM from menu “Boards” as a target board. Then, in the “Options” menu, set FX2 speed to 48 MHz. Refer to illustration (Fig. 3).

Provide a path to FPGA Time Resolving Acquisition Board firmware: the appropriate .rbf file from this distribution and configure FPGA with the selected .rbf file (click “Configure FPGA” button). The successful completion of this step ensures that the FPGA board is installed correctly and is ready to be used in a Time Resolving Acquisition Board mode.

Step 3: Testing the installation. The simplest way to test the installation is to use the LabView example fpgatest.vi, provided in this package. It can be used with any .rbf firmware. The program tests that electronic events are properly recorded by the board, received by a PC and made available to an end LabView user. The receipt and initial processing of data are handled by a stand-alone dll. This test is successful if the electronic events statistics is correctly displayed by a vi.



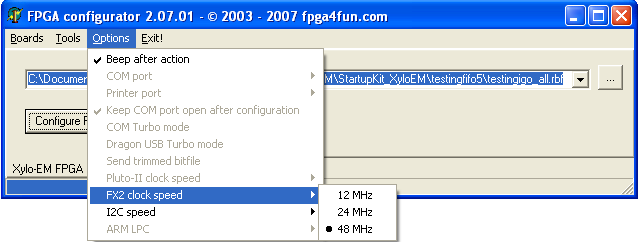


Fig. 3. Step 2 of software installation process. Setting up the configuration program.

If the test fails, unplug a USB cable, wait for 10-20 seconds and plug it back in. Repeat steps 2 and 3. If this does not help, the PC needs to be restarted. Repeat steps 2 and 3. If test fails again, check all the connections. It is recommended to use commercial function generators instead of real signal sources for debugging.

**Operation**

Using an FPGA configurator, select the analog\_FPGA.rbf file (or a custom-produced file corresponding to the mode of operation of choice) and click “Configure FPGA” button. This procedure has to be done after every off-on power cycle to a board.The operation mode of this device can be easily changed by loading the board with custom firmware (.rbf file). Advanced users are encouraged to download the source code, make custom firmware modifications, and share them with the project core, however, custom firmware is not supported by this project.

**LabView interface**

In this section we will discuss basic operation modes using standard existent software from the viewpoint of LabView integration. The basic example of operating the board, fpgatest.vi, covers all important stages of communication, and can be used for various design extensions. fpgatest.vi establishes communication with an FPGA, sets event thresholds and plots three histograms that characterize pulses’ amplitude, length and integral, see fig. 1.

The LabView interface, fpga\_dll.dll is written in C++ and interfaces to LabView vi’s via calls to external dll functions. For convenience and compatibility, these function calls are encapsulated in vi’s and should be called from users programs.

fpga\_init\_2.vi:

 This function initializes the communication protocol between the computer and the board. It returns a *handle* to the communication channel to be used for all subsequent operations with the board. On error, it reports an internal driver error number for debugging.

Inputs: none

Outputs:

device handle: FPGA handle to be used for communication.

error: error message.

SetFPGAThreshold.vi:

 This function communicates with the board and sets the threshold level. A threshold level is a positive integer T, T<32. It sets a threshold at the digital output that is equal to 8\*T. When input voltage exceeds threshold value, it triggers an event. The metrics (refer to Fig. 1) of pulse each event will be acquired and stored until getdata.vi is used to read the collected information.

NOTE: If FIFOs of FPGA were not empty, this function will return an array of collected information in the same way as getdata.vi would.

Inputs:

handle\_in: the output of fpga\_init.vi

Threshold level: Integer how many times the dll should query the board before returning the execution to labview. It is recommended to keep this number high enough to ensure that all real-time data makes it to the PC, but low enough so that the custom labview application keeps a comfortable update rate.

Data In: empty array to be filled with collected information.

Outputs:

handle\_out: “returns” the handle to the system. Can be used for sequential calls to this and other board vi’s.

Error: error message.

Dataout: array of data on events stored in FPGA.

getdata.vi:

 This function communicates with the board, and acquires data on previously recorded events that are stored in FPGA FIFOs. of events (correlations) and allows writing a raw file of all events to a hard disk for the purposes of further statistical analysis. Please refer to the appropriate section below for file format.

Inputs:

handle: the output of fpga\_init.vi

data: empty array to be filled with collected information.

Outputs:

handle 2: “returns” the handle to the system. Can be used for sequential calls to this and other board vi’s.

length: length of the received packet

data 2: an array of data about detected events. Each event is represented by its amplitude, length and integral, as illustrated in Fig. 1. Each event generates an 8-byte word. Refer to table 5 for byte designation. The example labview code contains a sub-vi “Data to Pulse.vi” that illustrates how to create three separate arrays for pulse amplitudes, lengths and integrals from a single array downloaded by getdata.vi.

Note that advanced users can write their own module that analyses the acquired events in real time. This function could replace the default in the driver. This procedure is described later.

Table 5: Information recorded about each event

|  |  |
| --- | --- |
| **Offset, bytes** | **Meaning** |
| 0 | Integral, byte 0 |
| 1 | Integral, byte 1 (x256) |
| 2 | Integral, byte 2 (x65536) |
| 3 | Integral, byte 3 (x16777216) |
| 4 | Length, byte 0 |
| 5 | Length, byte 1 (x256) |
| 6 | Length, byte 2 (x65536) |
| 7 | Amplitude |

fpga\_close\_2.vi:

This function closes the communication protocol.

Inputs: handle\_in

Outputs: none

**Real-Time Data Access (advanced)**

Users may want to develop a custom real-time data processing routine, or bypass LabView altogether. To do so, one needs a C/C++ compiler. Function \_declspec(dllexport) HANDLE FPGA\_Open() opens a communication session and returns a handle to FPGA hardware. This handle should be used for the duration of a communication session. To close a session, use   
\_declspec(dllexport) void FPGA\_Close(HANDLE XyloDeviceHandle).

The main input/ouput function that communicates with an FPGA is:

\_declspec(dllexport) int FPGA\_ANALOG\_Interface (int runs, char fpga\_command, int saveclicks, char \* filename, HANDLE XyloDeviceHandle, int \* length, unsigned char \* data)

On success, the function returns 1 and fills an array “data” with collected data. If an error has occurred, the function returns a negative value or zero.

The parameters are:

Runs: how many communication cycles does this function make before returning execution to the caller (LabView by default).

Fpga\_command: one byte communication from PC to FPGA, used to start, stop acquisition, and to set thresholds. Setting a threshold can be combined with a start or a stop acquisition commands.

Possible commands:

0b00000000 no command

0bxxxxx001 input a new threshold, where 0bxxxxx is a new threshold value. FPGA sets a threshold at 0bxxxxx000. See description of SetFPGAThreshold.vi for more information.

0b00000010 stop acquisition

0b00000100 start acquisition

Saveclicks: if set to 1, saves the raw output from FPGA to a file.

Filename: name of the file to save raw output if Saveclicks=1.

XyloDeviceHandle: handle to an FPGA device.

The outputs are:

Length: length of received data packet.

Data: pointer to the data array.

Function FPGA\_ANALOG\_Interface can be called from a user C/C++ program or a custom dll node of LabView.

Suggested code sequence for users program that communicates with an FPGA device is given below:

void main ( void )

{

//Initialize FPGA session

HANDLE fpga = FPGA\_Open();

//Check for initialization errors

if ( fpga == (HANDLE)-1 ) printf ( "!!!!!!!!!!!!! FPGA INIT ERROR" );

//set a threshold level to 4 0b00010001 and start data acquisition 0b00000100

//a combined command is 0b00010001+0b00000100=0b00010101

int res;

int length;

unsigned char data [65535];

res = FPGA\_ANALOG\_Interface (1, 0b00010101, 0, " ", fpga, length, data);

//check if FPGA data exchange was a success

if ( res != 1 ) printf ( "!!!!!!!!!!!!! ERROR %d", res );

//acquire data for as long as it is necessary

int stop=0;

while (!stop)

{

//send an empty command 0xb00000000 and recieve FPGA data

res = FPGA\_ANALOG\_Interface (1, 0b00000000, 0, " ", fpga, length, data);

//check if FPGA data exchange was a success

if ( res != 1 ) printf ( "!!!!!!!!!!!!! ERROR %d", res );

//use collected data in a user routine

stop=USER\_FUNCTION(length, data);

}

//stop data acqisition by sending 0b00000010

res = FPGA\_ANALOG\_Interface (1, 0b00000010, 0, " ", fpga, length, data);

//check if FPGA data exchange was a success

if ( res != 1 ) printf ( "!!!!!!!!!!!!! ERROR %d", res );

//if needed, use collected data in a user routine

USER\_FUNCTION(length, data);

//Close FPGA session

FPGA\_Close(fpga);

}