Simple and Inexpensive FPGA-based Fast Multichannel Acquisition Board

Technical Description, Installation and Operation Manual

Sergey Polyakov & Joffrey Peters

January 27, 2015

Abstract

This document describes how to build and operate an FPGA board to detect pulses and generate real-time coincidence statistics and/or generate time-resolved data of electrical pulses, such as these generated by Single-Photon Avalanche Detectors (SPADs). This board connects to a 32- or 64-bit PC via USB 2.0 and can be used with popular data acquisition programs, or custom software using the included dynamically linked library or open source code.

1 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. To implement even a simple photon-counting test bench, one needs to heavily invest in not only SPAD detectors, but also expensive photon counting hardware and software. The fact that most commercial solutions known to the author provide proprietary (as opposed to open source) software makes it difficult to adapt these solutions to custom needs, especially when real-time data processing is needed. The main principles of development of this board are:

- Easy assembly, installation and operation
- Low cost
- Open source software and FPGA firmware
- Simple connectivity with 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. This software was developed for 32- and 64-bit Windows PCs and for use with National Instruments' LabVIEW[1].

[1] Certain commercial equipment, instruments, or materials are identified in this report in order to specify the experimental procedure adequately. Such identification is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the materials or equipment identified are necessarily the best available for the purpose.

2 Description

The board is a Xylo-EM FPGA board is made by knjn.com. It detects "signal" TTL pulses on 4 channels and either timestamps their arrival based on an internal or externally provided clock, or records count and coincidence statistics. The timestamps or coincidence statistics are driven by either the board's own internal clock with 20.83 ns increments (i.e. the internal clock runs at 48 MHz) or can be driven by an external clock with increments down to 10.4 ns (i.e. a clock up to 96 MHz), which is sufficient to lock the pulse timestamping clock to the repetition rate of typical ps and fs Ti:Saph lasers. Additionally, the boards can output TTL pulses or toggle the TTL level of two output channels through software. As the board takes data, the user may query the board via USB 2.0 to retrieve the timestamped events, or event statistics. The peak transfer rate observed is > 8 MB/s (corresponding to $\gtrsim 2$ million counts per second on all four channels combined), but is only limited by the speed of the particular USB connection. The information is then processed and/or recorded to the hard drive by the driver and is available in real time to the end user, either through the dynamic-link library (DLL) which handles the USB connection, or via a LabVIEW interface. The number of time stamping channels can be increased by the user through modification of the FPGA firmware. The software was developed and tested using the Xylo-EM FPGA board which uses an Altera Cyclone II FPGA and a Cypress FX2 USB 2.0 chip. It also could be used on knjn's Saxo FPGA boards (with limited functionality). The Xylo-EM board is commercially available, costs less than \$200, and is the principal expense of the project.

The following tables list system requirements and board characteristics as well as a parts list to set up a Xylo-EM for use with this software.

Table 1: System requirements and board characteristics

Parameter	**Value**
Operating systems supported	32-bit: Windows 2000, XP, 7 64-bit: Windows 7
Computer interface	USB 2.0
Time counter reset (start) channel	1
Timestamping (stop) channels	4 (expandable as necessary with firmware modifications)
TTL output channels	2 (expandable as necessary with firmware modifications)
Timestamp increment (internal clock)	20.83 ns(48 MHz clock)
Minimal timestamp increment (external clock)	< 10.4 ns (> 96 MHz clock)
TTL pulse width	1 internal clock cycle, 20.83 ns
Board deadtime (after click event)	1 clock cycle after trigger
Minimum input signal duration	1 clock cycle
Event threshold level (start, stop, clock)	TTL ($\gtrapprox 1.6$ V, positive edge, non-adjustable)
Maximum USB transfer rate	> 13 MB/s or $\gtrapprox 3 \times 10^6$ counts/s total depending on computer-USB interface
Estimated total cost	$\approx$ \$240
Estimated assembly and installation time	4 hours

Table 2: Parts list with example links

Part	**Cost**
Xylo-EM FPGA board	$1 \times$ \$150
Generic electronics box) 3"x2"x6" (min)	$1\times \approx$ \$20
USB 2.0 cable	$1\times$ typically included with Xylo-EM purchase
Female BNC jacks	$8\times \approx$ \$3 = \$24 (or more, if adding channels)
Coaxial cable (RG-174)	$\approx$ \$0.75 ($\approx$ 3 ft)
Splitters and 50 Ω terminators	8 ea. $\approx$ \$42
Mounting screws (4-40) and standoffs	As needed ($\sim 2-3$ ea. standoffs/mounting screws & nuts)
Total	$\approx$ \$240

Detailed description

Events are recorded when voltages at the input channels are above threshold (> 1.6 V) at the rising edge of the clock (internal or external). This implies that the signal input should be high for at least a full clock cycle in order to reliably register. Further, due to the method of registering the input signals and locking them to an asynchronous clock, there is a one-clock-cycle dead time after a detection event. As many commercial APDs output signals of > 30 ns, this is not an issue for an external clock rate greater than 67 MHz, or experimental repetition rates below 16.6 MHz using the internal clock. However, an additional input pulse on the next clock cycle will not be distinguishable from a continued high input from a longer input pulse, such that input signals should be sparse compared to the clock rate in order for the pulses to be individually registered reliably. With firmware modifications, it is possible to run the internal clock at a higher rate. Double or more of this internal clock rate has been achieved by the authors.

An additional consequence of the clock registration is a delay in click registration and time-tagging. The clicks take 3 clock cycles to register with the time-tagger, which does not matter for most experiments, but for any requiring extreme speed, this implementation may be unsuitable.

Coincidences are registered if two channels are high during the rising edge of the same clock cycle. To extend this time, the user may take timestamped data and do post-analysis, or may augment the included FPGA firmware to extend the coincidence overlap. The former method is probably an easier place to begin if the user is not familiar already with Hardware Description Languages (in particular, Verilog).

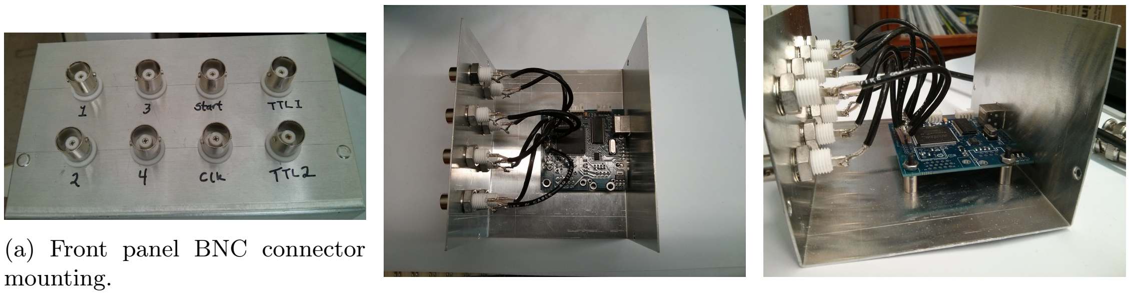
Timestamps are based on number of clock cycles since the last input at the Start pin, or Clear command issued to the FPGA. These are registered when the input channel is high at the rising edge of a clock, but the firmware could be augmented to change the registration of a click to when both the clock and input are high, though this was not done as it increases the timing jitter unnecessarily when the input signal is broader than one clock cycle, as is commonly the case for commercial APDs.

3 Assembly and installation

3.1 Assembly

This manual is for the Xylo-EM FPGA board. For a different board, changes in firmware and assembly may be necessary. Following are the assembly steps to set up a Xylo-EM board for use with this software. It is important that all soldering and construction is done while the board is not powered (i.e. not connected to a PC or power source via the board's USB connector).

1. Mount the BNC connectors on the box as shown in Fig. 1a. Solder to the board with shielded cables, connecting the BNC jacks carefully to the board's ground pins, avoiding shorts.



(a) Front panel BNC connector mounting.

(b) Board mounting, top view.

(c) Board mounting, side view.

Figure 1: Board assembly.

2. Mount the FPGA test board to the box with standoffs and screws through the mounting holes as shown in Figs. 1b, 1c. Make sure that the USB connector is accessible from the outside of the box.

3. Solder BNC connectors to FPGA pins with short (3-4") segments of shielded (coaxial) cable. Make sure that all pieces are of the same length to reduce systematic timing issues. Refer to Table 3 and Fig. 2 for pin assignments.

Table 3: Pinout configuration. Note: pins 93, 104, 112, 113, 114, 132, 133 (shown in Fig. 2) could be also wired in a similar way for future extensions and/or for debugging.

Pin	**Purpose**
70	Detector 1 (stop)
73	Detector 2 (stop)
79	Detector 3 (stop)
92	Detector 4 (stop)
97	Clear counter (start)
88	External clock input
100	TTL 1 Output
101	TTL 2 Output

3.2 Connecting the board

1. If using an external clock, connect the clock to the Clk pin (pin 88). For clock-zeroing and synchronization, input a pulsed TTL signal to the Start pin (pin 97). Connect detectors or other inputs to the appropriate input channels listed in Table 3.

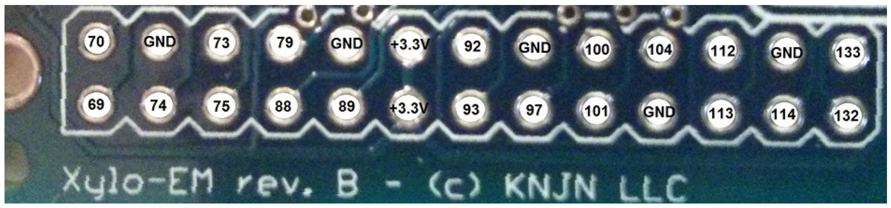


Figure 2: Xylo-EM Pinout. This information and a full board layout diagram can be found in the Xylo-EM documentation from knjn which comes with the board.

2. TTL channels 1 and 2 are connected to pins 100 and 101 respectively. The user may output independent toggleable TTL levels on the two channels, or independent a TTL pulses of 20.83 ns.

3. It is important to match the impedance to 50 Ω at least at one end of the connecting cable to avoid multiple reflections. Terminate all unused inputs on board with 50 Ω terminators, as shown in Fig. 3. For example, in Fig. 3, the source from Detectors 1 & 2 is TTL, and is thus 50 Ω terminated at the FPGA breakout for channels 1 & 2.

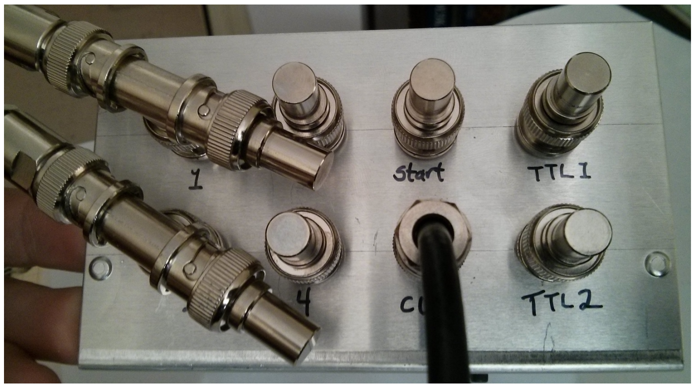


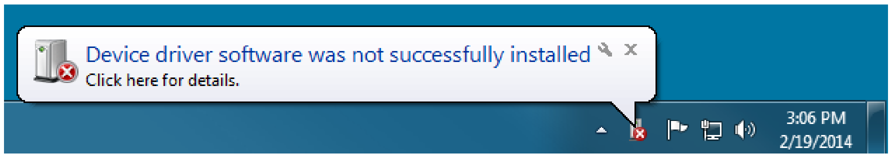
Figure 3: Proper 50 Ω BNC termination.

Note: some commercial detectors produce a digital output of < 3.5 V. If such detectors are used, termination at either end may bring the signal below the threshold voltage. One possible solution is to use a short (< 1 ft) cable between the detector and the FPGA, and avoid terminators. One could also use simple comparators, setting the comparison voltage to about 1 V and the output at TTL level. (If one really enjoyed using FPGA boards like these, one could use a stand-alone FPGA test board that simply translates input to output with a short cable connecting it to the detector, then connect the two FPGA boards via a terminated BNC cable.)

3.3 Software installation

1. Unpack software distribution. Unzip the board's startup kit package into a folder, e.g. `C:\FPGA`. Unpack the Stats FPGA software to the same folder.

2. Driver installation. When you plug the Xylo-EM board into your computer, it may not recognize the USB controller and board and you will get a message like that in Fig. 4.



Device driver software was not successfully installed
Click here for details.
3:06 PM
2/19/2014

Figure 4: Windows 7 computers may not recognize the USB controller on the Xylo-EM.

To rectify this issue, you must open the Device Manager (simply type `Device Manager` into Windows' `Start` menu to access this). This may require Administrator privileges. The Xylo-EM will show up as an unknown device as in Fig. 5.

Right-click this device and click `Update Driver Software...` to initiate a driver update wizard. In the wizard, select `Browse my computer for driver software` (Fig. 6), then the `Browse...` button (Fig. 7).

Navigate to the signed CyUSB driver folder at e.g. `C:\FPGA\ Driver - USB\ 3. CyUSB signed, Cypress GUID, USB ID 8613 only`. Select the `x64` folder for a 64-bit platform or the `x86` folder for a 32-bit platform (see Fig. 8).

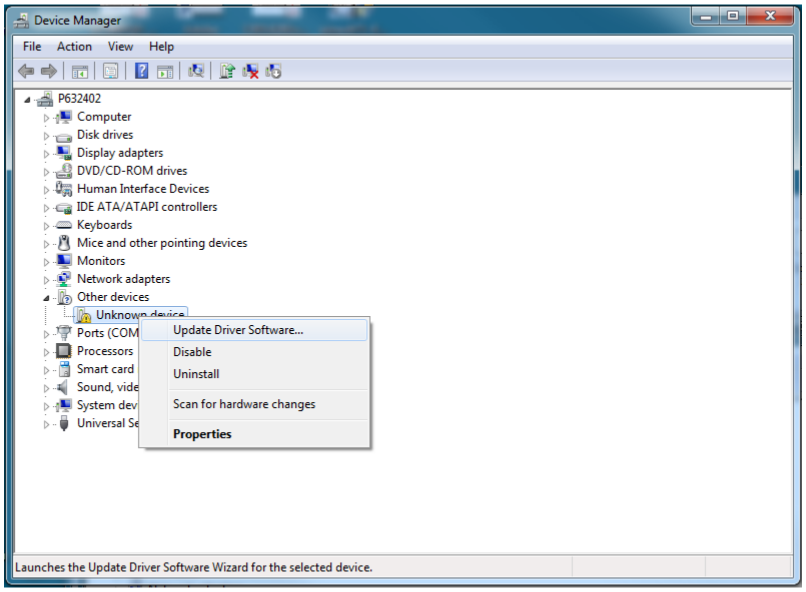
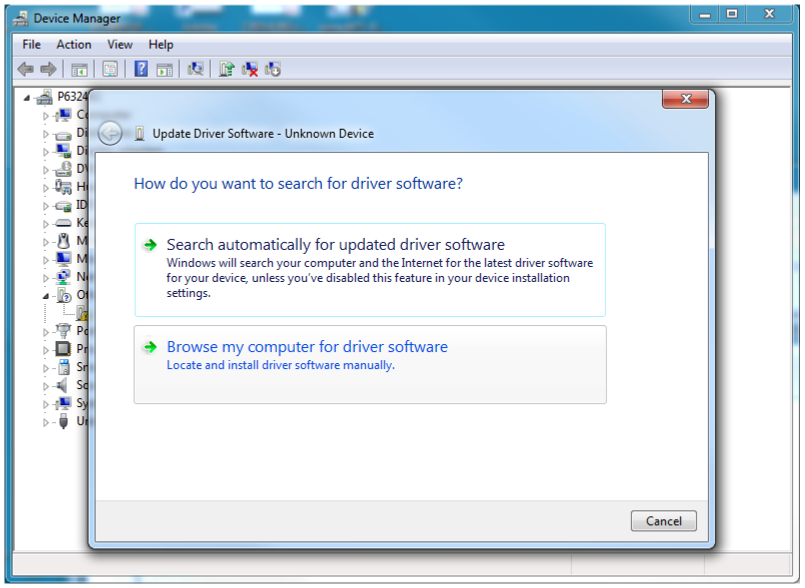


Figure 5: The Xylo-EM will be an unrecognized device in the Device Manager.



How do you want to search for driver software?

Search automatically for updated driver software
Windows will search your computer and the Internet for the latest driver software for your device, unless you've disabled this feature in your device installation settings.

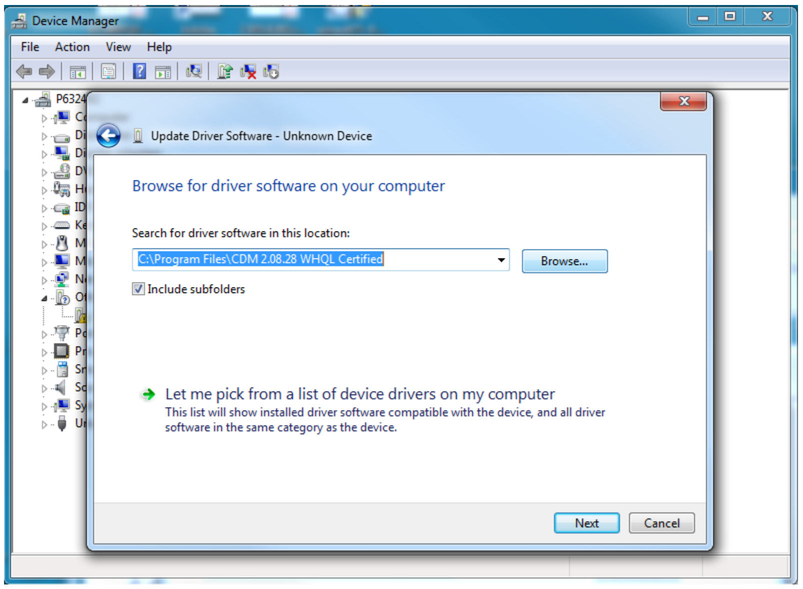
Browse my computer for driver software
Locate and install driver software manually.

Figure 6: Click `Browse my computer for driver software`.

Windows should now recognize the device as a `Cypress EZ-USB FX2LP No EEPROM(3.4.5.000)` and should show up under Universal Serial Bus controllers in the Device Manager.

3. Configuring the board. Open `FPGAconf.exe`, which is distributed along with the board's start-up kit (it would be in `C:\FPGA\` in our recommended folder structure). Select `Xylo-EM` as the target board from the `Boards` menu. Then, in the `Options` menu, set `FX2 clock speed` to `48 MHz` as in Fig. 9, set the `USB Driver` to `CyUSB`, the `USB device` to `0`, and the `CyUSB GUID` to `Cypress`.

   Click the `...` button to open a file selection dialog, then provide the path to the FPGA TimeTag firmware. The firmwares are the `.rbf` files in the appropriate subfolders in the e.g. `C:\FPGA\Stats\Firmware` folder. Each subfolder indicates what the firmware is for. There are four folders, one each for internal and externally clocked pulse time tagging (`TimeTag` as well as for counting and statistics of pulses



Update Driver Software - Unknown Device

Browse for driver software on your computer

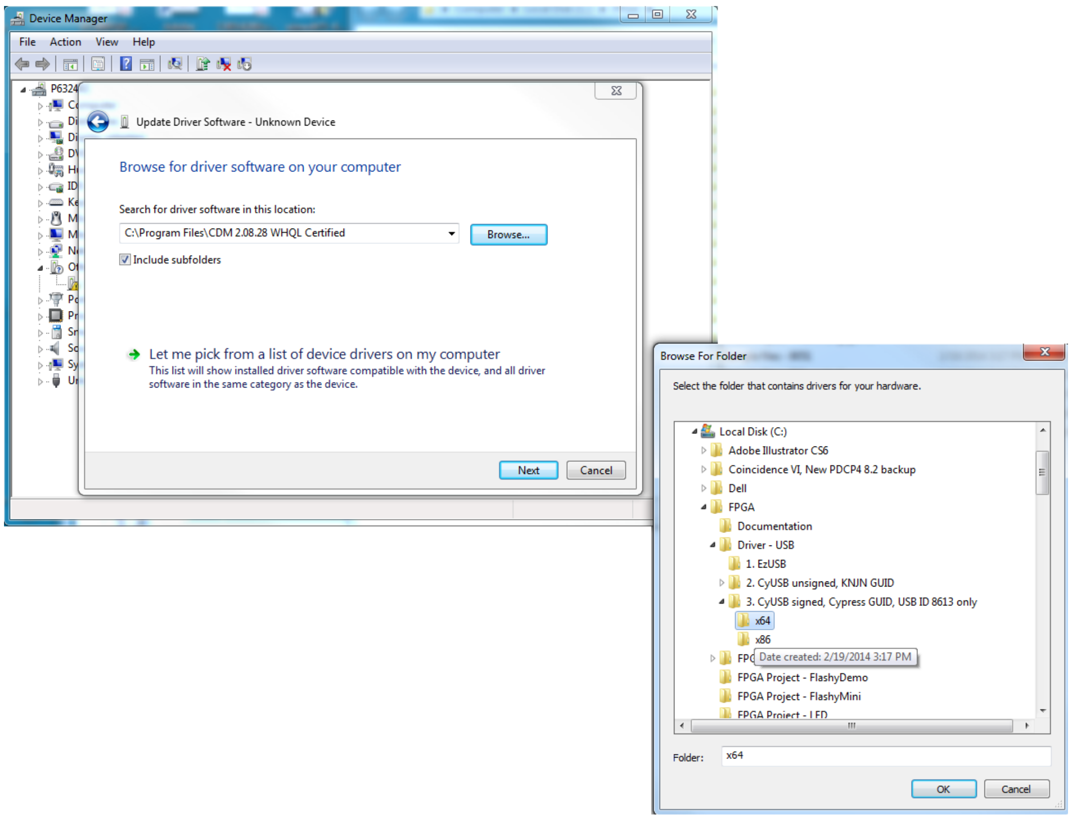
Search for driver software in this location:

C:\Program Files\CDM 2.08.28 WHQL Certified

☑ Include subfolders

Let me pick from a list of device drivers on my computer
This list will show installed driver software compatible with the device, and all driver software in the same category as the device.

Figure 7: Click `Browse...`.



Update Driver Software - Unknown Device

Browse for driver software on your computer

Search for driver software in this location:

C:\Program Files\CDM 2.08.28 WHQL Certified

☑ Include subfolders

Let me pick from a list of device drivers on my computer
This list will show installed driver software compatible with the device, and all driver software in the same category as the device.

Browse For Folder

Select the folder that contains drivers for your hardware.

```
Local Disk (C:)
  Adobe Illustrator CS6
  Coincidence VI, New PDPC4 8.2 backup
  Dell
  FPGA
    Documentation
    Driver - USB
      1. EzUSB
      2. CyUSB unsigned, KNJN GUID
      3. CyUSB signed, Cypress GUID, USB ID 8613 only
        x64
        x86
    FPG  Date created: 2/19/2014 3:17 PM
    FPGA Project - FlashyDemo
    FPGA Project - FlashyMini
    FPGA Project - LED
```

Folder: x64

Figure 8: Browse to the driver folder and select the folder appropriate for your platform.

(`Counts`). Next, click the `Configure` button to configure the FPGA with the selected firmware. A beep indicates the successful completion of this step and indicates that the FPGA board is ready to be used in the selected (`TimeTag` or `Counts` internally or externally clocked) mode.

4. Testing the installation. The simplest way to test the installation is to use the LabVEW example project

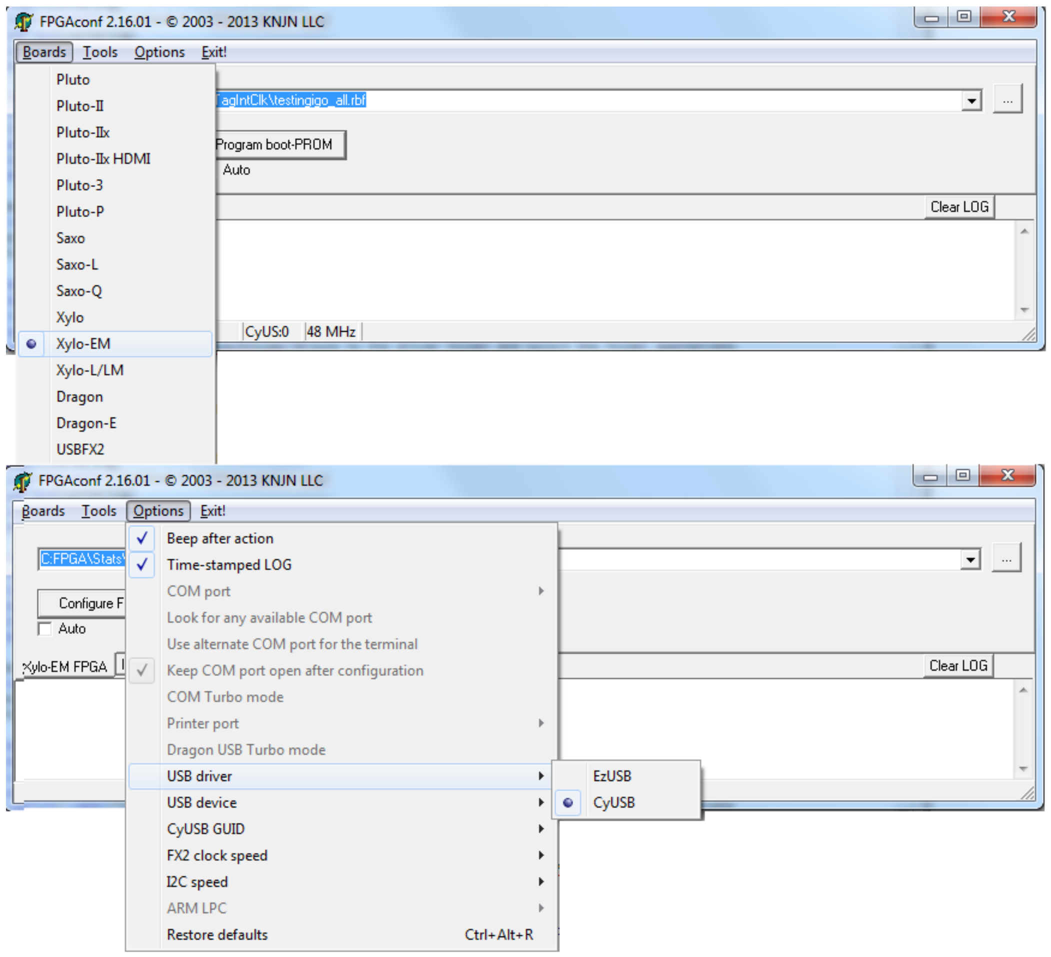
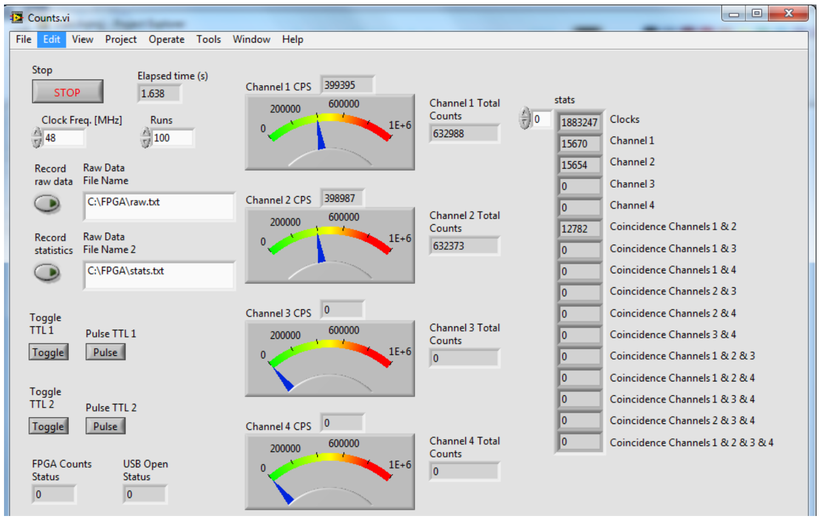


Figure 9: Step 2 of software installation process. Setting up the configuration program.

`Stats.lvproj` in the `Stats\StatsLV\` folder provided with this software package. Use an internally clocked firmware and be sure to use `TimeTag` firmware with `TimeTag.vi` or the `Counts` firmware with the `Counts.vi`. Provide TTL inputs to the Channels 1-4 to see that pulses are counted. Connect an oscilloscope to TTL outputs 1 & 2, and check that pulsing and toggling the TTL channels produces the expected output. The receipt and initial processing of data are handled by a stand-aloneDLL. This test is successful if the electronic events statistics is correctly displayed by the VI. Sample figures of `Counts.vi` and `TimeTag.vi` properly registering counts on channels 1 and 2 are shown in Fig. 10. More information on using these example programs is given in the next section.

Note: If you do not have a current redistributable for Microsoft's Visual C++ installed, you may get an error message at some point, indicating that `MSVCP110DLL` is not installed. To fix this problem, search the internet for "Visual C++ Redistributable for Visual Studio 2012". Download the file from www.microsoft.com only. At the time of writing, Update 4 was the current version and the file could be downloaded at the following web address: http://www.microsoft.com/en-au/download/confirmation.aspx?id=30679.

If you do not get the `MSVCP110DLL` error message, but your LabVIEW test fails, unplug the USB cable, wait for 10-20 seconds and plug it back in. Repeat steps 2 and 3. If this does not help, the PC may need to be restarted. Repeat steps 2 and 3. If test fails again, check all the connections and that the device is recognized properly in the Device Manager (We recommended using commercial function generators instead of real signal sources for debugging.



Stop

Elapsed time (s): 1.638

Clock Freq. [MHz]: 48

Runs: 100

Record raw data

Raw Data File Name: C:\FPGA\raw.txt

Record statistics

Raw Data File Name 2: C:\FPGA\stats.txt

Toggle TTL 1 · Pulse TTL 1

Toggle TTL 2 · Pulse TTL 2

FPGA Counts Status: 0

USB Open Status: 0

Channel 1 CPS: 399395 · Channel 1 Total Counts: 632988

Channel 2 CPS: 398987 · Channel 2 Total Counts: 632373

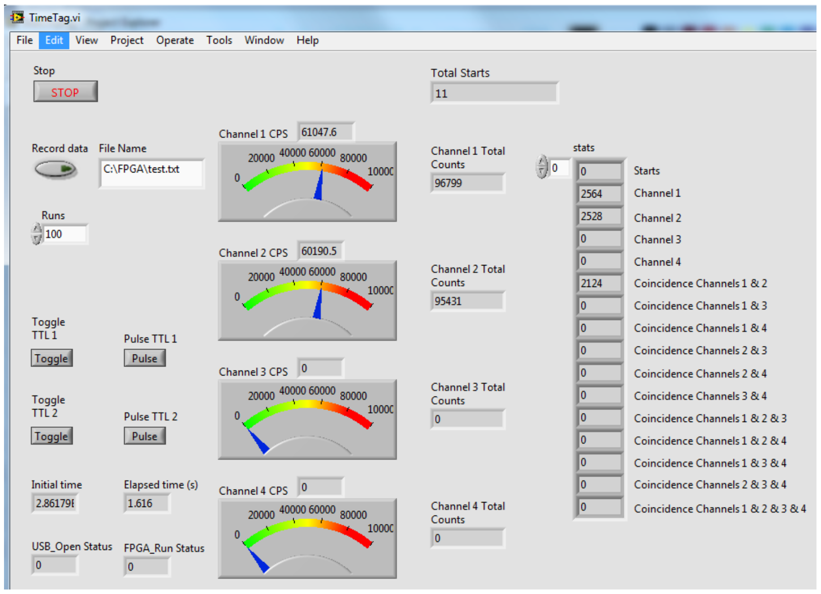
Channel 3 CPS: 0 · Channel 3 Total Counts: 0

Channel 4 CPS: 0 · Channel 4 Total Counts: 0

stats: 0

1883247	Clocks
15670	Channel 1
15654	Channel 2
0	Channel 3
0	Channel 4
12782	Coincidence Channels 1 & 2
0	Coincidence Channels 1 & 3
0	Coincidence Channels 1 & 4
0	Coincidence Channels 2 & 3
0	Coincidence Channels 2 & 4
0	Coincidence Channels 3 & 4
0	Coincidence Channels 1 & 2 & 3
0	Coincidence Channels 1 & 2 & 4
0	Coincidence Channels 1 & 3 & 4
0	Coincidence Channels 2 & 3 & 4
0	Coincidence Channels 1 & 2 & 3 & 4

(a) `Counts.vi` registering a 50% duty cycle 400 kHz TTL signal.



Stop

Total Starts: 11

Record data

File Name: C:\FPGA\test.txt

Runs: 100

Toggle TTL 1 · Pulse TTL 1

Toggle TTL 2 · Pulse TTL 2

Initial time: 2.86179E

Elapsed time (s): 1.616

USB_Open Status: 0

FPGA_Run Status: 0

Channel 1 CPS: 61047.6 · Channel 1 Total Counts: 96799

Channel 2 CPS: 60190.5 · Channel 2 Total Counts: 95431

Channel 3 CPS: 0 · Channel 3 Total Counts: 0

Channel 4 CPS: 0 · Channel 4 Total Counts: 0

stats: 0

0	Starts
2564	Channel 1
2528	Channel 2
0	Channel 3
0	Channel 4
2124	Coincidence Channels 1 & 2
0	Coincidence Channels 1 & 3
0	Coincidence Channels 1 & 4
0	Coincidence Channels 2 & 3
0	Coincidence Channels 2 & 4
0	Coincidence Channels 3 & 4
0	Coincidence Channels 1 & 2 & 3
0	Coincidence Channels 1 & 2 & 4
0	Coincidence Channels 1 & 3 & 4
0	Coincidence Channels 2 & 3 & 4
0	Coincidence Channels 1 & 2 & 3 & 4

(b) `TimeTag.vi` registering a 50% duty cycle 60 kHz TTL signal.

Figure 10: Examples of the included LabVIEW programs operating.

4 Operation

There are several default modes of operation. The operation mode can be changed by loading the board with an appropriate pre-compiled firmware (`.rbf` file). Using the FPGA Configurator application, select the `.rbf` file corresponding to the mode of operation of choice and click "Configure FPGA" button. This procedure must be done after every off-on power cycle to the board. The following table gives default modes of operation for TimeTag operation. Advanced users are encouraged to make custom firmware modifications and share them with the project core, however, custom firmware is not supported by this project.

Note that because of the ability to use an external clock, this board can easily synchronize to an experiment, thus reducing timing jitter. In some cases, the sync signal produced by the electronics is not TTL, and must therefore be converted to TTL before it may be used as a clock for this FPGA kit.

With the appropriate firmware uploaded to the Xylo-EM board, you may use the LabVIEW software included, or create your own software in LabVIEW or using the included DLL and your favored programming

Table 4: Available operation modes. Load the `.rbf` file in the given folder to load the firmware.

Operation mode	**Folder**
Counts, Internal clock	`\Stats\Firmware\CountIntClk\`
Counts, External clock	`\Stats\Firmware\CountExtClk\`
TimeTag, Internal clock	`\Stats\Firmware\TimeTagIntClk\`
TimeTag, External clock, doubled	`\Stats\Firmware\TimeTagExtClk\`

language. We suggest starting with the LabVIEW examples provided with this project. These use of these programs and the LabVIEW interface is outlined below.

The two example programs `Counts.vi` and `TimeTag.vi` may be found in the `Stats.lvproj` project in the `\Stats\StatsLV\` folder. Each program initiates a USB connection with the Xylo-EM board when started, then proceeds to query the FPGA in the counting/statistics or time-tagging mode, displaying certain elements of the retrieved data to the user at a user-specified interval. The count rates on each input channel (1-4) are displayed both numerically and on needle displays. Total counts and correlation statistics are shown in both programs on the right (see the examples in Fig. 10). In the Counts program, this data is generated by the FPGA in real time, while for TimeTag, the time-tagged data is returned to the DLL where it is parsed and the statistics are displayed to the user in LabVIEW. Data-recording is available to the user by toggling Record Data buttons and entering the full path and file name of a file in which the user wishes to store data. The data format of the click and correlation statistics as well as of the saved files is detailed in later sections.

The Stop button stops the program from querying the FPGA, stops the FPGA from taking data, and gracefully retrieves the last click information from the FPGA before stopping entirely. Pulse and Toggle commands for TTL channels 1 and 2 are available as buttons in each program. The `Runs` input is an averaging feature which tells the DLL how many times to query the FPGA before reporting the sum of the results to the LabVIEW interface. This was implemented because the LabVIEW interface is typically slower than the USB query rate, and reducing the data rate to the visual interface helps keep LabVIEW smooth and reduces jitter in output results due to jitter in USB query timings (USB queries are handled asynchronously by the operating system).

5 LabVIEW interface

In this section we describe the building blocks of the included LabVIEW programs so that the user may create her own programs using the functionality of the FPGA counter or time stamper. The basic structure of the LabVIEW code consists of wrapper classes which form an interface to the underlying DLL functionality. The LabVIEW project `Stats.lvproj` includes simple examples of operating the board in both of its main modes: counting/statistics in `Counts.vi` and time-tagging with `TimeTag.vi`. These examples cover all important stages of communication, and can be used for various design extensions. Both examples additionally allow the user to pulse or toggle the two TTL output channels independently. We describe the `VIs` that underlie this functionality.

The LabVIEW interface consists of a set of LabVIEW wrapper libraries. The `StatsLib.lvlib` library is the users' main interface to the underlying DLL functionality. It detects whether the user is on a 64- or 32-bit platform and selects the appropriate LabVIEW libraries which in turn wrap the functionality contained in `StatsDLL` (or `Stats32DLL` on a 32-bit platform). The DLL files were written in C++ and compiled with Microsoft's Visual Studio C++ Compiler. These contain the functionality for interfacing with the FPGA board via USB. The user need not access this code or the DLL files directly - the LabVIEW interface was designed to be the primary interface for most users.

For those interested in expanding or augmenting the functionality of the software, the DLL file and C++ source code are also included in the software bundle. The C++ code contains relatively useful comments to help an advanced user modify the code as she pleases. The user is encouraged to explore the simple example `VIs` to see how they work before delving deeper into the code.

USB Open.vi:

This function initializes the USB communication protocol between the computer and the board. Call this `.vi` before issuing commands to the FPGA. `USB Open.vi` returns an integer which encodes the status. A `0` indicates correct operation, while a `-1` indicates that the USB connection was already initiated. This is handled by an error handler which outputs an error message describing the error.

- **Inputs**: none
- **Outputs**:

  `function return`: an integer giving the status of the output. `0` indicates no errors, while negative values indicate certain errors.

  `error out`: error status including error message if an error occurred.

USB Close.vi:

This function closes the USB communication protocol, but leaves the mode of the board (acquisition, idle) unchanged. It is recommended to switch the board to idle by calling the `FPGA Counts.vi` or `FPGA TimeTag.vi` with the appropriate command before calling `USB Close.vi`. This is demonstrated in the sample programs in `Stats.lvproj`. An integer encoding the status is returned. A `0` indicates correct operation, while a `-2` indicates that the USB connection was never initiated. This is handled by an error handler which outputs an error message describing the error.

- **Inputs**: none
- **Outputs**:

  `function return`: an integer giving the status of the output. `0` indicates no errors, while negative values indicate certain errors.

  `error out`: error status including error message if an error occurred.

FPGA Counts.vi:

This VI communicates with the board, acquires click event statistics from each channel as well as the number of clock cycles elapsed and coincidence statistics. This data may be saved to the hard disk for the purposes of further statistical analysis. Additionally simple correlation statistics are output for the user to employ. The file format is described in Sec. 6 below.

- **Inputs**:

  `runs`: integer number of USB queries the DLL should perform before returning the results to this LabVIEW `VI`. It is recommended to keep this number high enough to ensure that the LabVIEW interface isn't jittery, but low enough that internal buffers in the FPGA don't overflow with data (this would take a *lot* of data - over 2^{48} clocks, or 2^{32} counts on inidividual channels, or 2^16 four-fold coincidences).

  `saveData`: boolean telling the DLL whether or not to save the data to file.

  `fileName`: string giving the full file name and path of the file in which data should be stored. If a file with the same name exists, data will be appended to that file.

  `fpgaCommand`: An unsigned byte containing a character representing a command to the FPGA. The commands accepted are detailed in Table 5.

  `error in`: error signal from previously executed `VI`s.

- **Outputs**:

`stats` a 16-element array containing click and coincidence counts accrued since the last USB communication with the FPGA. The array contains the number of clock cycles over which the data were collected, clicks on each channel, and the various coincidences between channels. The particular format of these data are detailed in the Sec. 6.

`data` the raw data containing the above click statistics in a more dense format (32 bytes), which is described in Sec. 6 below.

`function return`: an integer giving the status of the output. `0` indicates no errors, while negative values indicate certain errors.

`error out`: error status including error message if an error occurred.

FPGA TimeTag.vi:

This VI communicates with the board, acquires time-tagged click events from each channel and allows writing a data file of all events to the hard drive for the purposes of further statistical analysis. Additionally simple correlation statistics are output for the user similarly to the output of `Counts.vi`. The file format is described in Sec. 6 below.

- **Inputs**:

  `runs`: integer number of USB queries the DLL should perform before returning the results to this LabVIEW `VI`. It is recommended to keep this number high enough to ensure that the LabVIEW interface isn't jittery, but low enough that internal buffers in the FPGA don't overflow with data (this takes a *lot* of data).

  `saveData`: boolean telling the DLL whether or not to save the data to file.

  `fileName`: string giving the full file name and path of the file in which data should be stored. If a file with the same name exists, data will be appended to that file.

  `fpgaCommand`: An unsigned byte containing a character representing a command to the FPGA. The commands accepted are detailed in Table 5.

  `error in`: error signal from previously executed `VI`s.

- **Outputs**:

  `stats` a 16-element array containing click and coincidence counts during the time since the last USB communication with the FPGA. The array contains starts, clicks on each channel, and the various coincidences. The particular format of these data are detailed in the Sec. 6 section.

  `function return`: an integer giving the status of the output. `0` indicates no errors, while negative values indicate certain errors.

  `error out`: error status including error message if an error occurred.

FPGA Toggle.vi:

This VI communicates with the board and toggles one or both of the TTL output channels. The `fpgaCommand` inputs to toggle may also be included with the `fpgaCommand`s issued with `Counts.vi` or `TimeTag.vi`. The unsigned byte coding for a TTL toggle is `32` (`0x20`) as shown in Table 5..

- **Inputs**:

  `fpgaCommand`: An unsigned byte containing a character representing a command to the FPGA. This is expected to be only the channel to toggle. The commands and the TTL channels to which they correspond may be found in Table 5. The command for the toggle itself is internally added by the DLL. Add the two commands ($64 + 128 = 192$) to toggle both channels at once.

  `error in`: error signal from previously executed `VI`s.

- **Outputs**:

  `function return`: an integer giving the status of the output. `0` indicates no errors, while negative values indicate certain errors.

  `error out`: error status including error message if an error occurred.

FPGA Pulse.vi:

This VI communicates with the board and pulses one or both of the TTL output channels. The `fpgaCommand` inputs to pulse may also be included with the `fpgaCommand`s issued with `Counts.vi` or `TimeTag.vi`. The unsigned byte coding for a TTL pulse is `16` (`0x10`) as shown in Table 5. If the TTL level on the pulsed channel is high when the pulse command is issued, the pulse will only cause the TTL level to toggle low.

- **Inputs**:

  `fpgaCommand`: An unsigned byte containing a character representing a command to the FPGA. This is expected to be only the channel to pulse. The commands and the TTL channels to which they correspond may be found in Table 5. The command for the pulse itself is internally added by the DLL. Add the two channels ($64 + 128 = 192$) to pulse both at once.

  `error in`: error signal from previously executed `VI`s.

- **Outputs**:

  `function return`: an integer giving the status of the output. `0` indicates no errors, while negative values indicate certain errors.

  `error out`: error status including error message if an error occurred.

Table 5: Available FPGA commands and the functions which accept the commands.

FPGA Command	**Description**	**Accepting functions**
0	No change	`FPGA TimeTag.vi`, `FPGA Counts.vi`, `FPGA Toggle.vi`, `FPGA Pulse.vi`
1	Clear timestamp clock	`FPGA TimeTag.vi`
2	Disable detection	`FPGA TimeTag.vi`, `FPGA Counts.vi`
4	Enable detection	`FPGA TimeTag.vi`, `FPGA Counts.vi`
8	Get Data	`FPGA Counts.vi`
10	Pulse	`FPGA TimeTag.vi`, `FPGA Counts.vi`
20	Toggle	`FPGA TimeTag.vi`, `FPGA Counts.vi`
40	TTL Ch. 1	`FPGA TimeTag.vi`, `FPGA Counts.vi`, `FPGA Toggle.vi`, `FPGA Pulse.vi`
80	TTL Ch. 2	`FPGA TimeTag.vi`, `FPGA Counts.vi`, `FPGA Toggle.vi`, `FPGA Pulse.vi`

The above `VI`s have unsigned byte `fpgaCommand` inputs. These may be combined to issue several of the above commands at the same time. For example, any of the pulsing or toggling commands may be issued with the `fpgaCommand`s used for `Counts.vi` or `TimeTag.vi`. To do this, add the integer arguments and pass that value. For example, to take data with `Counts.vi` and toggle TTL channel 2 at the same time, the `fpgaCommand` would be: $4 + 8 + 32 + 128 = 172$.

6 Data Format

Clicks registered by the FPGA are communicated to the user in several ways. The data available to the user using `FPGA Counts.vi` or the example program `Counts.vi` which demonstrates the Counts functionality consists of an array containing click and coincidence clicks. The array `stats` array returned to the user is a 16 element array with the entries described in Table 6. The raw data `data` returned to the user of Counts has the same basic format as that in Table 6, but is stored in a 64-byte unsigned character string. The first array element (element 0) in Table 6 is represented by the first 6 bytes of the `data` output of `FPGA Counts.vi`. The subsequent 14 array elements representing singles, two-fold, and three-fold coincidence counts consecutively take up 4-bytes of `data` each, leaving the last 2 of the 64 bytes to represent the four-fold coincidences. This limits the number of counts of the different types the maximum unsigned integer value for the data sizes indicated in the third column of Table 6. The reason for more significant digits available for clocks and starts, and fewer for 4-fold coincidences is based on an assumption about the forms of typical data, in which there are fewer events than clocks, and fewer coincidences than singles, with particularly few 4-fold coincidences. This raw data is converted into the 16-element array output `stats` in the DLL and is passed to the user for convenience.

The sample program `Counts.vi` gives the user two options for saving data. Pressing the `Record statistics` button saves the `stats` output from each transaction with the DLL (representing `runs` USB queries and results from the FPGA) as a new line in the file at the specified path. Thus, each line in this file will have the data format shown in Table 6. Pressing the `Record raw data` button however records the data from each USB transaction separately as 8 lines of 8 bytes each separated by spaces for each USB transaction which each returns 64 bytes of data (in the format detailed above).

Table 6: Data Format of `stats` output. The 0th array element contains either the number of clocks in the returned set of statistics from `FPGA Counts.vi` or the number of Start triggers in this data set from `FPGA TimeTag.vi`. The third column gives the number of bytes dedicated to storing these events in the `Counts` project.

Array Element	**Channels**	**Bytes of data**
0	Clocks or Starts	6
1	Ch. 1 singles	4
2	Ch. 2 singles	4
3	Ch. 3 singles	4
4	Ch. 4 singles	4
5	Ch. 1 & Ch. 2 coincidences	4
6	Ch. 1 & Ch. 3 coincidences	4
7	Ch. 1 & Ch. 4 coincidences	4
8	Ch. 2 & Ch. 3 coincidences	4
9	Ch. 2 & Ch. 4 coincidences	4
10	Ch. 3 & Ch. 4 coincidences	4
11	Ch. 1 & Ch. 2 & Ch. 3 coincidences	4
12	Ch. 1 & Ch. 2 & Ch. 4 coincidences	4
13	Ch. 1 & Ch. 3 & Ch. 4 coincidences	4
14	Ch. 2 & Ch. 3 & Ch. 4 coincidences	4
15	Ch. 1 & Ch. 2 & Ch. 3 & Ch. 4 coincidences	2

The data made available to the user from `FPGA TimeTag.vi` or from within the sample program `TimeTag.vi` is exposed in two forms. The raw data can be stored in a file by clicking the `Record data` button in the `TimeTag.vi` program. Each line represents a recorded, time-tagged event and is broken into two pieces. The first is a 5-digit integer indicator showing which channels were high in this event (which channels had clicks), and whether or not a Start event was registered. A breakdown of the meaning of the pseudo-binary codes is given in Table 7. For coincidences this integer will simply show `1`s for each of the registered channels, for example, a coincidence on channels 1 & 3 without a Start event will give the output `00101`. The Start bit is high when the user inputs a TTL pulse on the Start channel (pin 97), or when the internal clock resets, which happens every ≈ 2.796 s when using a 48 MHz. The second piece of each line is an integer representing the number of clock cycles since the last Start event.

Table 7: Pseudo-binary click registration format

Hexadecimal:	**Meaning**
10000	Start
01000	Ch. 4
00100	Ch. 3
00010	Ch. 2
00001	Ch. 1

Additionally, these timetagged data are parsed by the DLL and singles and coincidence statistics are reported to the user through the `stats` output. The format of these statistics are identical to those output by `FPGA Counts.vi`, with the exception of the first element of the array, which gives the number of Start events since the last USB transfer of data, rather than the number of clock cycles elapsed.

6.1 Advanced time-tagging data access

Advanced users may wish to store the time-tagged data in some other format, or do some processing on the data within the DLL library by changing the included C++ source code. In this case, consult Table 8 for the format of the 4-byte time-tagged events as they are returned by the FPGA. The user may also enable direct writing of this binary format to file by un-commenting out the relevant lines of the C++ code, and commenting out the output format currently in use, then re-compiling the DLL for use.

The data returned by the FPGA is in little-endian format, as is used on `x86` platforms. Thus, Byte 0 is the lowest byte and consists of bits 7 to 0, and Byte 3 is the most significant byte and consists of bits 31 to 24 (see Table 8). After the data file is written, the user may perform post-processing on the data.

Table 8: Raw Timetagged Data Format

Bit:	26-0	27	28	29	30	31
Meaning:	Timestamp	Ch. 1	Ch. 2	Ch. 3	Ch. 4	Counter cleared

Some users may wish to develop custom real-time data processing routines. To do so, the user will need to augment the included C++ code. The Visual Studio Express 2012 project used to compile the DLL is included. We advise using Visual Studio Express 2012 or newer, as it includes a 64-bit compiler, and the CyUSB libraries used for the USB communication with the Cypress FX2 chip on the Xylo-EM board were compiled with a Microsoft Compiler and use Windows-specific libaries (specifically `Setupapi.lib`). Additionally, users will need to download the Cypress CyUSB development kit `Cypress Suite USB 3.4.7` in order to gain access to the `CyUSB.lib`, and `CyUSB.h` files. The simplest way to access real-time data is for the user to write a function which substitutes for the default coincidencecalculation function. This function is called within the `FPGA_TimeTag()` function in the DLL, and thus could be called after recompilation without changing any of the externally accessible function signatures, allowing complete compatibility with the existing LabVIEW interface. To do this, the user should write a custom function, to replace the function `void correlate (unsigned char* data, int length, __int64 *stats)`. The function may have any other name, say `foo()`, but must have the same signature to avoid additional modifications to the code. The user may then simply change the `#define REALTIME_FUNCTION correlate` statement to point to their custom function name, in the example, `#define REALTIME_FUNCTION foo`. Of course, more extensive changes to the code can be made.

7 Further improvements (roadmap)

A variety of other features and modifications could be made to this project. Some faster internal clock versions of the code have been made, and this may be pushed much further, as the expected maximum frequency of the board is 396 MHz which corresponds to a potential 2.52 ns timestamping interval. A relatively straightforward change would be to increase the number of input channels to take advantage of the number of output pins from the Xylo-EM. There is an SDRAM chip on the Xylo-EM which could be used as an additional buffer to increase the amount of data the FPGA could store between transfers, and potentially to increase the net USB transfer rate.

Additionally, we are open to working to help implement other firmwares to help support interesting scientific projects.