

OpenPET User's Guide Documentation

Release 2.0

OpenPET

October 30, 2015

1	Abbreviations	1
2	OpenPET v2.0 Release Notes	3
2.1	What's New	3
2.2	Upgrading from Previous Releases of OpenPET	6
3	System Overview	7
3.1	Definitions	8
3.2	System Configuration	11
3.3	Timing & Timing Signals	15
3.4	Firmware & Software Structures	16
4	Getting Started - <i>Small System Oscilloscope Mode</i>	19
4.1	Getting Hardware	19
4.2	Assembling the Hardware	19
4.3	Downloading the Software and Firmware	22
4.4	Running OpenPET System	32
4.5	Example System Setup & Data Acquisition of a Small System in Oscilloscope Mode	34
4.6	Example System Setup & Data Acquisition of a Small System in Singles Mode	36
4.7	Data Analysis	36
4.8	Getting Help	44
5	Detector Board	45
5.1	Bus IO	46
5.2	16-Channel Detector Board	48
6	Support Board	53
6.1	Support Board with Detection Firmware	53
6.2	Support Board with Coincidence Firmware	56
6.3	Support Board with Detection & Coincidence Firmware	56
7	Commands	59
7.1	Description and Examples of Command IDs	65
8	OpenPET Control and Analysis Tools (OpenPET CAT)	79
8.1	Installing ROOT	79
8.2	Using OpenPET CAT	79
8.3	OpenPET CAT Graphic User Interface	82

9	Acknowledgements	103
10	Appendices	105
10.1	Appendix 1: Default Values	105
10.2	Appendix 2: Troubleshooting Diagnostics	107
11	Errata	109
11.1	Firmware	109
11.2	Embedded Software	109
11.3	Software	109
11.4	Hardware	109
12	Index	111

Abbreviations

CDUC: Coincidence Detector Unit Controller
CI: Coincidence Interface Board
C/R: Commands and Responses
CU: Coincidence Unit
CUC: Coincidence Unit Controller
DB: Detector Board
DST: Destination
DU: Detector Unit
DUC: Detector Unit Controller
EPCS: Enhanced Programming Configuration Serial device
FIFO: First-in, First-out Data Buffer
FPGA: Field-Programmable Gate Array
MB: Multiplexer Board
MBC: Multiplexer Board Controller
PLL: Phase-Locked Loop
SB: Support Board
SRC: Source
TDC: Time-to-Digital Converter

OpenPET v2.0 Release Notes

2.1 What's New

2.1.1 Firmware

- Complete rewrite from scratch.
- All code is revision controlled and reviewed in Bitbucket.
- Unified architecture for all FPGAs and boards.
- Minimal NIOS II Microprocessor on all FPGAs.
- Standard 32-bit SPI bus for inter-module communication. A OSI model like architecture is used.
- New clocking scheme on all FPGAs. e.g. Single PLL based clocking on all FPGAs with global reset in all modules.
- Reusable and modular building blocks on all FPGAs. e.g. SoftwareFirmware Interface module, Cross Clock Domain modules, Processing Unit module, etc.
- Utilize Double-Data-Rate high-speed communication for data path through backplane.
- Generic command path design. e.g., allow an arbitrary command length instead of 80-bits.
- Data is pushed from the lowest child to uppermost parent. Uppermost parent can throttle the speed of the acquisition based on the medium used, i.e., QuickUSB or Ethernet.
- Multistage synchronizers to streamline clock-domain-crossing between different boards, FPGAs, and components.
- New Software-Firmware Interface to allow simple SPI Master/Slave communications.
- SPI interface is based on Altera's standard SPI core.
- Simplified directory structure and file names.
- Renamed FPGA IOs to have descriptive names.
- All firmware modules are organized for readability and flexibility. e.g. no stray code.
- New Generic SRAM interface.
- Firmware image creation is automated.
- LEDs are now properly connected.
- All modules use the rising edge of the clock, with global reset logic.

- Slice is a clock (not a pulse) with a period of 8 System Clock cycles or 16.
- New commands. Previous commands are no longer valid and should not be used.
- New modes: IDLE, SCOPE, and SINGLES. In IDLE mode, the firmware does nothing. In SCOPE mode the firmware collects raw ADC data and saves it on external storage e.g. Workstation. In SINGLES mode the firmware processes the raw ADC data (current SINGLES example computes the energy i.e., area under the curve) and saves it on an external storage e.g. Workstation.
- New settings, each mode above has its own 32-bit settings.
- New action, each mode above has its own 32-bit action. Currently there are three actions. RESET, RUN, and STOP.
- Synopsys Design Constraint (SDC) files are extensively used to specify the timing and area constraints of the design.
- No timing violations.
- All code is written in VHDL 1993 standard. `numeric_std` vhdl package is used. No proprietary packages are used.
- Simple statemachines for all Scope and Singles modes.

SupportBoard

- Generic design. i.e. (a) compile time parameters to enable/disable components and modules e.g. `g_en_osc_mode` and `g_en_sng_mode`, (b) variable FIFO depths e.g. `g_DATA_FIFO_DEPTH`, different detector boards e.g. `g_adc_channels`, `g_adc_number`, and `g_adc_resolution`, (c) different slice widths e.g. `g_slice_div`, (d) optional debugging e.g. `g_debug`.
- JTAG-UART is enabled to display debugging info for additional verbosity.
- New QuickUSB interface which supports both Host Interface Boards and SupportBoard connectors.
- Faster Scope mode push architecture.
- Better timing and synchronization due to the use of PLL based clocking and resets. A single PLL clocks all parts of the OpenPET platform including the external QuickUSB module.
- Unified code for IO FPGAs. Both FPGAs share the same code base now. No need for two separate projects.
- Generic queuing algorithm is implemented (same code is used on main and io FPGAs). The queue lines up SCOPE mode data from multiple detector boards then sends them out to external storage and processing.
- Generic arbitration code (same code is used on main and io FPGAs) for random and fair detector board selection. This is useful in SINGLES mode. Colliding Single Words are randomly selected using a true number generator (using ring oscillators) implemented in FPGA fabric.

DetectorBoard

- An optional TDC core is included.
- Default firmware image is now uncompressed to allow users to generate any firmware logic they desire without any size restrictions.
- Generic ADC interface: (a) Supports multiple TI ADC chips and resolutions, (b) generic deserialization without the use of PLLs.
- Better timing and synchronization due to the use of PLL based clocking and resets. A single PLL clocks all parts the detector board including the ADCs.

- Optional debugging interface to all modules.
- Generic FIFO depths that can be changed on compile time to increase/decrease number of raw samples.
- Advanced Scope mode with multiple parameters like samples before trigger, trigger window, etc.
- Singles mode with generic pipeline interface. Number of pipeline stages is configured at compile time. This allows us to parameterize the processing time for a user-defined singles event.
- Firmware based thresholds for Scope mode.
- Example Singles mode is included to show users how to compute pulse energies or area under the curve.

2.1.2 Embedded Software

- Complete rewrite from scratch.
- Standard 32-bit SPI interface across all FPGAs and boards. A OSI model like architecture is used. e.g. Point to point communication, broadcast communication, time-to-live, number of retries, timeout, etc. The communication protocol follows a request-response architecture.
- Interrupt based handing for SPI commands, QuickUSB commands, Software-Firmware commands.
- Asynchronous, i.e., non-blocking commands are implemented. Parent doesn't wait for child to respond, it will poll the reply later on at its leisure.
- BSPs are auto-generated at compile time. BSP settings are stored in TCL scripts.

SupportBoard

Main FPGA

- Child FPGA images are now considered any compressed. This allows the embedded software to program any firmware image regardless of size.
- Generic QuickUSB interface. Allows parameterization of OpenPET command length, time-to-live, timeout, etc.
- Ethernet compatible code base.
- UART-over-JTAG console for additional verbosity.

IO FPGAs

- Embedded Software is embedded in firmware image (bitstream) at build time. Reduces the number of files to manage.
- Simple design. Extremely small NIOS-II. Code size is 7KB.

DetectorBoard

- Embedded Software is embedded in firmware image (bitstream) at build time. Reduces the number of files to manage.
- Extremely small NIOS-II. Code size is 11KB.
- New peripheral device drivers for ADC, DAC, and SRAM.
- Direct access to all ADC and DAC registers.

2.1.3 Software

- New cross-platform scripting interface. Supports Microsoft Windows, Mac OS X, and GNU/Linux.
- New Python Library *OpenPETlib.py* for easy python scripting.
- New *openpet* executable for Windows x64. Executable is basically *openpet.py* built with *pyinstaller*.
- Example cross-platform scripts for Scope and Singles modes. Example plotting script *plot.py* for quick Scope mode plotting.
- Python library utilizes multiprocessing and queuing to achieve maximum QuickUSB throughput regardless of storage speed.
- Optional realtime user defined data processing.

2.1.4 Hardware

- No changes.

2.2 Upgrading from Previous Releases of OpenPET

Migration guide here.

System Overview

This document describes the OpenPET electronics system. The purpose of the OpenPET electronics is to provide a system that can be used by a large variety of users, primarily people who are developing prototype nuclear medical imaging systems. These electronics must be extremely flexible, as the type of detector, camera geometry, definition of event words, and algorithm for creating the event word given the detector outputs will vary from camera to camera. This implies that users must be able to modify the electronics easily, which further implies that they have easy access to documentation, including the schematics and documents needed to fabricate the circuit boards (Gerber files, bill of materials, etc.) and source code (for both firmware and software). They also need support, in the form of instructions, user manuals, and a knowledge base. And they want fabricated circuit boards to be readily available.

Thus, the OpenPET electronics system includes hardware, firmware, and software. It is scalable enough to provide solutions ranging from a “test bench” for a small number of detector modules to a complete camera. It is also “open source” to both maximize flexibility and minimize redundant development.

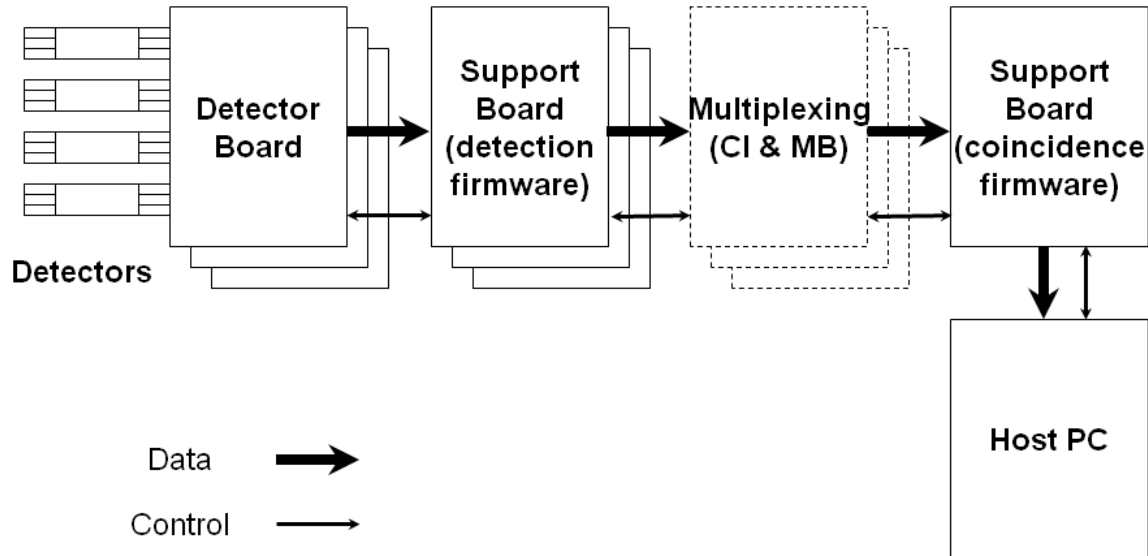


Fig. 3.1: Schematic of the OpenPET system architecture.

The basic system architecture is shown in Fig. 3.1. There are four types of custom electronics boards in the system: the Detector Board (DB), the Support Board (SB), the Coincidence Interface Board (CI), and the Multiplexer Board (MB). The Support Board plays two roles in the system, depending on the firmware.

The general data flow is that analog signals from detector modules provide the inputs to the Detector Board. This board processes the analog signals to create a singles event word, which is a digital representation of this single gamma ray interaction. The singles event words are passed to the Support Board loaded with detection firmware, whose main function here is to multiplex the singles event words from multiple Detector Boards. The singles event words are then passed through the Coincidence Interface Board to the Multiplexer Board, which can provide a further layer of multiplexing for singles event words, if necessary. Next the multiplexed singles event words are passed to another Support Board loaded with coincidence firmware, which searches through the singles event words for pairs that are in time coincidence and then forms coincidence event words. These coincidence event words are then passed to the Host PC. Optionally, the Support Board with coincidence firmware can act as a multiplexer and pass unaltered singles event words to the Host PC.

3.1 Definitions

The OpenPET components are housed in an assembly whose form factor is the same as a 12-slot VME crate that accommodates 6U boards. A Support Board essentially replaces the backplane of the VME crate and all the other boards plug into it. The plug-in boards have the same form factor as a VME 6U board, except that the position of

the connectors is offset (compared to true VME boards) to prevent OpenPET boards from being plugged into standard VME systems and vice versa.

3.1.1 Support Crate

A Support Crate ([Fig. 3.2](#)) is conceptually similar to a VME crate (with controller), namely an intelligent support structure that “functional” boards can be plugged into. It consists of a mechanical frame with 12 plug-in slots, a Support Board (that has a considerable amount of programmable processing power and also acts as a backplane), power supplies, cooling fans, and appropriate boards plugged into slots 9-11. Slots 0-8 are vacant. Slot 9 holds a Host PC Interface Board, which is used to communicate with the Host PC; this board is optional. Slot 10 holds a User IO Board, which allows users to interface to external components such as EKG signals and motor controllers; this board is optional. Slot 11 holds a Debugging Board, which has interfaces to logic analyzers, a number of diagnostic LEDs, an external clock input, and a JTAG connector; this board is optional. Some ancillary components (such as DRAM memory and a QuickUSB board) are also necessary for a functioning Support Crate. By programming the Support Board with appropriate (but different) firmware, the Support Crate becomes part of either a Detector Unit or a Coincidence Unit.

3.1.2 Detector Unit

A Detector Unit (DU), as shown in [Fig. 3.2](#) consists of a Support Crate with between one and eight Detector Boards plugged into slots 0-7. Each Detector Board can process up to 32 analog input signals. A Detector Unit can therefore process up to 256 analog signals, which corresponds to 64 conventional block detector modules (with 4 analog outputs per module). In a Small System ([Fig. 3.4](#)), Slot 8 is empty (if data is transferred to the Host PC through USB or Ethernet via the Host PC Interface Board plugged into Slot 9). In a Standard ([Fig. 3.5](#)) or Large System ([Fig. 3.6](#)), a Coincidence Interface Board must be plugged into Slot 8 of the Detector Unit. There are two versions of the Coincidence Interface Board: Coincidence Interface Board-1 (CI-1) for the Standard System and Coincidence Interface Board-8 (CI-8) for the Large System. At present, the Coincidence Interface Board-8 has not been designed or specified. These boards transfer event data and bidirectional control data between the Detector Unit and a Coincidence Unit.

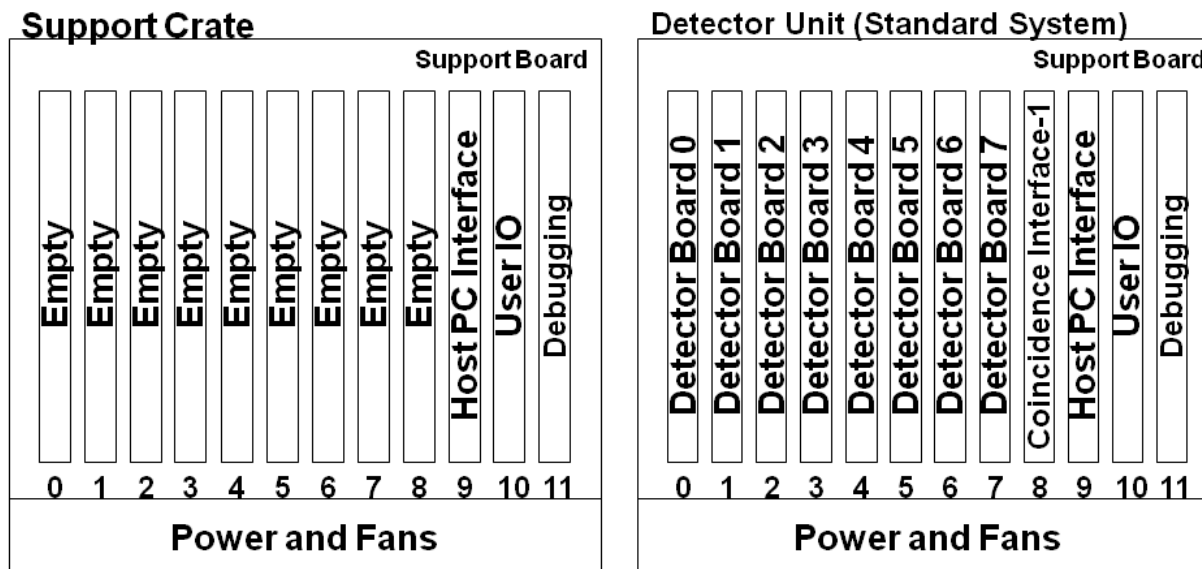


Fig. 3.2: Support Crate (left) and Detector Unit (right). A Detector Unit is a Support Crate with up to 8 Detector Boards (in Slots 0-7). For a Small System, Slot 8 is usually empty, although a Coincidence Interface Board can (optionally) be plugged into it. For Standard and Large Systems, a Coincidence Interface Board must be plugged into Slot 8.

In a Small System (see *Small System* (page 12)), the Support Board in the Detector Unit is programmed to multiplex outputs from the Detector Boards, process coincident events, and pass the coincident events to the Host PC. It can also be programmed to multiplex singles events and pass them to the Host PC. In a Standard or Large System (see *Standard System* (page 13) and *Large System* (page 14)), the Support Board in the Detector Unit is programmed to multiplex singles events from the Detector Boards and forward them to a Coincidence Unit.

3.1.3 Coincidence Unit

In Small Systems, the coincidence processing is performed on the Detector Unit's Support Board. In Standard and Large Systems, the coincidence processing is performed in a Coincidence Unit (CU) as shown in Fig. 3.3.

The Coincidence Unit for a Standard System consists of a Support Crate with between one and eight Multiplexer Boards plugged into slots 0-7. The Support Board is loaded with firmware to perform the coincidence processing. Each Multiplexer Board communicates with one Detector Unit via the Coincidence Interface Board using a cable. Similar to the Coincidence Interface Board, there are two versions of the Multiplexer Board: Multiplexer Board-1 (MB-1) for the Standard System and Multiplexer Board-8 (MB-8) for the Large System. At present, the Multiplexer Board-8 has not been designed or specified. The Coincidence Unit's Support Board is programmed to do the coincidence processing and pass the coincident events to the Host PC, although it can also function as a multiplexer and forward singles events.

Data is transferred to the Host PC either through USB or Ethernet via the Host PC Interface Board plugged into Slot 9.

In a Coincidence Unit, the Support Board that acts as a backplane for the Support Crate is loaded with coincidence firmware. In this case, the Support Board with related firmware and software is called a **Coincidence Unit Controller (CUC)**.

In the Coincidence Unit for the Standard System, each MB-1 connects with only one Detector Unit via a single cable, allowing up to 64 Detector Boards (or 512 block detector modules) in the system. In the Coincidence Unit for a Large System, the MB-8s plugged into slots 0-7 connect via cables (one cable per Detector Unit) with up to 8 Detector Units, allowing up to 512 Detector Boards (or 4096 block detector modules) in the system. The MB-8s are programmed to serve as multiplexers for events coming from up to 8 Detector Units. Due to the nature of multiplexing, this allows a larger number of channels to be serviced, but does not increase the maximum total event rate (singles or coincidence).

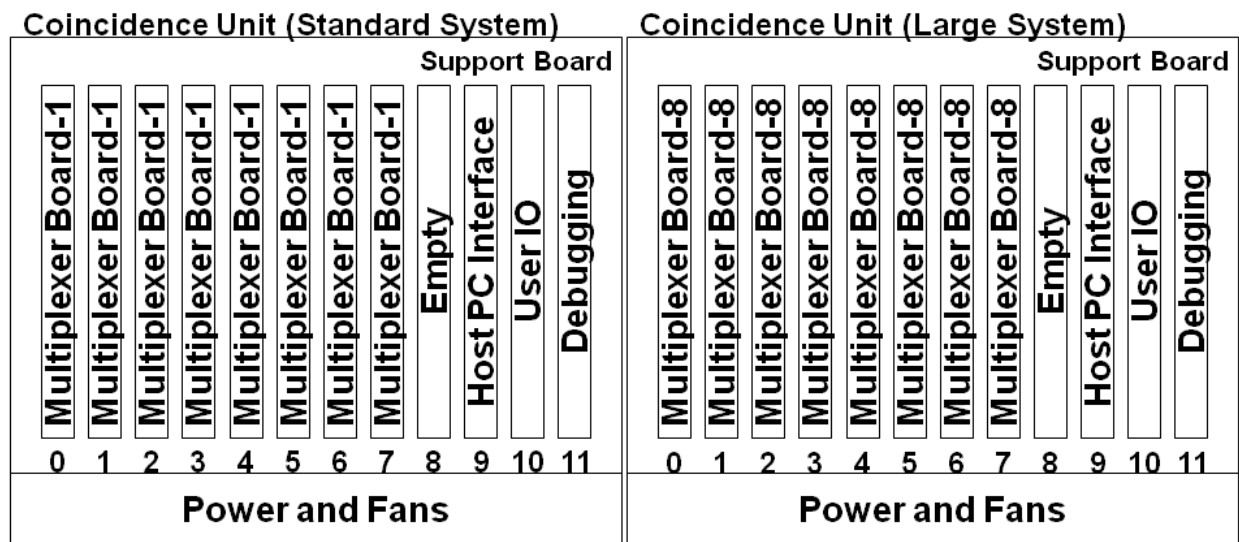


Fig. 3.3: Coincidence Unit for a Standard System (left) and a Large System (right). For both Standard and Large Systems, a Coincidence Unit is a Support Crate with up to 8 Multiplexer Boards (in slots 0-7). In a Large System, the Multiplexer Boards function as multiplexers.

3.2 System Configuration

OpenPET can be configured either as a Small System, Standard System, or Large System, with the difference primarily due to the number of analog signals that can be read out. To determine which system components you need, the first step is to determine how many DBs are necessary. Each DB can process up to 16 or 32 analog signals (depending

on the DB used, mixing of different types of DB is not supported), so the minimum number of DBs necessary is the number of analog signals divided by 16 or 32. While information can be shared between DBs, processing is far easier if all the signals from the same detector module are on the same DB. Thus, if your detector module has 5 analog outputs and you use a 32-channel DB, it is easiest to have each DB process 30 analog signals (i.e., from six detector modules) and not use the other two channels on each Detector Board. Thus, the initial estimate for the number of DBs needed is the number of analog signals divided by the number of analog signals you will have each DB process. This estimate may be modified due to the camera topology, as described in the following subsections. Once you have determined this initial estimate for the number of Detector Boards, you can determine whether you will need a Small, Standard, or Large OpenPET System.

3.2.1 Small System

If the total number of Detector Boards is 8 or fewer, you can use a Small System (Fig. 3.4). This consists of a single Detector Unit (defined in *Detector Unit* (page 9)) connected to a Host PC. A Detector Unit (DU) can have anywhere between 1 and 8 DBs plugged into it. The initial estimate of the number of DBs in your system may need to be increased to support the camera topology. In the Small System, the default coincidence processing algorithm searches for coincidences between singles events that originate on different DBs, but it doesn't allow coincidences between singles events that originate on the same DB. Thus, more DBs may be necessary to allow all the desired coincidences, as none of the modules in a DB can be in coincidence with each other.

As an example, consider a four-headed PET system, where each head consists of a 3x3 array of detector modules and each detector module has five analog outputs. The OpenPET system would be configured as follows. Each 32-channel DB would service 6 detector modules, using 30 analog channels and leaving two channels on each DB unused. As the system consists of 36 detector modules (four heads of nine modules each), the initial estimate for the number of DBs is six. To see whether the appropriate coincidences can be accommodated, we first try to distribute the detector modules as follows. DB 0 services six modules from head 0, DB 1 services three modules from head 0 and three from head 1, DB 2 services six modules from head 1, DB 3 services six modules from head 2, DB 4 services three modules from head 2 and three from head 3, and DB 5 services six modules from head 3. Unfortunately, this will not work, as two Detector Boards (numbers 1 and 4) service modules from two different heads, which means that the system will not look for all possible coincidences between detector modules that are in different heads. No amount of redistributing the modules among the DBs will satisfy these criteria either. Thus, the only way the coincidence criteria can be satisfied (using the default coincidence processing software) is to use two DBs per head, for a total of eight Detector Boards. While the coincidence processing software can be rewritten to allow coincidences between two singles events that originate in the same DB, this is likely to take significantly longer and cost more than purchasing two additional Detector Boards.

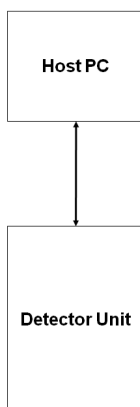


Fig. 3.4: Configuration of a Small OpenPET System.

3.2.2 Standard System

If the total number of DBs is between 9 and 64, you can use a Standard System (Fig. 3.5). This consists of between 2 and 8 DUs (defined in *Detector Unit* (page 9)) and a single CU (defined in *Coincidence Unit* (page 10)) connected to a Host PC. The DUs are connected to the CU via the Coincidence Interface Board CI-1 in each of the DUs and the Multiplexer Board MB-1 in the CU. Determining the number of DUs and DBs per DU needed follows the same principles as described in the Small System section. Each DU should only contain modules that will not be in coincidence with each other, as the default coincidence processing software does not allow coincidences between detectors that originate from the same DU. The DU should then contain the minimum number of DBs necessary to service all the required detector modules.

Each DU services a maximum of eight DBs, and each 32-channel DB services a maximum of 32 analog inputs (note that a conventional PET block detector has four analog outputs, one for each photomultiplier tube). Thus, the Standard System can support a maximum of 2048 analog inputs (32 analog channels per DB, 8 DBs per DU, and 8 DUs per CU), which corresponds to 512 block detector modules.

As an example, consider a cylindrical PET camera, where each detector module has five analog outputs and covers a 5 cm x 5 cm area. There are 44 detector modules per ring (roughly 70 cm diameter) and 5 rings (25 cm axial coverage). The OpenPET system would be configured as follows. The system consists of 220 detector modules. Each 32-channel DB would service 6 detector modules, using 30 analog channels and the remaining two channels on each DB would be unused. As there are a maximum of eight DBs per DU, a DU can service a maximum of 48 of these detector modules. If we divide the 220 modules by 48 modules per DU, we find that the system requires 4.583 DUs. Since DUs are quantized, it really needs 5 DUs, with each DU servicing 44 detector modules. In order to make sure that the correct coincident pairs will be collected, the modules in each DU should be selected so that each DU services a “pie slice” that spans $\sim 72^\circ$ azimuthally and the full 25 cm axial thickness.

OpenPET Standard System

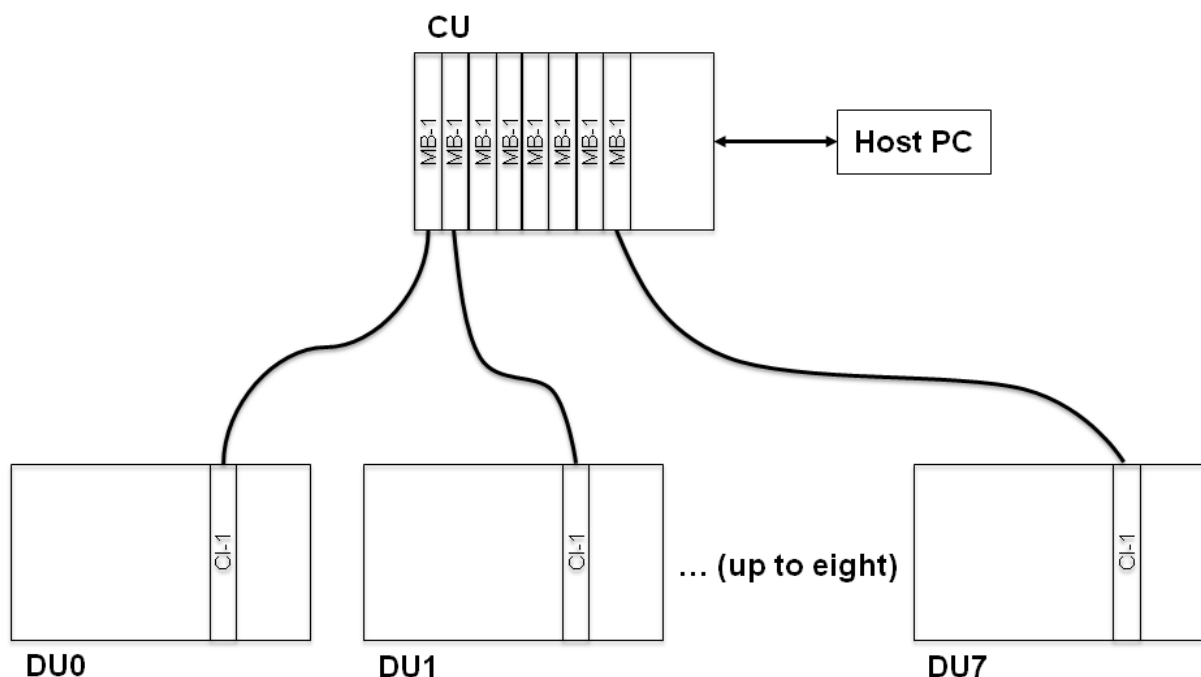


Fig. 3.5: Configuration of a Standard OpenPET System. Each of slots 0-7 in the Coincidence Unit contains a Multiplexer Board-1 that services a single DU. Slot 8 in the Detector Unit contains a Coincidence Interface Board-1 that transfers event data and bidirectional control between the DU and CU.

3.2.3 Large System

If the total number of DBs is between 65 and 512, you must use a Large System (Fig. 3.6). This consists of between 9 and 64 DUs (defined in *Detector Unit* (page 9)) and a single CU (defined in *Coincidence Unit* (page 10)) connected to a Host PC. The DUs are connected to the CU via the CI-8 in each of the DUs and the MB-8 in the CU. Determining the number of DUs and DBs per DU needed follows the same principles as described in the Standard System section. The difference between the Standard and Large Systems is that the Multiplexer Board-8 that plugs into slots 0-7 of the CU contains active circuitries (i.e., FPGA, etc.) that are configured as multiplexers. This allows each of the eight slots in the CU to service up to eight DUs (in a Standard System, each CU slot services a single DU). Thus, the Large System can support a maximum of 16,384 analog inputs (32 analog channels per DB, 8 DBs per DU, and 64 DUs per CU), which corresponds to 4,096 block detector modules. Again, the group of DUs processed by one slot in the CU should only contain modules that will not be in coincidence with each other, as the default coincidence processing software does not allow coincidences between detectors serviced by the same CU slot. The DU should then contain the minimum number of DBs necessary to service all the required detector modules.

OpenPET Large System

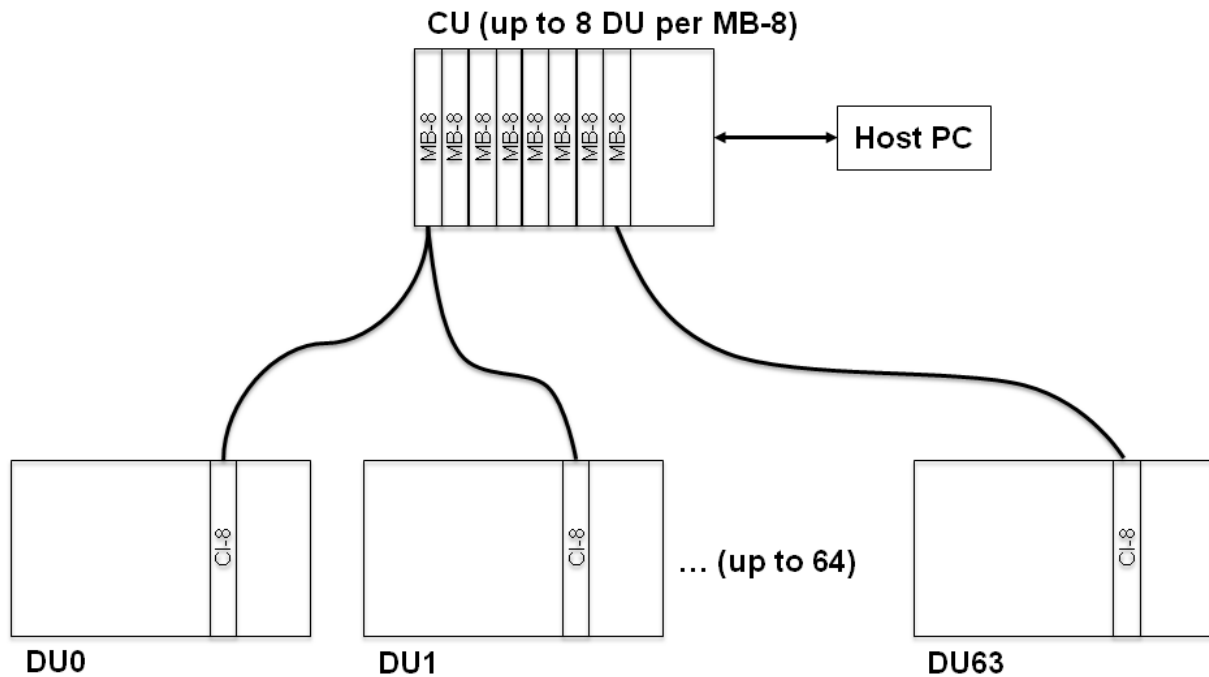


Fig. 3.6: Configuration of a Large OpenPET System. Each of slots 0-7 in the CU contains a Multiplexer Board-8 that operates as a multiplexer, allowing that slot to service between 1 and 8 DUs. Slot 8 in the Detector Unit contains a Coincidence Interface Board-8 that transfers event data and bidirectional control between the DU and CU.

3.3 Timing & Timing Signals

The system level timing signals are shown in Fig. 3.7. There are two timing signals—the system clock, which is an 80 MHz clock signal, and the time slice boundary, which defines the beginning of a time slice. The firmware will support both “short” and “long” event words. In “short” mode, the time slice boundary is generated every eight system clock cycles, while in “long” mode it is generated every sixteen system clock cycles. The choice creates a tradeoff—in “short” mode the dead time is a factor of two shorter, but the number of bits per event word is also a factor of two smaller.

The general concept is that the system divides time into small, fixed length time slices (100-200 ns or 8-16 clocks). All individual operations must occur within a single time slice, which implies that only singles event words that occur in the same time slice can be combined to form a coincident event. Since it can take significantly longer than a single time slice to fully process a single event, the system is pipelined so that the processing is divided into smaller steps that each can be completed in a single time slice.

During one time slice, each of the boards in a Standard System that output singles event words (i.e., DB, CI-1, and MB-1) can pass four singles event words. Thus, the maximum singles rate seen at the CI-1 output of each Detector Unit is 4 singles event words per time slice, or approximately 40 million “short” singles event words per second. Thus, 32 singles event words (four for each of the eight CI-1) enter the Coincidence Unit per time slice, or

approximately 320 million “short” singles event words per second. In a Standard System, there are 8 Detector Units with 28 possible Detector Unit - Detector Unit combinations. So theoretically the Coincidence Unit can identify 448 coincident events per time slice (16 for each of the 28 Detector Unit-Detector Unit combinations), corresponding to 4.48 billion coincidence event words per second. In practice, the maximum event rate is limited by the transfer rate between the Coincidence Unit and the Host PC, which is considerably slower.

3.3.1 System Clock

The system clock is an 80 MHz clock. In general, it is generated on the Support Board in the Coincidence Unit (although it can be generated on the Support Board in a Detector Unit such as in the Small System), and then buffered through the rest of the system. Propagation delays will introduce skewing, therefore each FPGA that outputs data will also output a copy of the system clock that is synchronized with its output data signals. In general, each board in the system regenerates the clock using a phase-locked loop (PLL) in order to maintain signal quality and to minimize phase drift.

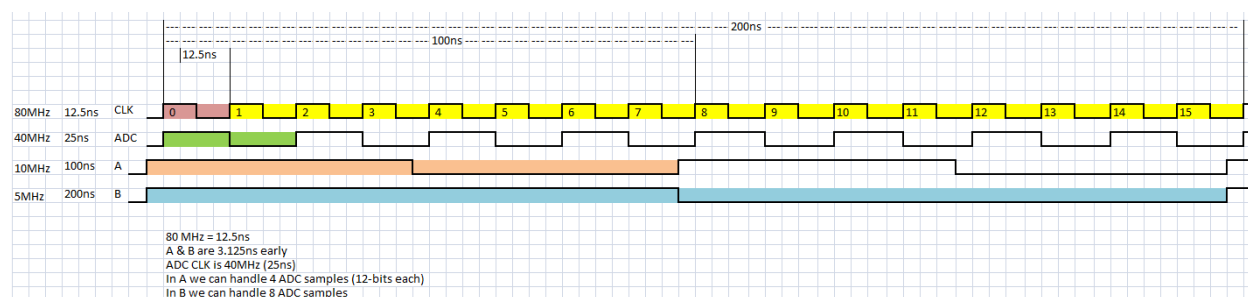


Fig. 3.7: System level timing signals.

3.3.2 Time Slice Boundary

The rising edge of the time slice boundary defines the beginning of a time slice. The width of the pulse is one system clock cycle, and the period is eight system clock cycles (for “short” event word mode) or sixteen system clock cycles (for “long” event word mode). In general, it is generated on the Support Board in the Coincidence Unit (although it can be generated on the Support Board in a Detector Unit such as in the Small System), and then buffered through the rest of the system. Propagation delays will introduce skewing; therefore each FPGA that outputs data will also output a copy of the time slice boundary that is synchronized with its output data signals.

3.3.3 Time Slice

The system divides time into small, fixed length time slices (100-200 ns or 8-16 clocks). All individual data processing operations must occur within a single time slice, which implies that only singles event words that occur in the same time slice can be combined to form a coincident event. While it can take significantly longer than one time slice to fully process a single event, the system is pipelined so that the processing is divided into smaller operations that each can be completed in a single time slice. It takes one time slice to transfer a singles event word.

3.4 Firmware & Software Structures

{ Write this after you’ve written the corresponding section for the Developers Guide? }

The OpenPET firmware and software structures are based on a computer network tree topology. The configuration strategy needs to fulfill the following two basic requirements:

- (1) Compatibility with different types of detector modules (e.g. single analog channel addressing, single crystal addressing for a conventional block detector, etc.);
- (2) Compatibility with different sized systems (e.g., Small, Standard and Large Systems).

In addition, the addressing strategy needs to be implementable, flexible and reliable.

3.4.1 Standard System

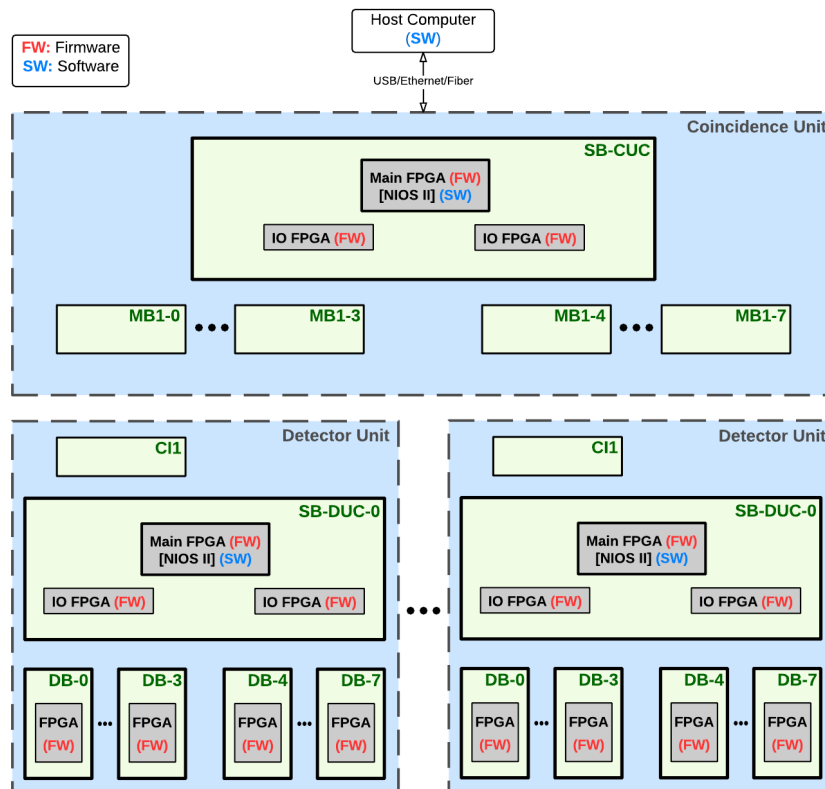


Fig. 3.8: Standard system firmware and software structure. (Add Host PC Interface Board, like Fig. 6 Developer Guide.)

3.4.2 Large System

3.4.3 Small System

As described earlier (*Small System* (page 12)), the OpenPET system can also be configured as a Small System. In a Small System, a support board is configured as a Coincidence Detector Unit Controller (CDUC), which interfaces with the detector boards and performs coincidence functions. Basically the CDUC performs the functions of both the CUC and DUC. The initial firmware and software for the first release has been developed for a Small System. The configuration for a Small System is shown in Figure ?.

Getting Started - *Small System Oscilloscope Mode*

4.1 Getting Hardware

The OpenPET system uses a custom Support Crate that is similar to a VME crate (see *Support Crate* (page 9)). Support Crates should be purchased through Elma: Support Crate Chassis, part number 12V12XXX78N2VCGX-LBL, <http://www.elma.com/en/us/>.

The OpenPET PC boards can be purchased from Terasic through their OpenPET website at <http://www.openpet.terasic.com>. For a small system in the first release, each Detector Unit (see *Detector Unit* (page 9)) requires the following OpenPET boards:

- 1 Support Board
- 1-8 16-Channel Detector Boards

You also need the following additional components:

- 1 Host PC with a Windows 7 operating system.
- 1 QuickUSB module: Bitwise Systems, part number QUSB2, <http://www.bitwisesys.com/qusb2-p/qusb2.htm>.
- 1 standard USB cable
- 1 USB-Blaster Cable: Terasic, Digi-Key part number P0302-ND, <http://www.digikey.com/product-highlights/us/en/terasic-usb-blaster-cable/3718>.

The QuickUSB module is a small PC board that contains circuitry to provide high-speed USB 2.0 capability. It is plugged into either the Host PC Interface Board or the Support Board. The USB-Blaster cable interfaces between a USB port on the Host PC to the Altera main FPGA on the Support Board, so configuration data can be sent from the PC to the FPGAs. More detailed instructions are provided in the following sections.

4.2 Assembling the Hardware

Warning: All OpenPET parts are Electrostatic Sensitive Devices. Connect ground wire before touching any ESD.

Once you receive the parts for an OpenPET support crate, some minor assembly is required. Fig. 4.1 shows the two main components: the empty Support Crate and the Support Board. In addition, you will receive a small custom jumper board (see Fig. 4.2c). You will also need a standard power cable for the crate, M2.5 x 12 mm Phillips head screws, M4 x 5 mm Phillips head screws, and appropriate screw drivers.

The first assembly step is to attach the Support Board to the crate. From the back side of the crate, align the screw holes on the Support Board to those on the Support Crate and secure it using M2.5 x 12 mm Phillips head screws. We recommend using at least 3 screws on the top, middle and bottom rows. It is also useful to place a piece of paper across the fans during assembly so any dropped screws don't fall into them. When the board is properly aligned, there should be about a 4 mm gap between the right edge of the Support Board and the right side of the Support Crate (Fig. 4.2a).

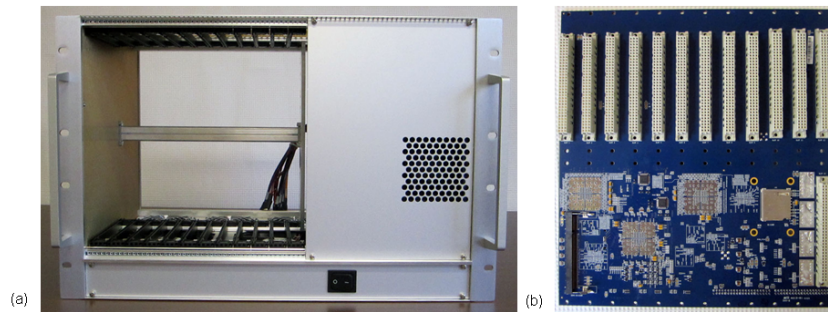


Fig. 4.1: (a) Empty Support Crate viewed from the front. (b) Support Board viewed from the front.

Once the Support Board is secured on the crate, you need to attach the power cables. The power cables and Support Board connectors are labeled. From bottom to top, the power cables are -5 V (orange cable), ground (black cables), +3.3 V (purple cables) and +5 V (red cable), as shown in Fig. 4.2b. Secure these power cables on to their respective Support Board connectors using M4 x 5 mm Phillips head screws. You may need to feed each screw through the power cable lug nut before attaching it to the Support Board, since the lug nuts fit tightly. In addition, there is a small bundle of cables that can be used for monitoring but these cables are not needed.

Finally you need to plug the custom small jumper board into the back of the Support Board. It plugs in to the connector just to the right of the Jtag connector (see Fig. 4.2c). Specifically, board plugs into the left column of pins on the connector (which is marked main, 58, 62, 65, 68). This jumper board identifies that the Jtag should communicate with the main FPGA on the Support Board.

The fully assembled Support Crate is shown in Fig. 4.3.

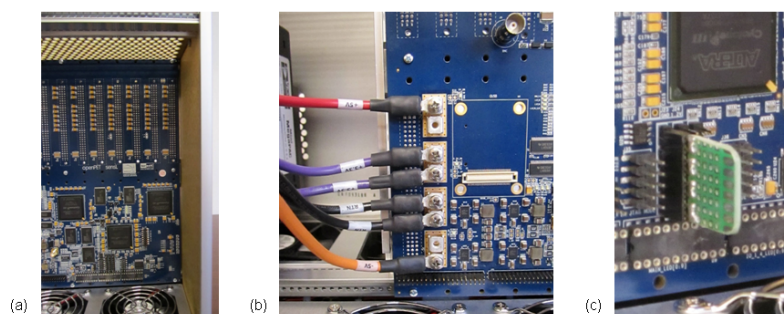


Fig. 4.2: (a) Close up of the small gap (~4 mm) between the Support Crate and the edge of the Support Board when assembled properly. (b) Close up of the power cables attached to the back of the Support Board. (c) Close up of the jumper board plugged into the back of the Support Board.



Fig. 4.3: The assembled OpenPET Support Crate viewed from the (a) front and (b) back.

4.3 Downloading the Software and Firmware

4.3.1 Installing QuickUSB

The OpenPET system requires the use of a QuickUSB module to provide high-speed USB 2.0 capability. You can purchase this online through Bitwise systems at <http://www.bitwisesys.com/qusb2-p/qusb2.htm> (product code QUSB2). In addition to the hardware module, this includes the QuickUSB library with a driver, interface DLL, and example programs for Linux, MacOSX and Windows. In addition, you will need a USB cable. However, you do not need to buy the QuickUSB Adapter Board as Bitwise implies.

First, you need to mount the QuickUSB module onto either the Host PC Interface Board in the U3 connector or onto the Support Board in the U6 connector (Fig. 4.4). The system is designed to work with the QuickUSB module mounted in either (but not both) of these locations.

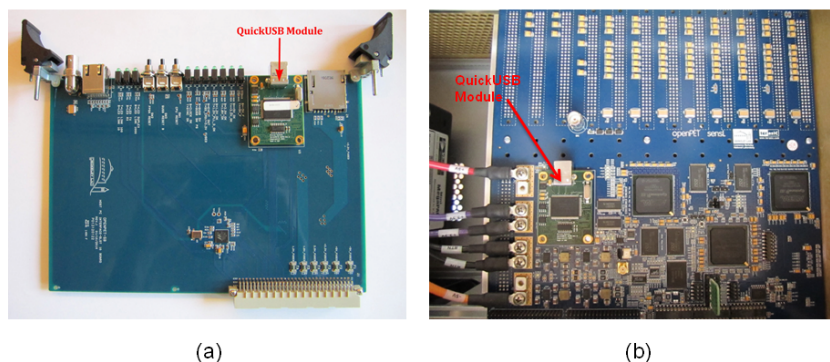


Fig. 4.4: QuickUSB module mounted onto the (a) Host PC Interface Board or (b) Support Board.

Next, you need to install the QuickUSB driver onto your Host PC. Browse and select the setup executable for the QuickUSB software (e.g., Desktop -> QuickUSB_Installation -> Windows -> setup.exe). This will launch the InstallShield Wizard in a new window (Fig. 4.5a). Step through the instructions by clicking the Next button at the bottom, agreeing to the software license agreement terms and defining the folder where the QuickUSB software should be installed (Fig. 4.5b). When the wizard is ready to begin installation of the library, click Install (Fig. 4.5c). The installation takes a few minutes and a status bar shows the progress. An additional window will then pop up that walks you through the installation of the QuickUSB device drivers (Fig. 4.5d). The InstallShield Wizard will indicate when the full installation is completed.

If you have further questions or problems with the installation, please refer to the Bitwise QuickUSB User Guide for details (e.g., http://www.bitwisesys.com/v/public/media/QuickUSB_User_Guide_v2.15.2.pdf).

Once you have installed the QuickUSB module, you need to confirm it has the correct firmware model. OpenPET uses the default firmware model i.e., “QuickUSB QUSB2 Module v2.15.2 (Simple I/O)”. If the version or the model is not correct, update the firmware using the QuickUSB Programmer (see Fig. 4.6). After you have successfully installed the correct firmware, you can confirm your installation by running the QuickUsb Diagnostics (see Fig. 4.7).

The OpenPET system requires the firmware model “QuickUSB QUSB2 Module v2.15.2 (Simple I/O)” to configure the QuickUSB module for the correct data transfer mode. For further details on this firmware model and configuration, please refer to the QuickUSB User Guide at http://www.bitwisesys.com/v/public/media/quickusb_user_guide.pdf.

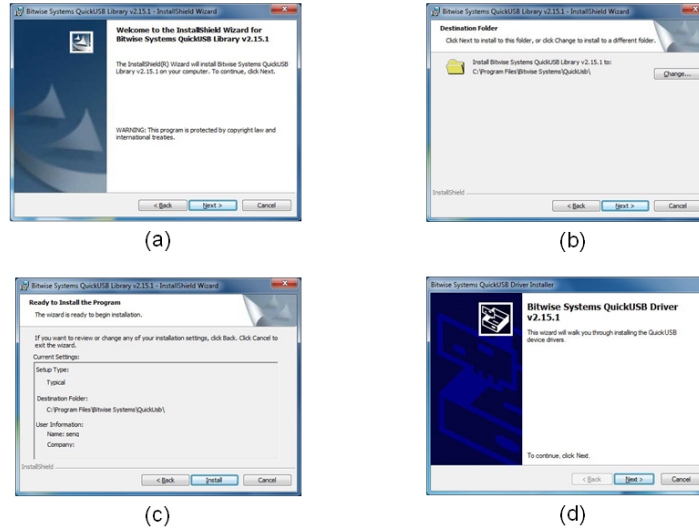


Fig. 4.5: Bitwise Systems QuickUSB Library v2.15.2 InstallShield Wizard: a) welcome window, (b) destination folder window, (c) installing the library window, and (d) installing the driver window.

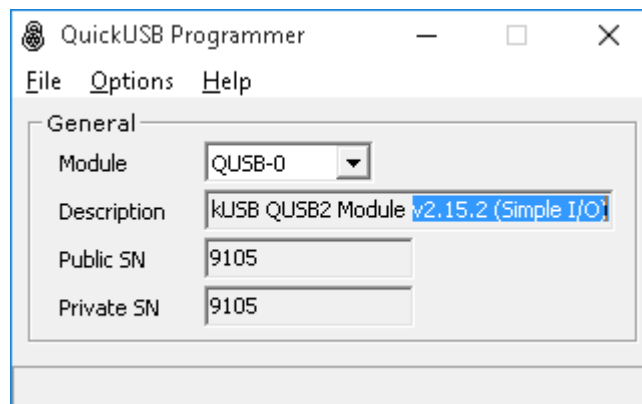


Fig. 4.6: QuickUSB Programmer used to program the QuickUSB module with the correct firmware: QuickUSB QUSB2 Module v2.15.2 (Simple I/O).

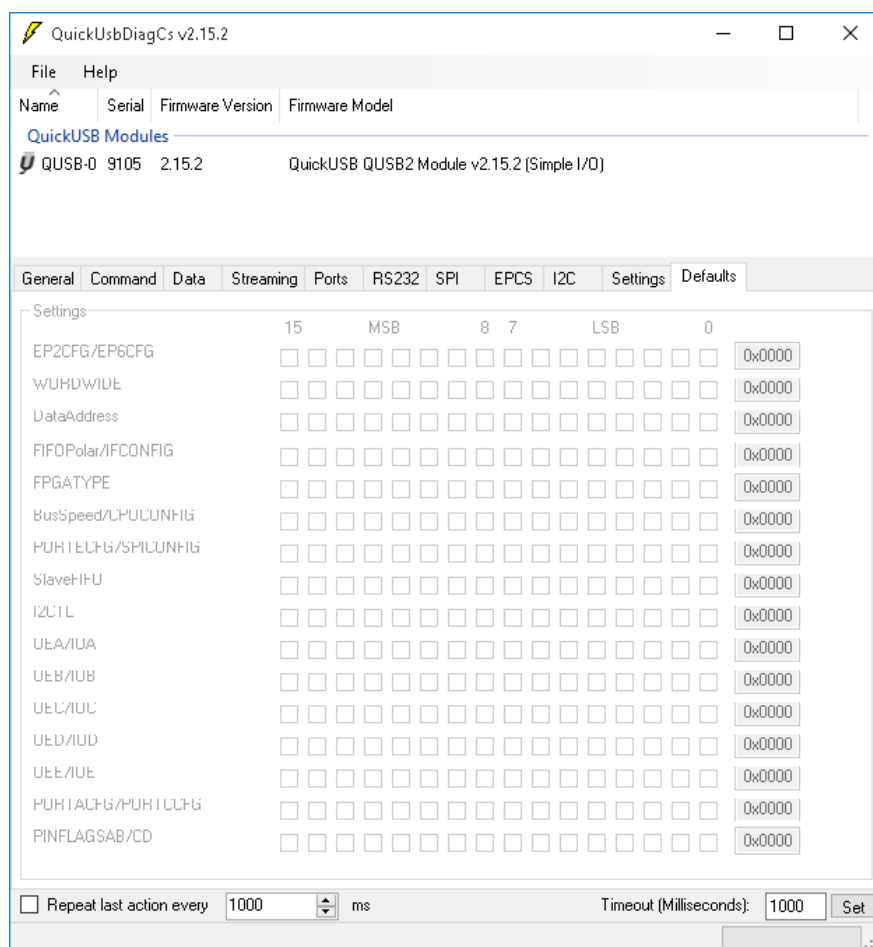


Fig. 4.7: QuickUsb Diagnostics used to confirm the QuickUSB module with the correct firmware: QuickUSB QUSB2 Module v2.15.2 (Simple I/O).

4.3.2 Installing Altera Tools

Even if you only plan to use the off the shelf OpenPET software and firmware, the OpenPET system requires the use of Altera design tools in order to load the appropriate firmware into the Support Board. These tools can be downloaded for free from the Altera Download Center website (e.g., <https://www.altera.com/download/sw/dnl-sw-index.jsp>, Fig. 4.8 a). From the Download Center, select to download the Quartus II Web Edition that is free and doesn't require a license.

On the Quartus II Web Edition download page Fig. 4.8 b, select release: 13.1 select the Windows operating system and the Akamai DLM Download Manager. Under the Individual Files tab, select the Quartus II Software and the Cyclone III, Cyclone IV device support. You can select additional options (such as the ModelSim-Altera Edition simulation tools), but they are not required to run OpenPET and they will lengthen the download time.

At this time, only Altera Quartus **13.1** is supported.

You are then required to log in with your username and password if you already have a myAltera account. If not, then create an account using your email address and complete the account registration information. Once your myAltera account has been created, you will be directed to your myAltera Home page. Select the Download Center link on the top right side of the page.

Once you have returned to the Download Center (Fig. 4.8 a), select the Quartus II Web Edition and specify the Windows operating system, DLM Download Manager, Quartus II Software, and the Cyclone III, Cyclone IV device support (if they aren't already specified). A separate Akamai NetSession Interface window will then pop up (Fig. 4.8 c) in which you should click Download on the installer and then Run to proceed with the software installation. You will have to agree to the End User License terms. A popup warning message will then appear in which you have to confirm that you want to open the executable file (e.g., QuartusSetupWeb-13.10.163.exe) from the Internet. You will then have to confirm that you want to allow this program to make changes to your computer (Fig. 4.8 d).

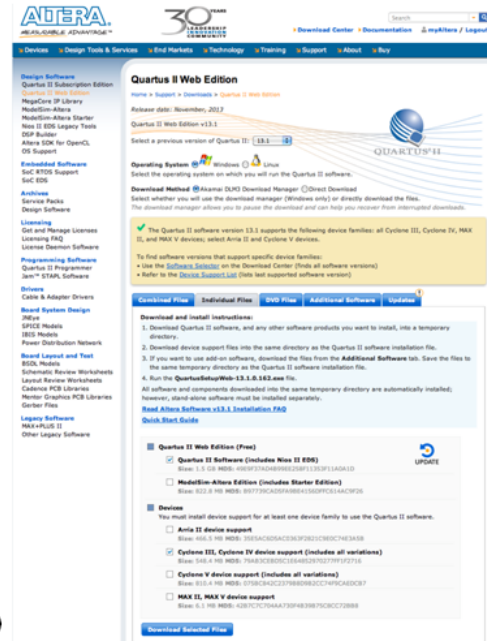
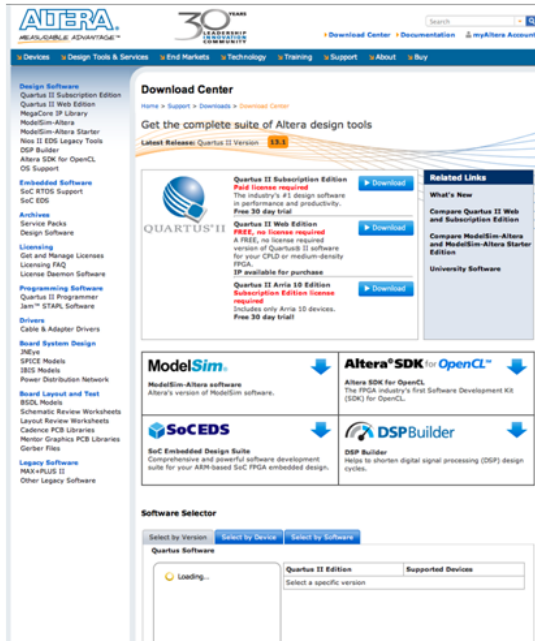
Finally the Quartus II Web Edition Setup Wizard window will appear and step you through the installation (Fig. 4.8 e). You will have to agree to a License Agreement again as well as specify the installation directory (e.g., C:\altera13.1). Then you should make sure that the components selected for installation are correct (e.g., Quartus II Software and Cyclone III/IV) and that you have the available disk space specified in the Summary. Once you start the installation, a status bar will show its progress. Once the Setup has finished installing the Altera tools, you can create shortcuts on your Desktop and launch Quartus II (Fig. 4.8 f).

If you have further questions or problems with the installation, please refer to the Altera support website (<http://www.altera.com/support/spt-index.html>) for user information. For specific instructions on how to use these Altera tools with the OpenPET system, see *Installing OpenPET Firmware & Software* (page 27).

4.3.3 Installing USB-Blaster Driver

The USB-Blaster cable interfaces between a USB port on your Host PC and the Altera main FPGA on the Support Board. It allows configuration data to be sent from the Host PC to the FPGAs.

You must install the Altera USB-Blaster or USB-Blaster II driver before you can use the program devices with the Quartus II software. These drivers are automatically copied to the drivers folder within the Altera folder during the installation described in the previous section. However, it needs to be installed at first use. You will be prompted to install the driver the first time the USB-Blaster cable is plugged in. Whether you need to install the USB-Blaster or USB-Blaster II driver depends on your cable. Please see the step-by-step Altera instructions for this driver installation at <http://www.altera.com/download/drivers/usb-blaster/dri-usb-blaster-vista.html>.



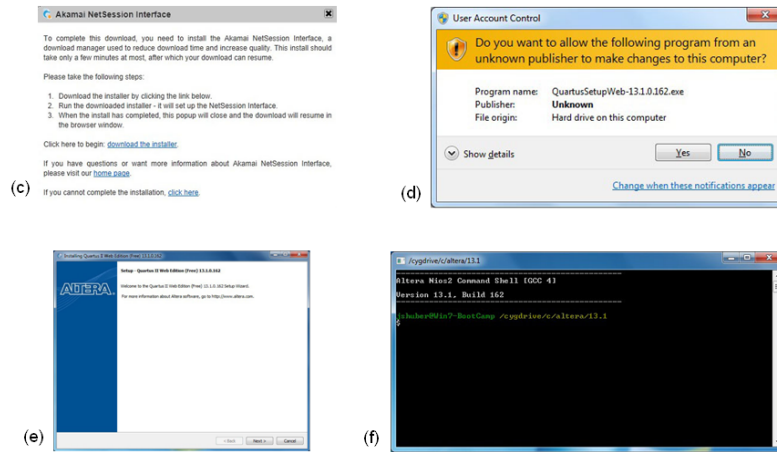


Fig. 4.8: Altera Tools installation windows: (a) download center window, (b) Quartus II Web Edition window, (c) Akamai NetSession Interface window, (d) User Account Control confirmation window, (e) Quartus II Web Edition Setup Wizard window and (f) Altera Nios2 Command Shell window.

4.3.4 Installing OpenPET Firmware & Software

Download onto your Host PC the latest version of the OpenPET firmware and software “Binary” zip from the OpenPET website at <http://openpet.lbl.gov/downloads/firmware-software/>. You will need to register as an OpenPET user and login before you can access this page. Once you unzip this file (e.g., OpenPET_v2.0.zip), you should see the **OpenPET_ROOTDIR** directory, as shown in Fig. 4.9 .

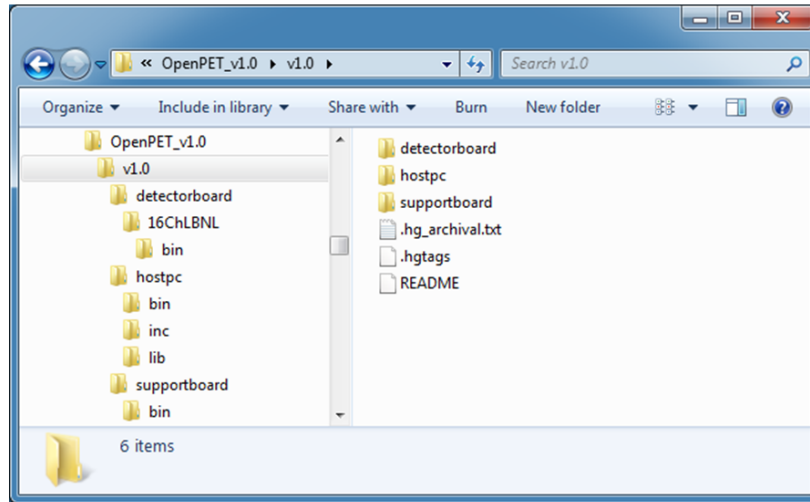


Fig. 4.9: OpenPET Directory Tree (**OpenPET_ROOTDIR**)

The directory '**OpenPET_ROOTDIR**/supportboard' contains the Support Board's firmware, as well as an additional script to program the board's flash.

The directory '**OpenPET_ROOTDIR**/detectorboard' contains the LBNL 16-channel Detector Board's firmware and a script to program the flash. Future releases will also include other Detector Boards.

The directory '**OpenPET_ROOTDIR**/hostpc/dist/openpet' contains the `openpet.exe`. This executable is a Microsoft Windows x64 executable for configuring the system and acquiring data. It is recommended that you add this directory to your MS Windows PATH environment variable. (You will have to do this every time a new OpenPET binary package is released.) Open a Windows Control Panel and type 'env' in the search box in the upper right corner as shown in Fig. 4.10 , then click on 'Edit environment variables for your account'. When the new popup window appears, go to user variables, double click on the variable 'PATH', and append the full path to variable value (see Fig. 4.11).

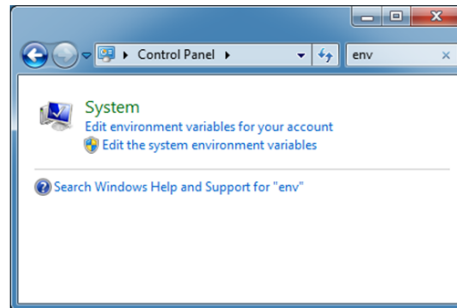


Fig. 4.10: Windows Environment Variable through control panel

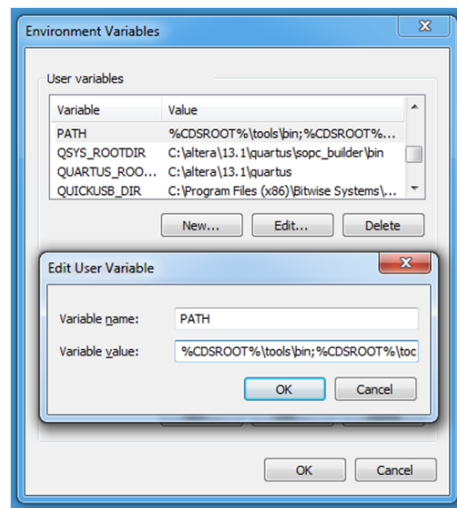


Fig. 4.11: Setting the PATH environment variable

4.3.5 Programming OpenPET Flash Images

Before we program the flash images on the Support Board, we have to setup some jumpers. First make sure the Support Crate is powered off and disconnected. Place jumpers on each Detector Board, connecting pins 1 and 2 on J1,

J2 and J3, so you can download firmware to the Detector Board (Fig. 4.12 (a)) using the instructions below. (Note: code can also be downloaded via the JTAG connector, which is still enabled with these jumpers in place.)

Connect the USB-Blaster hardware from the back of the Support Board to your Host PC (USB port), as shown in Fig. 4.12 (b); pin 1 (marked as red) should face down when connecting to the JTAG connector.

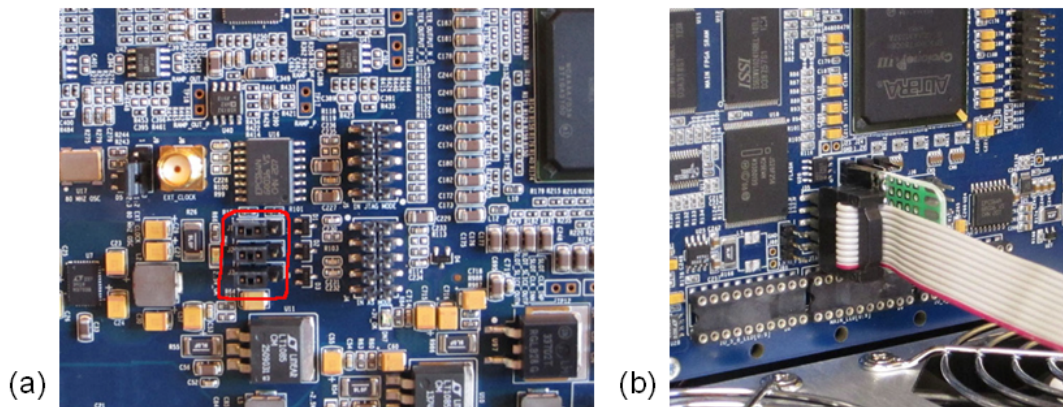


Fig. 4.12: (a) Detector Board with jumpers installed on pins 1 and 2 for J1, J2 and J3. (b) USB-Blaster plugged into the JTAG connector on the back of the Support Board. A custom jumper board is also shown.

After plugging in the cable, connect and turn the power on to your Support Crate. Now you are ready to download the OpenPET software and firmware to your OpenPET hardware in two steps:

- In the first step, you download the OpenPET software and firmware to the 3 FPGAs on the Support Board. Go to `OpenPET_ROOTDIR/supportboard` and launch `flashboard.bat` (Type = Windows Batch file). Please follow the prompts on the command line window. The result is shown in Fig. 4.13. Once this is done, it is important that you reboot the crate by turning it off then on again.

```

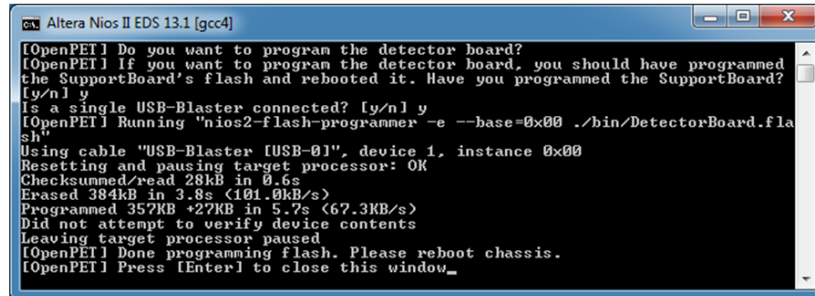
Altera Nios II EDS 13.1 [gcc4]
[OpenPET] Do you want to program the support board? Make sure USB-Blaster is connected. [y/n] y
[OpenPET] Running "quartus_pgm -c usb-blaster -n jtag -o ipv:/bin/supportboard.jic"
Info: *****
Info: Running Quartus II 32-bit Programmer
Info: Version 13.1.4 Build 182 03/12/2014 SJ Full Version
Info: Copyright (C) 1991-2014 Altera Corporation. All rights reserved.
Info: Your use of Altera Corporation's design tools, logic functions and other software and tools, and its AMPP partner logic functions, and any output files from any of the foregoing (including device programming or simulation files), and any associated documentation or information are expressly subject to the terms and conditions of the Altera Program License Subscription Agreement, Altera MegaCore Function License Agreement, or other applicable license agreement, including, without limitation, that your use is for the sole purpose of programming logic devices manufactured by Altera and sold by Altera or its authorized distributors. Please refer to the applicable agreement for further details.
Info: Processing started: Mon Sep 08 10:37:27 2014
Info: Command: quartus_pgm -c usb-blaster -n jtag -o ipv:/bin/supportboard.jic
Info: (213045): Using programming cable "USB-Blaster (USB-01)"
Info: (213041): Using programming file ./bin/supportboard.jic with checksum 0x496262B1 for device EP3C4001
Info: (209060): Started Programmer operation at Mon Sep 08 10:37:29 2014
Info: (209016): Configuring device index 1
Info: (209017): Device 1 contains JTAG ID code 0x020F40DD
Info: (209007): Configuration succeeded -- 1 device(s) configured
Info: (209018): Device 1 silicon ID is 0x16
Info: (209044): Erasing ASP configuration device(s)
Info: (209023): Programming device(s)
Info: (209021): Performing CRC verification on device(s)
Info: (209011): Successfully performed operation(s)
Info: (209061): Ended Programmer operation at Mon Sep 08 10:38:51 2014
Info: Quartus II 32-bit Programmer was successful. 0 errors, 0 warnings
Info: Peak virtual memory: 190 megabytes
Info: Processing ended: Mon Sep 08 10:38:51 2014
Info: Elapsed time: 00:01:24
Info: Total CPU time (on all processors): 00:00:03
[OpenPET] Done programming board. Please reboot chassis.
[OpenPET] Press [Enter] to close this window

```

Fig. 4.13: Support Board flash programming script

- In the second step, you download the OpenPET firmware to the Detector Board FPGA. Go to OpenPET_ROOTDIR/16ChLBNL/detectorboard and launch flashboard.bat (Type = Windows Batch file). Please follow the prompts on the command line window. The result is shown in Fig. 4.14. When done, it is important that you reboot the crate by turning it off then on again.

Warning: You cannot program the detector board flash before programming the Support Board.



```

Altera Nios II EDS 13.1 [gcc4]
[OpenPET] Do you want to program the detector board?
[OpenPET] If you want to program the detector board, you should have programmed
the SupportBoard's flash and rebooted it. Have you programmed the SupportBoard?
[y/n] y
Is a single USB-Blaster connected? [y/n] y
[OpenPET] Running "nios2-flash-programmer -e --base=0x00 ./bin/DetectorBoard.flash"
Using cable "USB-Blaster [USB-0]", device 1, instance 0x00
Resetting and pausing target processor: OK
Checksummed/read 28kB in 0.6s
Erased 384kB in 3.8s (101.0kB/s)
Programmed 357KB +27KB in 5.7s (67.3KB/s)
Did not attempt to verify device contents
Leaving target processor paused
[OpenPET] Done programming flash. Please reboot chassis.
[OpenPET] Press [Enter] to close this window_

```

Fig. 4.14: Detector Board flash programming script

4.4 Running OpenPET System

Any operating system (Windows, GNU/Linux, or Mac OS) and programmable language (C, C++, Delphi, MATLAB, VB.NET, and VC#) supported by QuickUSB can be used to interface with the OpenPET system. OpenPET provides multiple methods to control and configure the system. The simplest method is to use `openpet.exe` which is a Microsoft Windows executable that can configure and acquire data from an OpenPET system. Additionally, platform independent example Python scripts are also provided to streamline the configuration and acquisition process.

Prior to running the system with `openpet.exe`, you need to have the Small System hardware configured, powered on and loaded with the correct OpenPET firmware and software (see [Assembling the Hardware](#) (page 19) and [Downloading the Software and Firmware](#) (page 22) sections). You also need to plug in a QuickUSB cable to connect the Host PC (USB port) with either the Host PC Interface Board (plug into USB port on the front panel) or Support Board (plug into QuickUSB module directly on SB in back of crate).

4.4.1 Commands

OpenPET utilizes a standard 32-bit wide Serial Peripheral Interface (SPI) to facilitate serial communications between any parent node and its children. The communication protocol follows a [request-response](#) architecture, where a parent node writes a command to a child or multiple children then reads back the response.

OpenPET commands are 80-bits wide as shown in [Fig. 4.15](#). The first 16 most significant bits are the command id, followed by, source address (16 bits), destination address (16 bits), and payload (32 bits).

CMD ID (16 bits)	SRC Addr (16 bits)	DST Addr (16 bits)	Payload (32 bits)
---------------------	-----------------------	-----------------------	----------------------

Fig. 4.15: OpenPET command (80-bits)

Starting from the most significant bit (MSB):

- Command ID is defined below
- SRC/DST source/destination address is defined below
- Payload is defined per command (look in command folder)

The command id (Fig. 4.16) specifies the function of the command, using a 16-bit number. The most significant bit has two uses:

- used as a flag to denote a response/reply/acknowledgment from a node to its parent (direction=child-to-parent).
- used as a flag to denote a non-blocking command i.e., asynchronous command (direction=parent-to-child).

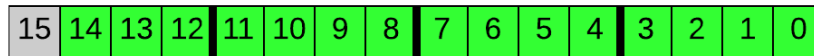


Fig. 4.16: Command ID (16-bits)

Starting from the least significant bit (LSB):

(14:0) Command ID

(15) Dual use flag

- Child sets it to '1' when it responds to a parent
- Parent sets it to '1' when it doesn't want to wait for the targeted child's response, i.e., non-blocking command or asynchronous command.

Note: The targeted child will not reply to other commands if it is still busy executing this asynchronous command.

The source address is a 16-bit number that defines where the command originates. Typically, commands that originate at the Host PC have a source address of 0x4000. The destination address is a 16-bit number that identifies where the response should be received and processed. Both the source and destination addresses have the same address format, as shown in Fig. 4.17. The payload is a 32-bit number that specifies additional information/settings for each command. See [Commands](#) (page 59) chapter for further details.

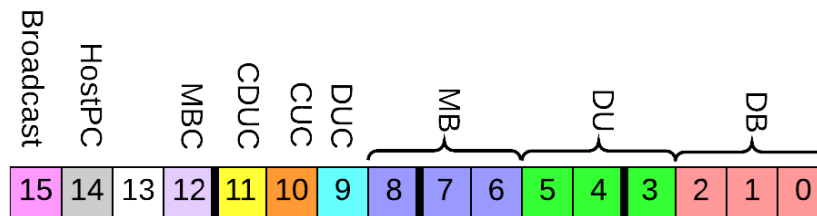


Fig. 4.17: Source/Destination address (16-bits)

Starting from least significant bit (LSB):

(2:0) Detector Board Address

(5:3) Detector Unit Address

(8:6) Multiplexer Board Address

(9) Detector Unit Controller source/destination flag

(10) Coincidence Unit Controller source/destination flag

(11) Coincidence Detector Unit Controller source/destination flag

(12) Multiplexer Board Controller source/destination flag

```
(13) Not used
(14) Host PC source/destination flag
(15) Broadcast flag
```

The executable `openpet` is used to control and configure the system. It has several optional arguments, list them by running `openpet` with `-h` or `--help` switches.

The syntax of sending a command is:

```
$ openpet -c ID DST PAYLOAD
```

where *ID* is the command id, *DST* is the destination address, and *PAYLOAD* is the payload for that command. For example, to send a ping command to a detector board in slot 1:

```
$ openpet -c 0x0001 0x0001 0
```

It is possible to use decimal numbers instead of hex:

```
$ openpet -c 1 1 0
```

Tables 1 and 2 in [Commands](#) (page 59) show the list of the current OpenPET commands available. See [Commands](#) (page 59) for more details, including detailed examples.

4.4.2 Data Acquisition

The `openpet` executable is also used for acquiring data after the system has been correctly initialized. The syntax of sending this command is:

```
$ openpet -a DURATION -o FILENAME
```

where *DURATION* is the acquisition time in seconds or fractions of seconds and *FILENAME* is the output data file name. The content of the file will have the standard OpenPET data format. The `-o` switch is optional. If omitted `openpet` will then generate a unique file name called `%Y%m%d-%H%M%S.openpet.d`, with the four digit year, then month, day, hour, minute, second.

4.5 Example System Setup & Data Acquisition of a Small System in Oscilloscope Mode

This section outlines an example of how to use the executables to initialize the system and take test data, in order to help determine whether your system is working properly. More information on diagnostic testing is also available in [Appendix 2: Troubleshooting Diagnostics](#) (page 107).

We list here an example string of commands – see [Commands](#) (page 59) for more details on these commands. All of the commands below are sent through a the default QuickUSB module index, i.e., 0. On power-up the system configures all components to default values, see [Appendix 1: Default Values](#) (page 105) for all default values.

The system must be properly programmed (see [Programming OpenPET Flash Images](#) (page 29)) before executing any of the commands below. Plug-in at least one detector board to the VME chassis and insert it in slot address 3 (this Detector Board will respond to any broadcast command with the destination address assigned to it in the example). Connect a pulse generator to any channel and set the amplitude to 600mV peak-to-peak and the frequency to 1 KHz. Now we will configure the OpenPET system.

4.5.1 Simple Example

- Set the acquisition mode to scope. Note that the MSB in DST is set to 1 to denote broadcast to all nodes:

```
$ openpet -c 3 0x8003 0x00000001
```

- Set scope settings to 0x02000101. Number of Samples is 16, samples before trigger is 0, and trigger window is 2:

```
$ openpet -c 5 0x8003 0x02000100
```

- Acquire scope data for 10 seconds, save acquired data to test.openpetd

```
$ openpet -a 10 -o test.openpetd
```

The expected data should have a waveform similar to the one shown in Fig. 4.18.

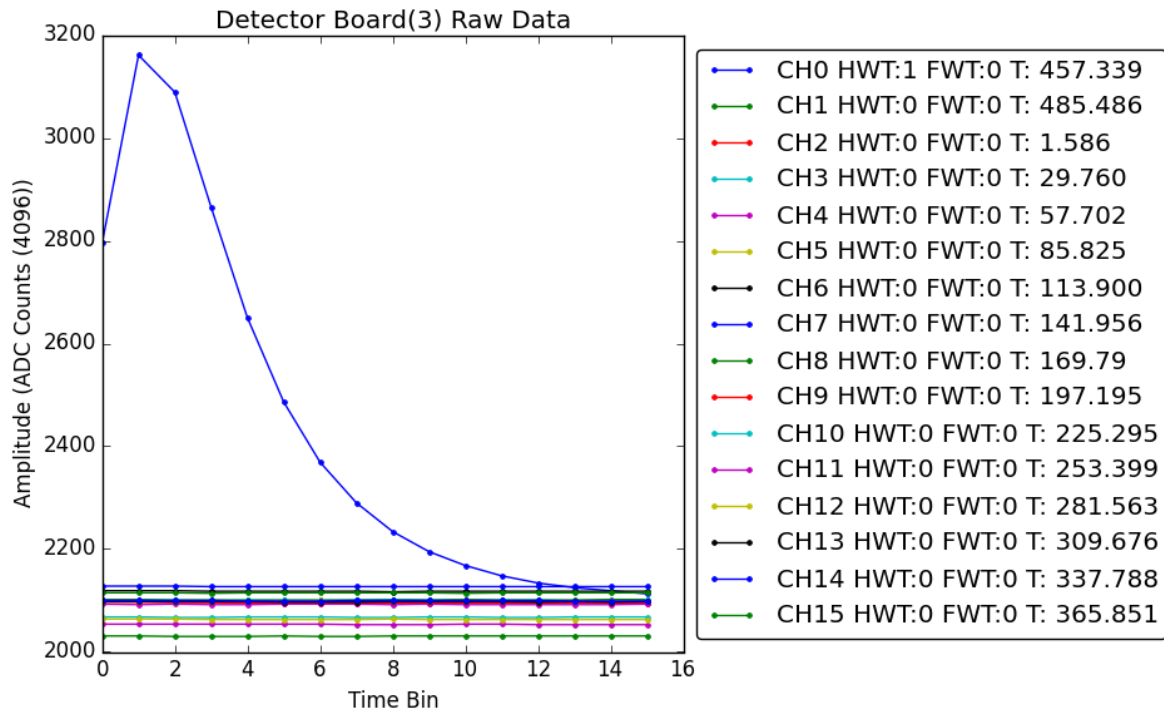


Fig. 4.18: Expected Waveform

4.5.2 Intermediate Example

The following example uses the high-resolution TDC, which is not the default TDC distributed in the version 2.0 firmware. In a later release, the user will be able to select the type of TDC through the command.

- Set the Energy DAC threshold on all DBs to 400mV. Note that the MSB in DST is set to 1 to denote broadcast to all nodes:

```
$ openpet -c 0x0106 0x8003 0x8007E064
```

- Set the acquisition mode to scope. Note that the MSB in DST is set to 1 to denote broadcast to all nodes:

```
$ openpet -c 3 0x8003 0x00000001
```

- Set mode settings to 0x02000101. Number of Samples is 16, samples before trigger is 0, and trigger window is 2:

```
$ openpet -c 5 0x8003 0x02000100
```

- Set trigger mask to allow all channels to trigger

```
$ openpet -c 9 0x8003 0xFFFFFFFF
```

- Reset high-resolution TDC

```
$ openpet -c 0x0101 0x8003 0x00000080
```

- Put high-resolution TDC in calibration mode

```
$ openpet -c 0x0101 0x8003 2
```

- Wait for 5 seconds, for high-resolution TDC to calibrate

```
$ timeout /t 5 (Microsoft Windows)
$ sleep 5 (UNIX/POSIX)
```

- Put high-resolution TDC in normal mode

```
$ openpet -c 0x0101 0x8003 4
```

- Acquire scope data for 10 seconds, data will be saved in an autogenerated file name

```
$ openpet -a 10
```

4.6 Example System Setup & Data Acquisition of a Small System in Singles Mode

Under development.

4.7 Data Analysis

4.7.1 Scope Mode

Scope mode data will be written to a binary file for further analysis and processing. The format for this data is specified in the following sections.

Users can also use an optional analysis tools package based on the ROOT framework, called OpenPET Control and Analysis Tools (OpenPET CAT). See *OpenPET Control and Analysis Tools (OpenPET CAT)* (page 79) for further details on how to install and use OpenPET CAT.

Additionally, an example Python script is provided to plot scope mode data using matplotlib. Data acquired in *Simple Example* (page 35) can be plotted by typing

```
$ python plot.py test.openpetd
```

Data Description and Arrangement

After running the system in scope mode, the DB(s) will periodically collect data until a channel triggers. Once a trigger is detected the raw data is passed from the DB(s) to the CDUC and then to the HostPC. If the CDUC FIFOs are full, the DB will inhibit all further triggers until the CDUC FIFOs can handle more data.

Packets in the data path are 32-bit wide. The four most significant bits are used to identify the packet. *Packet id* is used to verify packets received at a parent as well as identify, validate, and confirm the integrity of the packets in the data path. Below are the list of OpenPET packet ids:

0x0	Reserved
0x1	ADC Data
0x2	Reserved
0x3	Channel Header
0x4	DetectorBoard Header
0x5	DUC Header
0x6	CUC Header
0x7	CDUC Header
0x8	MBC Header
0x9	HostPC Header
0xA-F	Not Used

Single Detector Board

In scope mode, each DB outputs a block of data sequentially as shown in [Fig. 4.19](#). The first packet of the data block is always the 32-bit DB header (depicted in [Fig. 4.20](#)) then a 32-bit channel header (shown in [Fig. 4.21](#)) along with its raw ADC data samples (shown in [Fig. 4.22](#)) then the next channel and its data and so on.

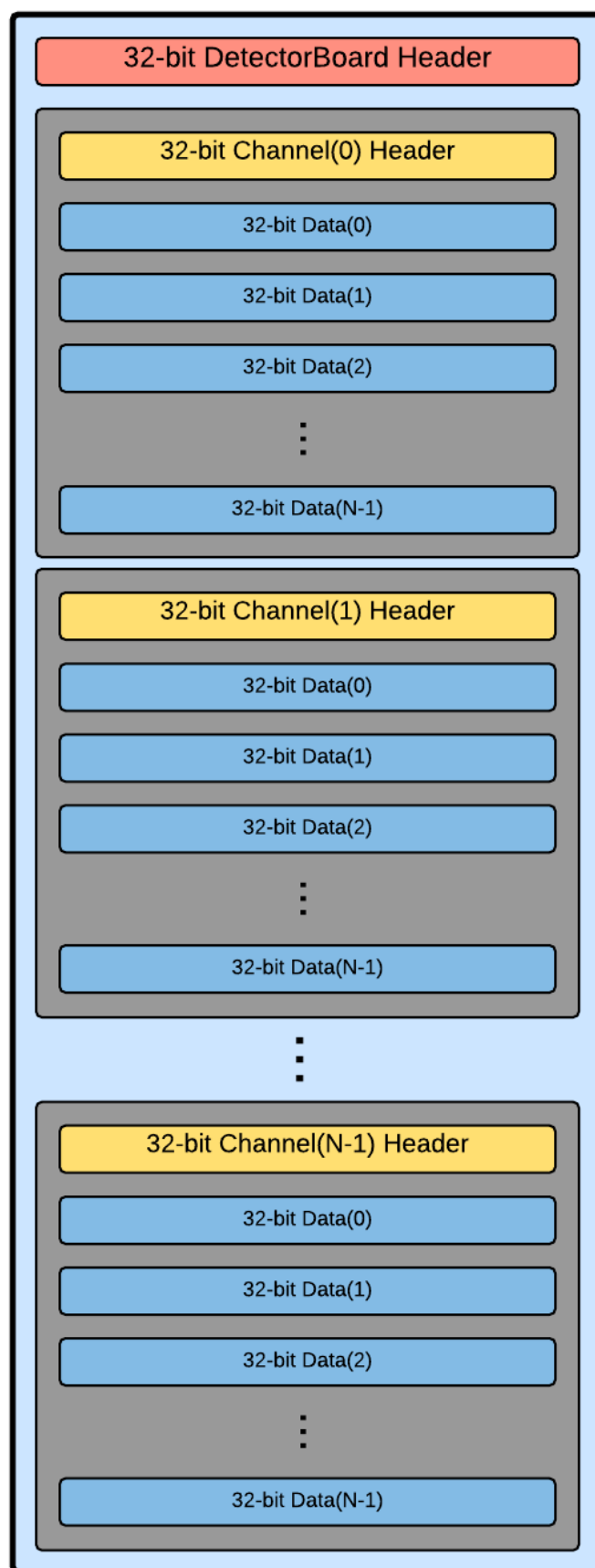


Fig. 4.19: DB Data Block in Scope Mode

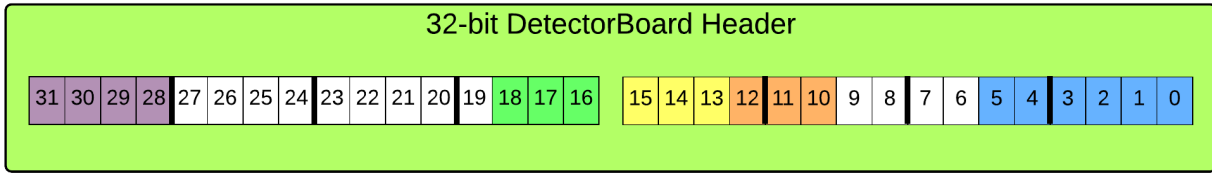


Fig. 4.20: 32-bit DB Header

The 32-bit DB header contains the following information, starting from least significant bit (LSB):

```
(5:0)   Number of channel header packets (i.e. 0 to 63)
(9:6)   not used
(12:10) Detector Board Address (populated by parent)
(15:13) DU Address (populated by parent)
(18:16) MB Address (populated by parent)
(27:19) not used
(31:28) Packet ID (must equal to 0x4)
* More bits are used for future expansions.
```

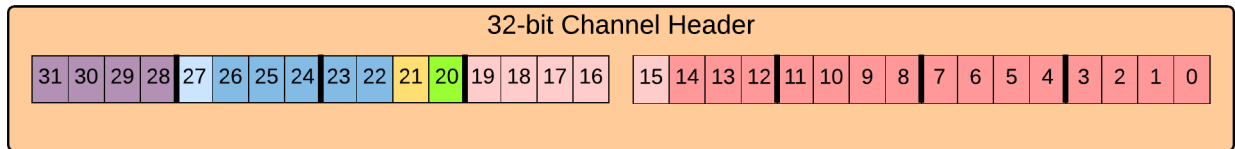


Fig. 4.21: 32-bit Channel Header

The 32-bit DB header contains the following information, starting from least significant bit (LSB):

```
(19:0)   TDC data (if used) (9:0) is fine count, the rest is coarse.
(20)     Hardware trigger hit (energy)
(21)     Firmware trigger hit
(27:22)  Channel address (i.e. 0 to 63)
(31:28)  Packet ID (must equal to 0x3)
* More bits are used for future expansions.
```

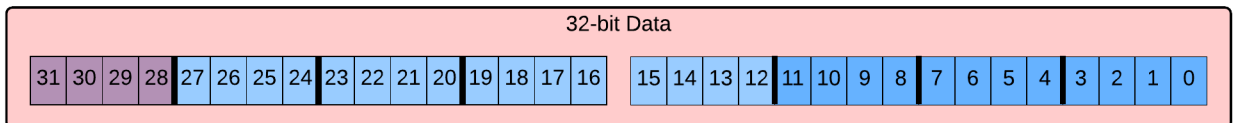


Fig. 4.22: Raw ADC Sample

The 32-bit raw ADC sample contains the following information, starting from least significant bit (LSB):

```
(27:0)   Raw ADC data. Currently (11:0) is utilized.
(31:28)  Packet ID (must equal to 0x1)
* More bits are used for future expansions.
```

Offset	[31..24]	[23..16]	[15..8]	[7..0]	Comment
0	Epoch timestamp				Local Time
1	Reserved				
2	Acquisition Duration				From Software
3	Acquisition Mode Payload				See cmd id 0x0003
4	Acquisition Mode Settings Payload				See cmd id 0x0005
5	Reserved				future data format
6	Misc.Software Settings				processed/raw?
7	Reserved				
...					
9					
10	User Defined				
...					
999					
B=1000	Detector Board Header				M=Num of channels
B+1	Channel Header (0)				
B+2	ADC Sample (0)				N=number of ADC samples. *
B+3	ADC Sample (1)				
	...				
B+N	ADC Sample (N-1)				
B+1+N	Channel Header (1)				
	...				
	Channel Header (M-1)				
	...				
B+M*(1+N)	Detector Board Header				Next Detector Board
	...				
	Detector Board Header				Next Detector Board
	...				

Fig. 4.23: Scope Mode binary file format.

The data is prepended with a header as shown in Fig. 4.23. All the settings including the acquisition time are stored in the header.

Multiple Detector Boards

In scope mode, a detector board will output its data whenever a trigger occurs. In other words, the IO FPGA doesn't arrange or sort detector board data coming from multiple boards. Data blocks are passed to the Main FPGA based on first-come-first-serve scheme. Fairness is preserved by using a one-dimensional queue with a depth of 8 for each IO FPGAs and 4 for Main FPGA. The first data block data enters the queue, will be send to the main FPGA first.

4.7.2 Singles Mode

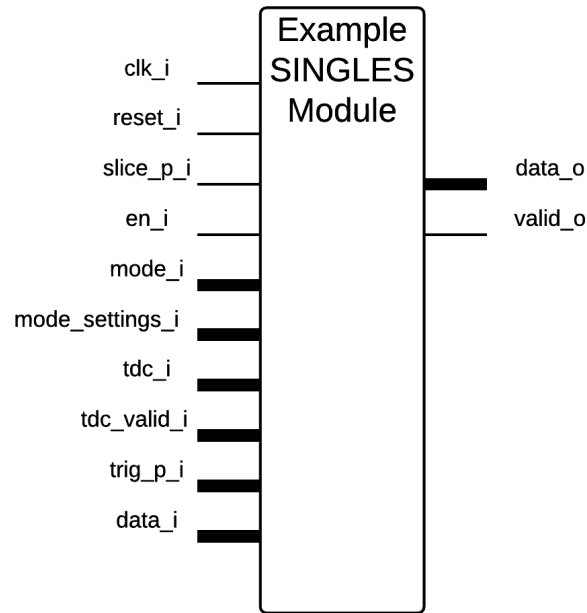


Fig. 4.24: Singles module example

Note: Thick I/O markers denote buses. Thin I/O markers denote single bits.

Data Description and Arrangement

The slide width value N is calculated as R times the slice period divided by the system clock period where R is the system clock period divided by the ADC clock period.

#	32-bit wide word [31:0]
0	Packet 0
1	Packet 1
...	...
N-1	Packet N-1

Fig. 4.25: Singles mode frame definition

Example 1:

```
Default System Clock Period = 12.5ns (80MHz)
Default ADC Clock Period = 25ns (40MHz)
Slice period = 100ns (10MHz)
R = 12.5/25 = 1/2
N = 1/2 * 100/12.5 = 4 packets
Total bits transferred per slice = 4 * 32 = 128 bits
```

Example 2:

```
Default System Clock Period = 12.5ns (80MHz)
Default ADC Clock Period = 25ns (40MHz)
Slice period = 200ns (5MHz)
R = 12.5/25 = 1/2
N = 1/2 * 100/12.5 = 8 packets
Total bits transferred per slice = 8 * 32 = 256 bits
```

Note: N should be an integer. Use ceil() to round to the nearest integer toward infinity.

Fig. 4.26 is an example for N = 4 and for a specific event processing where the energies of each channel of the 16-channel DB are calculated by integrating a fix number of samples of the signal waveforms.

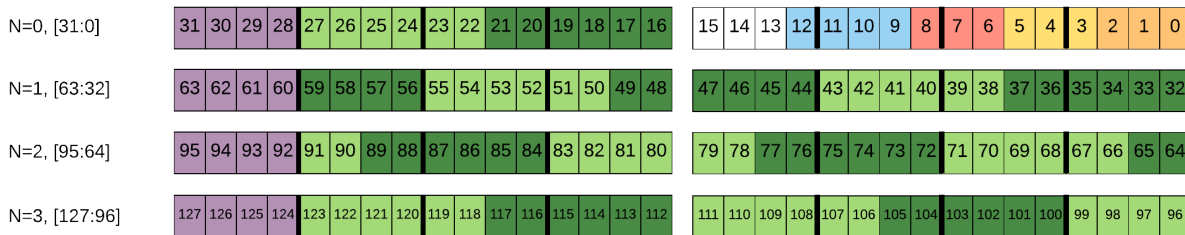


Fig. 4.26: Example when slide width equals 4.

```
Starting from least significant bit (LSB):
(2:0)    Detector Board Address (populated by parent)
(5:3)    Detector Unit Address (populated by parent)
(8:6)    Multiplexer Board Address (populated by parent)
(12:9)   Number of channels that triggered
(15:13)  Not Used
(21:16)  Channel 0 Energy
(27:22)  Channel 1 Energy
(31:28)  Packet ID
(37:32)  Channel 2 Energy
...
(123:118) Channel 15 Energy
(127:124) Packet ID
```

For a given 32-bit packet, the packet ID is the most significant four bits. Packet IDs are used to verify packets received at a parent. It also allows a node to identify, validate, and semi-confirm the integrity of the packet in the data path.

31	30	29	28
----	----	----	----

Fig. 4.27: Example of packet ID payload

Below are the list of singles mode packet IDs:

```
0x0   Reserved. DO NOT USE
0x1   Singles Packet
0x2-F Not used
```

In singles mode, the data file also has a header as shown in Fig. 4.28.

Offset	[31..24]	[23..16]	[15..8]	[7..0]	Comment
0	Epoch timestamp				Local Time
1	Reserved				
2	Acquisition Duration				From Software
3	Acquisition Mode Payload				See cmd id 0x0003
4	Acquisition Mode Settings Payload				See cmd id 0x0005
5	Reserved				future data format
6	Misc.Software Settings				processed/raw?
7	Reserved				
...					
9					
10	User Defined				
...					
999					
B=1000	Singles Packet (0)				Event can originate
B+1	Singles Packet (1)				from any DB.
	...				
B+N-1	Singles Packet (B+N-1)				N = Slice Width
B+N	Singles Packet (0)				Event can originate
B+N+1	Singles Packet (1)				from any DB.
	...				

Fig. 4.28: Singles Mode binary file format

The payload for writing the system acquisition settings in singles mode is given in Fig. 4.29.

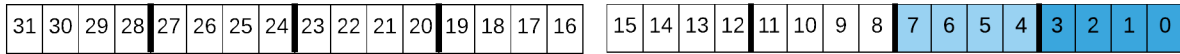


Fig. 4.29: System Acquisition Mode Settings payload for singles mode

```
Starting from least significant bit (LSB)
(3:0)   Total number of ADC clock ticks to finish a single Event computation
        (2^4-1 = 15)
(7:4)   Reserved, max ADC clock ticks to process data (2^8-1 = 255)
(15:8)  Reserved
(31:16) Not Used
```

Note:

- Event computation clocks ticks should be greater than 1.
 - Number of pipeline stages is pre-defined in the firmware as a constant
 - PipelineStages = $\text{ceil}(\text{EventCompuationClockTicks}/\text{SliceWidth}) + 1$, where $\text{ceil}()$ rounds the number to the next highest integer.
-

4.8 Getting Help

The goal of OpenPET is to create an active community of general users and developers, so that we can all pool our resources and expertise. In order to help build this OpenPET support network, we have created an opt-in email list for people that want to keep informed of OpenPET news and updates. You can add your email to this OpenPET email list by sending a message to sympa@lists.lbl.gov with the subject line “subscribe openpet-users@lbl.gov”.

For general questions, please see the documentation available on the OpenPET website, including presentations, publications and user guides (<http://openpet.lbl.gov/documentation/general-documents>). You can also check the website's Frequently Asked Questions at <http://openpet.lbl.gov/users/faq/>.

If you have specific questions, you can read and post questions using the OpenPET forums on our website at <http://openpet.lbl.gov/forums/>. Please check past discussion threads before creating a new one.

Detector Board

The purpose of the Detector Board is to accept analog inputs from the detector modules and convert them into singles event words. Generally, this requires determining the energy, interaction position, and arrival time associated with a single gamma ray interaction that occurs in the detector module, as well as applying as many corrections as possible before the associated singles event word is generated. This board must also have the ability to produce “singles events” that have alternate event formats, which are necessary for debugging, calibration, etc.

In order to process an analog input signal, the Detector Board contains an analog front-end circuit with amplification and filtering to reduce the noise and bandwidth of the signal. The processed analog signal is subsequently digitized by an analog-to-digital converter (ADC) to capture the processed analog signal. For systems that require good timing resolution such as PET, additional circuitry may be implemented in the front end to split the analog input signal into a fast timing path where the signal is amplified with a high-bandwidth amplifier. The amplified signal is then triggered by a fast leading edge discriminator to create a timing pulse for the signal arrival, which is time stamped by a time-to-digital converter (TDC) implemented inside the FPGA.

The back end of the Detector Board primarily consists of an FPGA and static random access memory (SRAM). The FPGA processes the digitized signals from the ADCs and, if necessary, combines information from multiple channels to compute the deposited energy, the interaction position, and the event time. Appropriate calibration correction factors that are stored in the SRAM can also be applied to the data. The computed information are formatted into a singles event word and then transferred to the Support Board (i.e., SB-DUC) via the standard OpenPET Bus IO, as described in the [Bus IO](#) (page 46). In addition, the firmware for the FPGA is loaded into the Detector Board by the Support Board via the standard Bus IO. Different firmware can be loaded into the DB FPGA to perform tasks other than event processing, such as debugging, testing, and calibration.

While the details of the signal processing depend strongly on the details of the detector module, the following is an example describing the processing performed for a block detector module with four analog outputs. Event processing is initiated by the OR of the low to high transitions from each of the timing signals. The total amount of signal observed by each of the four PMTs (A, B, C, & D) is computed by summing the output of each ADC for an appropriate period of time (typically 2-3 times the decay time of the scintillator). If necessary, pulse pile-up correction is also applied. These four signals are then summed to get a raw estimate of the energy ($E=A+B+C+D$), and the appropriate Anger logic estimators are computed ($X=(A+B)/E$, $Y=(A+C)/E$). Note that a fast division algorithm can be implemented using look-up tables in the detector memory. The X and Y values are used to address a crystal map table that resides in the detector memory, and so assign a crystal of interaction via a look-up table. The raw energy E and the crystal of interaction are again used to address another look-up table in the detector memory, and so determine whether the event satisfies the energy window criteria. Each timing signal is input to a TDC that is implemented in the detector FPGA, and so measures the (raw) position of the arrival time of each signal within a time slice, and a timing estimator computed from these digitized arrival times. The crystal of interaction and raw arrival time are used to address a look-up table stored in the detector memory and create a corrected arrival time. Thus, the position (crystal of interaction), energy, and arrival time are all computed. If the event satisfies the energy window criteria, these data are formatted to create a singles event word, and then passed to Support Board through the Bus IO block.

Several versions of the DB will be designed, mostly differing in how the analog inputs are being processed by the

front-end circuitries and in the number of analog input channels per DB. The design details are described below.

5.1 Bus IO

The Bus IO connecting the Detector Board to the Support Board (SB-DUC) is shown schematically in [Fig. 5.1](#). The bus is divided into several blocks.

One block provides the timing signals using 4 LVDS differential pairs. There are two fundamental timing signals - an 80 MHz system clock signal generated on the Support Board and a time slice boundary signal that defines the beginning of a time slice. The Support Board sends a copy of these system clock and time slice boundary signals, which are used to clock data from the Support Board to the Detector Board. Because there is propagation delay within the Detector Board, another copy of the system clock and time slice boundary signals (that is produced by the DB) is used to clock data from the Detector Board to the Support Board. In the OpenPET architecture, the system divides time into small, fixed time slices of either 100 ns (8 clocks) or 200 ns (16 clocks). Alternatively, custom time slices can be designed by modification of the firmware, if needed.

A second block defines 4 LVTTL single-ended lines to control data between the DB and SB using a standard digital serial protocol (e.g., SPI).

A third block in the Bus IO provides 4 LVTTL single-ended lines to program the FPGA on the Detector Board using serial protocols, which are generated by the Support Board.

A fourth block provides 16 differential pairs (grouped into 4 sets of 4 LVDS differential pairs) to transfer singles event words from the DB to SB. All individual operations must occur within a single time slice, which implies that only singles event words that occur in the same time slice can be combined to form a coincident event. A singles event word can be either a 32-bit word (100 ns time slice) or 64-bit word (200 ns time slice). Since it can take significantly longer than a single time slice to fully process a singles event, the system is pipelined so that the processing is divided into smaller steps that each can be completed in a single time slice. During one time slice, each set of 4 differential pairs can pass one singles event word (i.e. maximum of 4 singles event words per time slice). Thus, the maximum singles event rate that can be transferred out of each Detector Board is 40 million events per second.

Another block provides 8 spare LVDS differential pairs between the DB and SB for users to pass any information between these two boards.

The final block in the BUS IO supplies power to the Detector Board. Specifically, the Support Board supplies +5 V, +3.3 V, -5 V, and ground.

[Fig. 5.2](#) details the connections between the Support Board and the Detector Board.

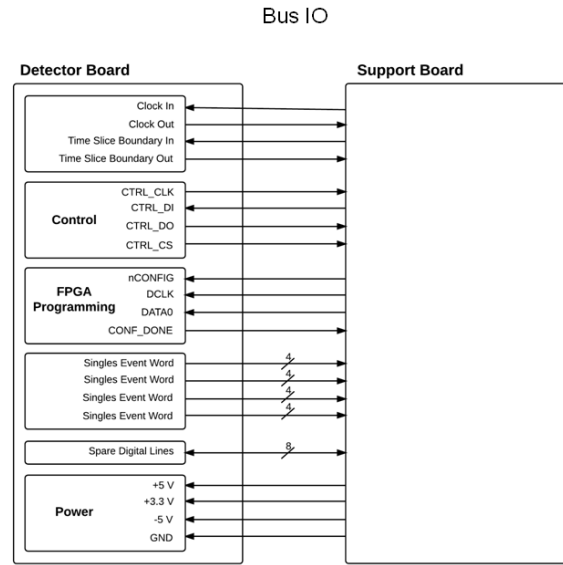


Fig. 5.1: Diagram of the BUS IO.

Pin Assignment for the Connector Between the DB and the SB

	A	B	C
1	D0+	GND	D1+
2	D0-	3.3V	D1-
3	D2+	GND	D3+
4	D2-	+5V	D3-
5	D4+	GND	D5+
6	D4-	-5V	D5-
7	D6+	GND	D7+
8	D6-	3.3V	D7-
9	D8+	GND	D9+
10	D8-	+5V	D9-
11	D10+	GND	D11+
12	D10-	-5V	D11-
13	D12+	GND	D13+
14	D12-	3.3V	D13-
15	D14+	GND	D15+
16	D14-	+5V	D15-
17	GND	-5V	GND
18	CLK_IN+	3.3V	CLK_OUT+
19	CLK_IN-	+5V	CLK_OUT-
20	GND	-5V	GND
21	SLICE_IN+	3.3V	SLICE_OUT+
22	SLICE_IN-	GND	SLICE_OUT-
23	GND	NC	GND
24	SPARE0+	SPARE1+	SPARE2+
25	SPARE0-	SPARE1-	SPARE2-
26	SPARE3+	SPARE4+	SPARE5+
27	SPARE3-	SPARE4-	SPARE5-
28	SPARE6+	SPARE7+	CTRL_CS
29	SPARE6-	SPARE7-	CTRL_CLK
30	CTRL_DO	NC	CTRL_DI
31	DCLK	DETECTOR BIAS	DATA0
32	nCONFIG	3.3V	CONF_DONE

Color	Group Description
Light Blue	VDS Data to DBFPGA & Coincidence Board
Dark Blue	Clock & Slice IN/OUT
Yellow	Undefined pins between DBFPGA & SB FPGA
Pink	Slow control SPI interface signals
Orange	DBFPGA serial programming pins
Green	No connection
Purple	Power & GND
Light Purple	Detector Bias Voltage (100 Vmax.)

Connector is
96 pin VME Connector

On DB
Vector Electronics RE96MSR-062
(Dig+Key part # V1235-ND)

On SB
Vector Electronics RE96FSP
(Dig+Key part # V1240-ND)

Fig. 5.2: Connections between the Support Board and the Detector Board.

5.2 16-Channel Detector Board

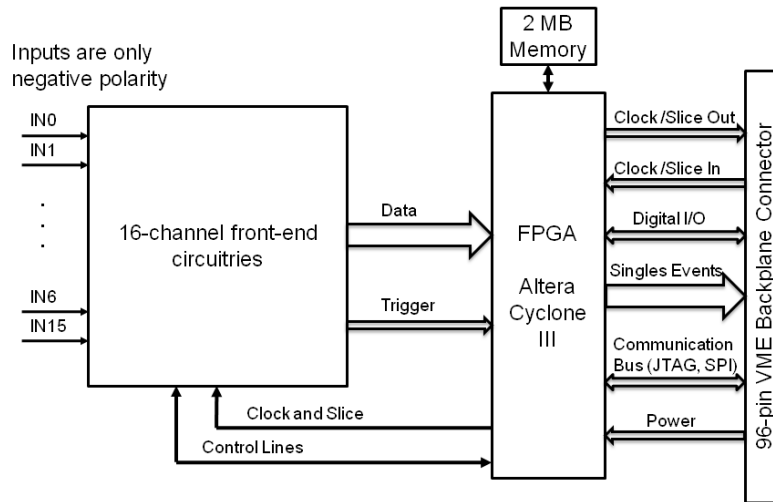


Fig. 5.3: Block diagram of the 16-channel Detector Board.

(Make new version. Replace clock, power, etc lines from the FPGA to 96-pin as BUS IO – i.e. shown in Fig 32?)

A schematic of the block diagram of the 16-channel Detector Board is shown in Fig. 5.3 and a photograph is shown in Fig. 5.4. This DB accepts up to 16 analog input signals, each of which is processed independently. The analog input signal is required to be negative polarity, ground referenced. The input analog signal is terminated with 50 ohms, and then split into two processing chains: an energy chain and a timing chain.

The processing circuit for one channel is shown in Fig. 5.5. The input stage accepts voltages between -0.8 V to 0 V. The input voltage can be attenuated to fall within the acceptable range by changing the attenuation resistor values on the energy chain. Only the signal on the energy chain needs to be attenuated, because the range of the input voltage is limited by the dynamic range of the ADC. In addition, there are 8 external differential LVDS IO pairs that can be used to interface to the DB FPGA. The flexibility of these digital IO allow any digital inputs and/or outputs to be applied to the Detector Board.

The energy chain amplifies the input signal and then splits the amplified signal into a comparator and an anti-aliasing filter with a cut-off frequency of 7 MHz. The comparator provides a trigger signal to start event processing; the trigger threshold is controlled by the energy DAC. The filtered output is sent to an ADC with programmable digital gain that digitizes the analog signal with a sampling rate between 10 MSPS to 65 MSPS.

The timing chain amplifies the input signal with a high-gain high bandwidth amplifier, followed by an ultra fast discriminator that converts the analog signal into a digital timing signal whose leading edge is synchronized to the interaction time. The ADC values and the timing signal are sent to the DB FPGA.

Inside the DB FPGA, a TDC generates a time stamp indicating the arrival time of the timing signal relative to the time slice boundary. The DB FPGA and its memory also analyzes the ADC data from this channel and (potentially) combines it with information from other channels to compute the energy deposit, the interaction position, and the event time. Appropriate calibration correction factors are also stored in the memory and applied to the data.

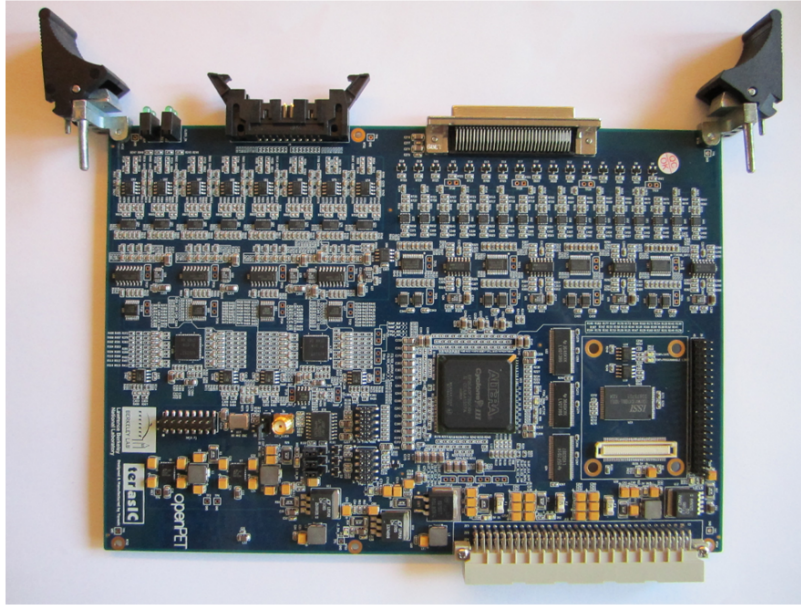


Fig. 5.4: Photograph of the 16-channel Detector Board.

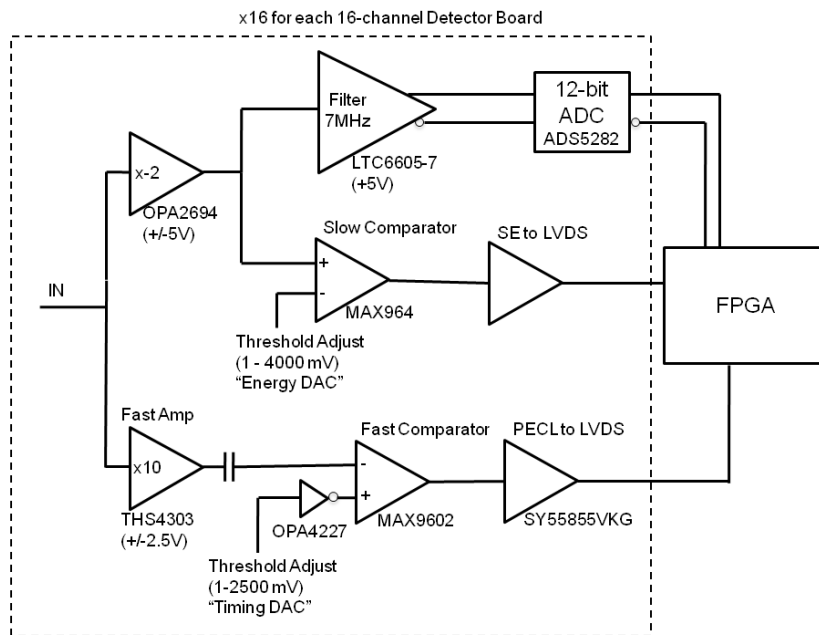


Fig. 5.5: Block diagram of the front-end circuitries of one channel of the 16-channel Detector Board.

5.2.1 Analog Signal Conditioning

Each input signal is required to be negative polarity. The signal is terminated, and then split into two processing chains: timing chain and energy chain. The input stage accepts voltages between -0.8 V to 0 V and has input diodes to protect against over and under voltage. The range of the input voltages is limited in part by the dynamic range of

the ADC. The input voltage can be attenuated to fall within the acceptable range by changing the attenuation resistor values where the signal is split into the timing chain and energy chain. Only the signal on the energy chain needs to be attenuated.

5.2.2 Timing Signal

This circuit is a high-speed leading edge discriminator with a threshold that is controlled via a timing DAC. The timing edge is a high to low transition.

5.2.3 ADC

Each analog input signal is digitized by a 12-bit ADC with digital programmable gain that digitizes the analog signal at a sampling rate ranging from 10 MSPS to 65 MSPS.

5.2.4 TDC

Each analog input signal is digitized by a 12-bit ADC with digital programmable gain that digitizes the analog signal at a sampling rate ranging from 10 MSPS to 65 MSPS.

5.2.5 Detector Memory

Two MBytes of SRAM memory is attached to and controlled by the DB FPGA. This control also includes loading the contents of the memory.

5.2.6 Analog Input Connections

Fig. 5.6 and the discussion below detail the analog input connections to the 16-channel Detector Board. Note that these inputs can be from a user-supplied analog conditioning circuit board (e.g., a preamplifier board), so the connector contains pins used to provide power to and communicate with one of these (optional) circuit boards.

Analog Inputs:

There are 16 analog input channels on the 16-channel DB. Each is a single-ended, negative polarity input signal with a negative-going leading edge. Input voltage levels should be between -0.8 V and 0 V.

Power and Ground:

The 16-channel DB is capable of supplying power to an analog conditioning circuit board. It supplies +5V, -5V, and ground. It is assumed that this is “digital quality” power - that the analog conditioning board will use these as inputs to on-board regulators to create analog quality power, as well as whatever other digital voltages are necessary. The maximum current for each of the voltage supplies is 1 A.

Connector:

We use a 68-pin D-sub connector with 0.050” Pitch x 0.100 Row to Row. The part number is AMP-Part-5787169-7 (Digi-Key part # A33512-ND).

PIN#	DESCRIPTION	DESCRIPTION	PIN#
1	+5V	-5V	2
3	GND	IN 16	4
5	GND	IN 15	6
7	GND	IN 14	8
9	GND	IN 13	10
11	GND	IN 12	12
13	GND	IN 11	14
15	GND	IN 10	16
17	GND	IN 09	18
19	GND	IN 08	20
21	GND	IN 07	22
23	GND	IN 06	24
25	GND	IN 05	26
27	GND	IN 04	28
29	GND	IN 03	30
31	GND	IN 02	32
33	GND	IN 01	34
35	+5V	-5V	36
37	GND	GND	38
39	GND	GND	40
41	GND	GND	42
43	GND	GND	44
45	GND	GND	46
47	GND	GND	48
49	GND	GND	50
51	GND	GND	52
53	GND	GND	54
55	GND	GND	56
57	GND	GND	58
59	GND	GND	60
61	GND	GND	62
63	GND	GND	64
65	GND	GND	66
67	GND	GND	68

Color	Group Description
	Power & GND
	Analog Input to DB

Connector on DB is
68 pin D-Sub Connector
AMP Part 5707100-7
0.050" Pitch x 0.100" Row to Row
Digi-Key Part # A23512-ND

Fig. 5.6: Pin assignment for the Analog Input Connector to the 16-channel Detector Board.

Support Board

In an OpenPET Support Crate, the custom Support Board is mounted as a backplane on a standard 12-slot VME chassis that accommodates 6U boards. This Support Board primarily plays two roles, depending on the firmware (Fig. 3.1). In a Detector Unit for a standard- or large-sized system, the Support Board is loaded with detection firmware and acts as a Detector Unit Controller (DUC). In a Coincidence Unit for a standard- or large-sized system, the Support Board is loaded with coincidence firmware and acts as a Coincidence Unit Controller (CUC).

The Support Board also plays a third role for the special case of a small system when it is configured as a Coincidence Detector Unit Controller (CDUC), which interfaces with the detector boards and performs coincidence functions. Basically the CDUC performs the functions of both the CUC and DUC.

6.1 Support Board with Detection Firmware

In a Detector Unit, the main purpose of the Support Board with detection firmware (i.e., DUC) is to accept singles event words from multiple Detector Boards, multiplex them, and pass these singles event words to the Coincidence Interface Board. In addition, it provides the control and power for the Detector Boards.

Shown schematically in Fig. 6.1, the Support Board acting as a DUC services up to 8 Detector Boards. The standard OpenPET Bus IO circuit, as described in *Bus IO* (page 46), connects the Support Board to each Detector Board. Three FPGAs on the Support Board multiplex the singles event words and pass them through the slot 8 Bus IO block to the Coincidence Interface Board. The event multiplexing and forwarding is shared among the three FPGAs (one Main FPGA and two slave IO FPGAs) due to limited pin count. Several other blocks, such as a clock-conditioning block that ensures the fidelity of the system clock, logic analyzer connectors, two RS-232 ports, and diagnostic LEDs, are not shown in Fig. 6.1.

High-level commands are sent via USB (or alternatively, through Ethernet or Fiber-Optic) from the Host PC to the Coincidence Unit and ultimately down to the Detector Unit Controller's Main FPGA, which interprets and executes these commands. This execution may involve controlling the Detector Board, such as by loading a program into the FPGA on the DB. Or it may involve higher-level functions, such as performing a calibration by instructing the DB to produce calibration data, analyzing the forthcoming calibration events, computing calibration parameters, and loading these parameters into the detector memory on the DB.

6.1.1 IO FPGAs

There are two slave IO FPGAs on the Support Board. These act primarily as a multiplexer for singles events, each taking up to 16 individual singles events that it can receive in a single time frame and passing up to 4 of them to the Main FPGA. Obviously, there is some possibility for data loss, and the multiplexing algorithm is designed to ensure that this loss is unbiased. Each IO FPGA also serves as a fan-in and fan-out for communication between the Main FPGA and the individual Detector Boards, and the two IO FPGAs can communicate with each other.

The digital signals between each IO FPGA and the Main FPGA are identical to the standard OpenPET Bus IO signals (Fig. 5.1): lines for the clock and Time Slice Boundary (both directions), 4 control lines, 4 FPGA programming lines, 16 event data lines, and eight user-definable data lines. There are also 32 user-definable digital lines between the two IO FPGAs, 16 sending data in each direction.

6.1.2 Main FPGA

A single physical Main FPGA performs the logical functions of both the master FPGA and the support microprocessor. Its FPGA-like functions are mostly limited to passing events from the IO FPGAs to the Coincidence Interface Board (providing multiplexing, if necessary).

Some of the logic blocks in the Main FPGA can be programmed using Nios II to be identical to microprocessor hardware, which then runs executable files programmed in C. This support microprocessor receives high-level commands from the Host PC via USB (or Gigabit Ethernet or Fiber-Optics), and then interprets and executes these commands. It is responsible for loading all the programs into the IO FPGAs and DB FPGA, as well as the contents of all the support memory, detector memory, and all other registers that are on the DB and SB. It also monitors the event stream and can insert diagnostic information (such as event rates) into the event stream or provide this information directly to the Host PC. Whenever possible, calibration routines are also performed on the support microprocessor.

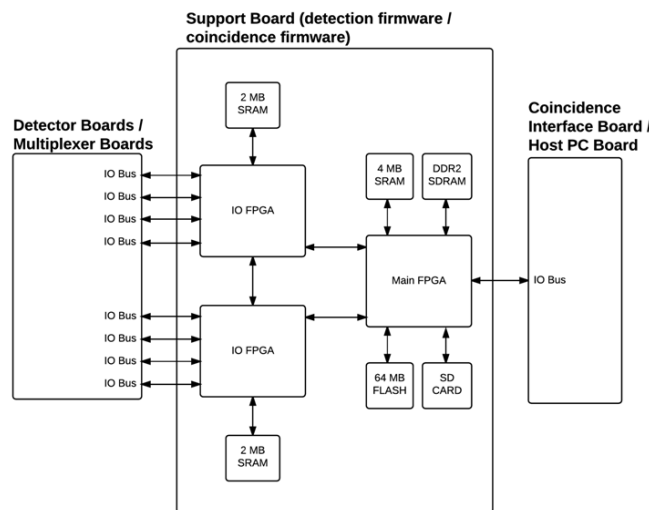


Fig. 6.1: Schematic of the Support Board, which is loaded with either detection firmware (when used in a Detector Unit) or coincidence firmware (when used in a Coincident Unit.)

6.1.3 Support Memory

There are multiple forms of memory on the Support Board, as shown Fig. 6.1. The SRAM is accessed by the FPGA and provides its output within 1 clock cycle of being addressed so is both reasonably fast and deterministic, which greatly simplifies incorporating it within FPGA algorithms. However, the SRAM capacity is fairly small. Thus, the Main FPGA is connected to 4 MB of SRAM and each of the IO FPGAs is connected to 2 MB of SRAM, for a total of 8 MB of SRAM on each Support Board. This memory is typically used to store look-up tables that apply real time calibration and corrections to the event data.

As the Main FPGA also emulates a microprocessor, RAM memory and disk storage are also necessary for it to function effectively. The RAM memory is provided via up to 1 GB of RAM that can be plugged into a DDR2 SDRAM connector (identical to that typically found in laptop computers). The disk storage is provided by a SD card (identical to that found in digital cameras) that is plugged into a SD card slot, when users desire disk storage that can be easily removed. Otherwise, standard disk storage is provided by a 64 MB FLASH memory chip that is connected to the Main FPG, in order to store the FPGA firmware needed for DB FPGAs, contents of all the support SRAM memory, detector memory, and all other registers that are on the DB and SB. This information can also be stored in the on-board EPCS memory, which is where it resides in the initial release.

6.1.4 Clock Conditioning

The clock-conditioning block consists of a PLL (phase-locked loop) that regenerates the system clock signal from the Support Board in a Coincidence Unit and passes it to the Support Board FPGAs in a Detector Unit and then to the Detector Boards. The block also includes space for a local clock oscillator, which is used to provide the system clock when the system is being used without a Coincidence Unit (i.e., when the support microprocessor passes events directly to the Host PC).

6.1.5 Connectors

The clock-conditioning block consists of a PLL (phase-locked loop) that regenerates the system clock signal from the Support Board in a Coincidence Unit and passes it to the Support Board FPGAs in a Detector Unit and then to the Detector Boards. The block also includes space for a local clock oscillator, which is used to provide the system clock when the system is being used without a Coincidence Unit (i.e., when the support microprocessor passes events directly to the Host PC).

6.1.6 Slots 0-7

Slots 0-7 in a Detector Unit each contain a Detector Board. Details on the function and design of the Detector Board are available in [Detector Board](#) (page 45).

6.1.7 Slots 8-11

The rightmost four slots (slot numbers 8 through 11) each contain a board with a specific purpose, but in general are used to facilitate connection to and communication between various parts of the system.

Coincidence Interface Board (Slot 8)

The purpose of the board in slot 8 is to communicate with the Coincidence Unit. The formats of the signals that are passed between the Detector Unit and the Coincidence Unit via the Coincidence Interface Board are identical to those being passed between the Detector Board and the Support Board on slots 0-7. The standard Bus IO is used in both cases.

There are two versions of the Coincidence Interface Board, one called Coincidence Interface Board CI-1 to interface with the Multiplexer Board MB-1 in the Standard System and another called Coincidence Interface Board CI-8 to interface with the Multiplexer Board MB-8 in the Large System. CI-1 is a passive board with no active components—just traces connecting the front panel and rear connectors. The CI-1 front panel connector then connects to a cable that brings these signals to MB-1 in the Coincidence Unit. On the other hand, CI-8 will have active components (e.g., FPGA, etc.) to interface with MB-8, although it has not yet been designed. These boards are only necessary if the system contains a Coincidence Unit.

Host PC Interface (Slot 9)

The purpose of the board in slot 9 is to interface with the Host PC. The Host PC Interface front panel contains an Ethernet connector, USB connector, SD card connector, three reset switches, 20 LEDs, and a detector bias voltage input (BNC, maximum 100 V, positive or negative polarity). The board itself holds a Gigabit Ethernet transceiver chip, as well as a connector that a QuickUSB card must be plugged into in order for the USB communication to function. In the initial release, only communication through the QuickUSB connector is supported.

User IO (Slot 10)

The purpose of the board in slot 10 is to provide User IO. It has an external clock input, two DB9 RS-232 connectors that are connected to the Main FPGA (which can be used to communicate to motor controllers, etc.) and 48 digital IO lines. An on-board jumper selects whether these 48 lines use 5 V or 3.3 V logic level. Each of the three FPGAs (the Main and the two IO) is connected to 16 IO lines. On board jumpers select the direction (input or output) of each IO line in groups of 4 (e.g., the direction Main FPGA IO lines 0-3 are set by a single jumper, and so must be the same).

Debugging (Slot 11)

The purpose for the board in slot 11 is Debugging. It contains a JTAG connector (that can be used to program the FPGAs directly), four Aligent 16902B connectors for logic analyzers (two connect to the Main FPGA, and one to each of the two IO FPGAs), and 30 user-defined LEDs (10 connected to each of the 3 FPGAs).

6.2 Support Board with Coincidence Firmware

In a Coincidence Unit, the Support Board acts as a Coincidence Unit Controller (CUC) when it is loaded with coincidence firmware, as shown in [Fig. 6.1](#). It also provides control and power for the Multiplexer Boards that are located in slots 0-7.

In the general data flow, singles event words are passed through a Coincidence Interface Board to a Multiplexer Board, which can provide a further layer of multiplexing for singles event words, if necessary. These multiplexed singles event words are then passed to the Support Board with coincidence firmware, which searches through the singles event words from multiple (up to eight) MBs for pairs that are in time coincidence and then forms coincidence event words. These coincidence event words are then passed to the Host PC through the Host PC Interface Board. Optionally, the Coincidence Unit Controller can act as a multiplexer and forward unaltered singles event words to the Host PC.

When the Support Board acts as a Coincidence Unit Controller, the role of the three FPGAs is similar to that described above. Each IO FPGA acts as a multiplexer for singles event words, taking up to 16 individual singles events from MBs and passing up to four of them to the Main FPGA, using a multiplexing algorithm that ensures unbiased loss. The Main FPGA acts as both a master FPGA and a support microprocessor. As a master FPGA, it primarily passes events from the Main FPGA to the Host PC Interface Board. As a support microprocessor, Nios II is used to program logical blocks in the FPGA to run executable files programmed in C. For instance, the Main FPGA is used to identify and form coincident event words.

High-level commands are sent via USB (or Ethernet or Fiber-Optic) from the Host PC to the Coincidence Unit. The NIOS II microprocessor is not connected directly to these communication interfaces. Instead, the NIOS II microprocessor talks to the communication interfaces through the register array implemented in the Main FPGAs.

6.3 Support Board with Detection & Coincidence Firmware

In the special case of a Small System, the Support Board is also capable of identifying coincident pairs of singles event words, formatting them into coincidence event format, and passing them to the support microprocessor (Main FPGA),

which then passes them to the Host PC. Thus it can act as a full-featured PET data acquisition system, albeit with a limited number of input channels and output event rate capability. It can also be programmed to multiplex singles events and pass them to the Host PC.

Commands

OpenPET utilizes a standard 32-bit wide Serial Peripheral Interface (SPI) to facilitate serial communications between any parent node and its children. The communication protocol follows a [request-response](#) architecture, where a parent node writes a command to a single child or multiple children and reads back the response. For a gentle introduction on OpenPET commands, see the [Commands](#) (page 32) section in Getting Started.

OpenPET commands are 80-bits wide as shown in [Fig. 7.1](#). The first 16 most significant bits are the command ID, followed by the source address (16 bits), destination address (16 bits), and payload (32 bits).

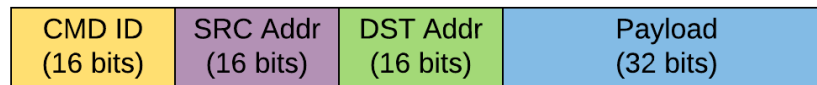


Fig. 7.1: OpenPET command (80-bits)

Starting from the most significant bit (MSB):

- Command ID is defined below
- SRC/DST source/destination address is defined below
- Payload is defined per command (look in command folder)

The command id ([Fig. 7.2](#)) specifies the function of the command, using a 16-bit number. The most significant bit has two uses:

- used as a flag to denote a response/reply/acknowledgment from a node to its parent (direction=child-to-parent).
- used as a flag to denote a non-blocking command i.e., asynchronous command (direction=parent-to-child).



Fig. 7.2: Command ID (16-bits)

Starting from the least significant bit (LSB):

(14:0)	Command ID
(15)	Dual use flag
	(a) Child sets it to '1' when it responds to a parent

(b) Parent sets it to '1' when it doesn't want to wait for the targeted child's response, i.e., non-blocking command or asynchronous command.
 Note: The targeted child will not reply to other commands if it is still busy executing this asynchronous command.

The source address is a 16-bit number that defines where the command originates. Typically, commands that originate at the Host PC have a source address of 0x4000. The destination address is a 16-bit number that identifies where the response should be received and processed. Both the source and destination addresses have the same address format, as shown in Fig. 7.3.

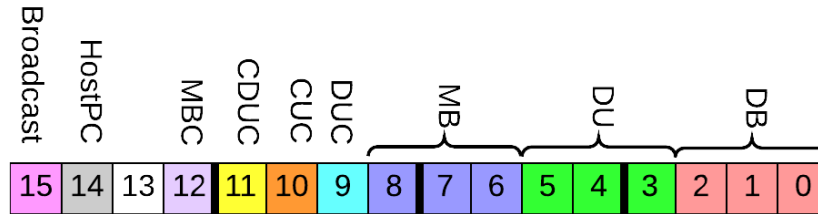


Fig. 7.3: Source/Destination address (16-bits)

Starting from least significant bit (LSB):

(2:0)	Detector Board Address
(5:3)	Detector Unit Address
(8:6)	Multiplexer Board Address
(9)	Detector Unit Controller source/destination flag
(10)	Coincidence Unit Controller source/destination flag
(11)	Coincidence Detector Unit Controller source/destination flag
(12)	Multiplexer Board Controller source/destination flag
(13)	Not used
(14)	Host PC source/destination flag
(15)	Broadcast flag

If the Broadcast flag is set in the destination address, the source node will pass the command down to all of its “children” and the child specified in the destination address will be read back to in order to create the response.

If the CUC or CDUC flags are set in the destination address, that corresponding unit will execute the command and respond. Namely, the response will not come from the unit specified in the destination address.

Finally, the payload is a 32-bit number that specifies additional information for each command; see below for examples.

Any operating system (Windows, GNU/Linux, or Mac OS) and programmable language (C, C++, Delphi, MATLAB, VB.NET, VC#, and Delphi) supported by QuickUSB can be used to interface with the OpenPET system. OpenPET provides multiple methods to control and configure the system. The simplest method is to use `openpet.exe` which is a Microsoft Windows executable that can configure and acquire data from an OpenPET system. Additionally, platform independent example Python scripts are also provided to streamline the configuration and acquisition process.

The executable `openpet` is used to control and configure the system. It has several optional arguments, list them by running `openpet` with `-h` or `--help` switches:

```
usage: openpet [-h] [-v] [-l] [-d DEVICEINDEX]
             [-c ID DST PAYLOAD | -a DURATION | -sr FILE DST SIZE OFFSET | -sw FILE DST OFFSET]
             [-o FILE] [-t TIMEOUT] [-n RETRIES]
```

optional arguments:

-h, --help	show this help message and exit
-v, --verbose	Show debugging info.
-l, --list	List quickusb devices.
-d DEVICEINDEX, --device DEVICEINDEX	Quickusb device index. List devices to see indexes. DEFAULT=0
-c ID DST PAYLOAD, --command ID DST PAYLOAD	Sends a command to a destination module with a specific payload.
-a DURATION, --acquire DURATION	Acquire data for the specified duration (in seconds). Partial seconds are OK.
-sr FILE DST SIZE OFFSET, --sram-read FILE DST SIZE OFFSET	Reads SRAM contents from OpenPET and writes it to FILE.
-sw FILE DST OFFSET, --sram-write FILE DST OFFSET	Writes FILE contents to SRAM.
-o FILE, --outputfile FILE	File name to save acquired data.
-t TIMEOUT, --timeout TIMEOUT	Timeout duration (in seconds) between retries. Partial seconds are OK. DEFAULT=0.200
-n RETRIES, --retries RETRIES	Number of times I should try to contact OpenPET System before giving up. DEFAULT=20

Tables 8.1 and 8.2 show a list of the current OpenPET commands, including their command ID and a brief description of their function. The payload is specified for each command in the examples that follow.

Table 7.1: Summary of the OpenPET system commands.

IDs	Name	Function
<i>0x0001</i> (page 65)	Ping	Sends a single ping request to destination.
<i>0x0002</i> (page 65)	Write Children Bitstream	Command Support Board(s) to configure all children boards from bitstream stored in EPCS.
<i>0x0003</i> (page 66)	Write System Acquisition Mode	Sets the system mode register in firmware to idle, scope, singles, etc.
<i>0x0004</i> (page 66)	Read System Acquisition Mode	Gets the system mode register from firmware.
<i>0x0005</i> (page 67)	Write System Acquisition Mode Settings	Sets the system mode settings register in firmware for the mode selected.
<i>0x0006</i> (page 68)	Read System Acquisition Mode Settings	Gets the system mode settings register from firmware.
<i>0x0007</i> (page 68)	Write System Acquisition Mode Action	Sets the system mode action register in firmware to reset, start, stop.
<i>0x0008</i> (page 69)	Read System Acquisition Mode Action	Gets the system mode action register from firmware.
<i>0x0009</i> (page 69)	Write Trigger Mask	Sets a mask to suppress triggers.
<i>0x000A</i> (page 70)	Read Trigger Mask	Gets the mask from firmware.
<i>0x000B</i> (page 70)	Write SRAM Data	Writes to external SRAM device. Auto-increments address.

0x000C (page 71)	Read SRAM Data	Reads from external SRAM device. Auto-increments address.
0x000D (page 71)	Zero out SRAM	SRAM content is zeroed out.
0x000F (page 71)	Reset	Reset all configurations.
0x0101 (page 72)	Write TDC Configuration	Sets the TDC control register.
0x0102 (page 72)	Read TDC Configuration	Gets the TDC control register.
0x0103 (page 73)	Reset ADC Configuration	Command Detector Boards(s) to set ADCs registers to OpenPET default values.
0x0104 (page 73)	Write ADC Register	Writes a register directly on ADC(s). See ADC datasheet for valid register maps.
0x0105 (page 74)	Reset DAC Configuration	Command Detector Board(s) to set DACs registers to OpenPET default values.
0x0106 (page 75)	Write DAC Register	Write a register directly on DAC(s). See DAC datasheet for valid register maps.
0x0107 (page 77)	Write Sawtooth pulse(s)	Command DAC(s) to send sawtooth pulses for a given duration of time.
0x0108 (page 77)	Write Firmware Threshold	Sets a firmware threshold level to trigger on.
0x0109 (page 78)	Read Firmware Threshold	Gets the firmware trigger threshold.

Table 7.2: Summary of the OpenPET error codes for replies.

IDs	Name	Function
0x0000	(Reserved) Busy	Child doesn't have anything to reply yet.
0xFFFF	(Reserved) Dead	Dead, nonexistent, or not programmed.
0x7F00	SW Command is Unknown	Software (Running on NIOS) command id is unknown to node.
0x7F01	SW Command Timed Out	Software (Running on NIOS) command id has timed out.
0x7F02	Targeted Child is Dead	Targeted child is dead, nonexistent, or not programmed.
0x7F03	Targeted Child is Busy	Targeted child is busy processing previous command.
0x7F04	FW Command is Unknown	Firmware (Running on FPGA fabric) command id is unknown to node.
0x7F05	FW Command Timed Out	Firmware (Running on FPGA fabric) command id has timed out.
0x7F06	SW incomplete QuickUSB packet	Software (Running on NIOS) received incomplete OpenPET command thru QuickUSB
0x7F07	SW QuickUSB interrupt routine can't keep up	Software (Running on NIOS) received QuickUSB packets faster than it can handle.

7.1 Description and Examples of Command IDs

7.1.1 Command ID: 0x0001

Description: Sends a single ping request to destination. If broadcast on DST was specified, then the read back will be performed on the address provided.

Payload: None e.g. 0 or any value.

Examples:

Send a ping command using the `openpet` executable to destination 0x0002 i.e., DB in slot 2 with 0 payload. `openpet` will print out the date, time, message type, [S]ENT or [R]ECEIVED indicator, command id, destination (when sending) or source (when receiving) address, and the payload. Note that the MSB of the command ID is set to '1' when receiving.:

```
$ openpet -c 0x0001 0x0002 0

2015-09-01 12:30:00,529 INFO [S] 0x0001 0x0002 0x00000000
2015-09-01 12:30:00,733 INFO [R] 0x8001 0x0002 0x00000000
```

Just like the first example, however, using decimal numbers instead of hex:

```
$ openpet -c 1 2 0
```

The destination address has the broadcast flag set to 1. This will cause the SB to send the ping command to all its children, however, the reply will be read back only from 0x0002:

```
$ openpet -c 0x0001 0x8002 0
```

The asynchronous flag in the command id is set to 1. This will cause the SB to immediately respond to the HostPC regardless if the ping command was successful on 0x0002 or not. This feature becomes useful when sending commands that take minutes to complete. Note, that the targeted destination will not be able to respond until it completes the execution of the asynchronous command, however, other nodes can be accessed and controlled independently:

```
$ openpet -c 0x1001 0x0002 0
```

QuickUSB device index 1 is used instead of the default device:

```
$ openpet -d 1 -c 1 2 0
```

The HostPC is willing to wait 1 second for the SB to provide a valid reply. The default timeout per try is 200ms:

```
$ openpet -t 1 -c 1 2 0
```

The HostPC tries a maximum of 3 times before giving up on the SB. The wait between each trial is 0.5 second:

```
$ openpet -n 3 -t 0.5 -c 1 2 0
```

More debugging information is displayed:

```
$ openpet -v -c 1 2 0
```

7.1.2 Command ID: 0x0002

Description: FPGA bitstream configuration is read from EPCS flash memory then written to all children. This command is executed on power-up by default.

Payload: None e.g. 0 or any value.

Examples:

Ask CDUC to configure all of its children i.e., detectorboard FPGAs (not io):

```
$ openpet -c 2 0x0800 0
```

7.1.3 Command ID: 0x0003

Description: Writes the System Acquisition Mode register in firmware.

Payload:

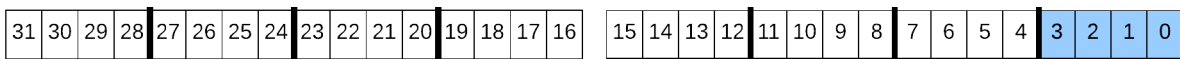


Fig. 7.4: System Acquisition Mode payload

```
Starting from least significant bit (LSB)
(3:0) Mode
(31:4) not used
```

Available Modes: 0x0 - IDLE, 0x1 - Oscilloscope, 0x2 - Singles

Examples:

Broadcast to all nodes to set System Acquisition Mode to scope mode. The command acknowledgment is received from DB in slot 3:

```
$ openpet -c 3 0x8003 1
```

7.1.4 Command ID: 0x0004

Description: Reads the System Acquisition Mode register from firmware.

Payload: None e.g. 0 or any value.

Examples:

Read the System Acquisition Mode register from firmware from slot 5. The payload of the reply is set to mode that previously set i.e., 1=scope mode:

```
$ openpet -c 4 5 0xDEADFEED

2015-09-02 11:10:00,528 INFO [S] 0x0004 0x0005 0xDEADFEED
2015-09-02 11:10:00,732 INFO [R] 0x8004 0x0005 0x00000001
```

7.1.5 Command ID: 0x0005

Description: Writes the System Acquisition Mode Settings register in firmware. The 32-bit setting payload value depends on the mode set in Command ID 0x0003.

Payload:

- Payload when System Acquisition Mode = 0x1 = Oscilloscope (scope mode):

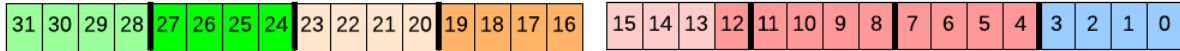


Fig. 7.5: System Acquisition Mode Settings payload for oscilloscope mode

Starting from least significant bit (LSB)

```
(3:0)   Reserved: Must be 0001
(12:4)  Total Number of ADC samples, see notes below (zero is accounted for)
(15:13) Reserved
(19:16) Number of ADC samples before energy trigger ( $2^4 = 16$ )
(23:20) Reserved
(27:24) Trigger window ( $2^4 = 16$ )
(31:28) Reserved
```

- Payload when System Acquisition Mode = 0x2 = Singles (singles mode):

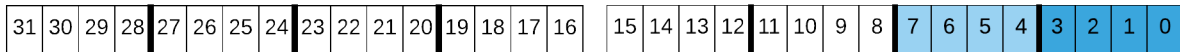


Fig. 7.6: System Acquisition Mode Settings payload for singles mode

Starting from least significant bit (LSB)

```
(3:0)   Total number of ADC clock ticks to finish a single Event computation
        ( $2^4 - 1 = 15$ )
(7:4)   Reserved, max ADC clock ticks to process data ( $2^8 - 1 = 255$ )
(15:8)  Reserved
(31:16) Not Used
```

Notes:

- **Scope Mode:**

- Total Number of samples should be greater than samples before trigger + trigger window.
- Total Number of samples should not exceed Firmware's maximum number of samples - (Number of Channel headers + Detector Board Header)

- **Singles Mode:**

- Event computation clocks ticks should be greater than 1.
- Number of pipeline stages is pre-defined in the firmware as a constant
- $\text{PipelineStages} = \text{ceil}(\text{EventComputationClockTicks} / \text{SliceWidth}) + 1$, where $\text{ceil}()$ rounds the number to the next highest integer.

Examples:

Broadcast to all nodes to set System Acquisition Mode to scope mode. Then use data format = 0, 16 samples, 0 samples before trigger, and a trigger window to 2:

```
$ openpet -c 3 0x8003 1
$ openpet -c 5 0x8003 0x02000100
```

Broadcast to all nodes to set System Acquisition Mode to singles mode. Then TODO FIXME:

```
$ openpet -c 3 0x8003 2
$ openpet -c 5 0x8003 0x00000001
```

7.1.6 Command ID: 0x0006

Description: Reads the System Acquisition Mode Settings register from firmware.

Payload: None e.g. 0 or any value.

Examples:

Reads the System Acquisition Mode Settings register from firmware from slot 3. The payload of the reply is set to the settings previously specified, e.g., scope mode settings:

```
$ openpet -c 6 3 0xDEADFEED

2015-09-02 11:10:00,528 INFO [S] 0x0006 0x0003 0xDEADFEED
2015-09-02 11:10:00,732 INFO [R] 0x8006 0x0003 0x02000100
```

7.1.7 Command ID: 0x0007

Description: Writes the System Acquisition Mode Action register in firmware.

Payload:

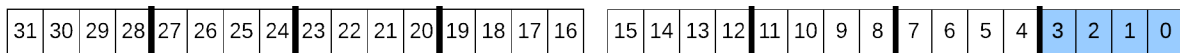


Fig. 7.7: System Acquisition Mode Action payload

```
Starting from the least significant bit (LSB)
(3:0) Action
(31:4) not used
```

Available actions:

Reset	0x0
Stop	0x1
Run	0x2

Examples:

Reset the System Acquisition Mode and Settings (payload = 0x0):

```
$ openpet -c 7 0x8003 0x00000000
```

Stop the current system acquisition (payload = 0x1):

```
$ openpet -c 7 0x8003 0x00000001
```

First, broadcast to all nodes to set System Acquisition Mode to scope mode. Then use data format = 0, 16 samples, 0 samples before trigger, and a trigger window to 2. Finally, run the acquire data command (payload = 0x2):

```
$ openpet -c 3 0x8003 1
$ openpet -c 5 0x8003 0x02000100
$ openpet -c 7 0x8003 0x00000002
```

7.1.8 Command ID: 0x0008

Description: Reads the System Acquisition Mode Action register from firmware.

Payload: None e.g. 0 or any value.

Examples:

Reads the System Acquisition Mode Action register from firmware from slot 3. The payload of the reply is set to the action previously specified, e.g., reset = 0:

```
$ openpet -c 8 3 0xDEADFEED
2015-09-02 11:10:00,528 INFO [S] 0x0008 0x0003 0xDEADFEED
2015-09-02 11:10:00,732 INFO [R] 0x8008 0x0003 0x00000000
```

If action previously set to terminate the current action, e.g., stop = 1:

```
$ openpet -c 8 3 0xDEADFEED
2015-09-02 11:10:00,528 INFO [S] 0x0008 0x0003 0xDEADFEED
2015-09-02 11:10:00,732 INFO [R] 0x8008 0x0003 0x00000001
```

If action previously set to run the system acquisition, e.g., run = 2:

```
$ openpet -c 8 3 0xDEADFEED
2015-09-02 11:10:00,528 INFO [S] 0x0008 0x0003 0xDEADFEED
2015-09-02 11:10:00,732 INFO [R] 0x8008 0x0003 0x00000002
```

7.1.9 Command ID: 0x0009

Description: Writes a mask to suppress triggers in firmware.

Payload:



Fig. 7.8: Trigger mask payload

```
Starting from the least significant bit (LSB)
(31:0) 1 bit per channel. Set bit to 1 to enable trigger on that channel.
```

Example:

Set trigger masks for channels 0, 5, and 8 in detector board 5:

```
$ openpet -c 9 5 0x00000121
```

7.1.10 Command ID: 0x000A

Description: Reads the trigger mask from firmware.

Payload: None e.g. 0 or any value.

Example:

Reads the trigger mask previously set in detector board 5:

```
$ openpet -c 10 5 0xDEADFEED

2015-09-02 11:10:00,528 INFO [S] 0x000A 0x0005 0xDEADFEED
2015-09-02 11:10:00,732 INFO [R] 0x800A 0x0005 0x00000121
```

7.1.11 Command ID: 0x000B

Description: Writes to external SRAM device. Auto-increments address.

Payload:

```
Starting from the least significant bit (LSB)
(31:0) Value to write to SRAM
```

Caveats: The Mode Action has to be in Reset for this command to work correctly.

Example:

Write some value to external SRAM in detector board 1:

```
$ openpet -c 11 1 0x12345678

2015-10-12 11:02:33,115 INFO [S] 0x000B 0x0001 0x12345678
2015-10-12 11:02:33,355 INFO [R] 0x800B 0x0001 0x12345678
```

Alternative: -sw option Write to SRAM in detector board 1 from file sramtest.bin starting from the top (offset = 0):

```
$ openpet -sw sramtest.bin 1 0

2015-10-12 11:05:22,956 INFO Writing content to SRAM.
2015-10-12 11:05:28,385 INFO Done SRAM writing.
```

7.1.12 Command ID: 0x000C

Description: Reads from external SRAM device. Auto-increments address.

Payload:

Starting from the least significant bit (LSB)
(31:0) SRAM address

Caveats: The Mode Action has to be in Reset for this command to work correctly.

Example:

Read data starting from SRAM address 0 in detector board 1:

```
$ openpet -c 12 1 0

2015-10-12 11:02:41,956 INFO [S] 0x000C 0x0001 0x00000000
2015-10-12 11:02:42,165 INFO [R] 0x800C 0x0001 0x12345678
```

Alternative: -sr option Read 100 values from SRAM to file sramtest.bin from detector board 1 starting from the top (offset = 0):

```
$ openpet -sr sramtest.bin 1 100 0

2015-10-12 11:03:11,296 INFO Reading SRAM content.
2015-10-12 11:03:15,635 INFO Done Reading SRAM Content.
```

7.1.13 Command ID: 0x000D

Description: Clears SRAM content. Sets to zero.

Payload: None e.g. 0 or any value.

Example:

Clears SRAM content on detector board 1:

```
$ openpet -c 13 1 0xDEADFEED

2015-10-09 11:10:00,528 INFO [S] 0x000D 0x0001 0xDEADFEED
2015-10-09 11:10:00,732 INFO [R] 0x800D 0x0001 0x00000000
```

7.1.14 Command ID: 0x000F

Description: System wide reset. It resets the configurations of firmware, software, and peripheral hardware to default values:

****Payload**:** None e.g. 0 or any value.

Example:

Broadcast a reset to all nodes and get a reply from 0x1:

```
$ openpet -c 0xF 0x8001 0

2015-10-09 11:10:00,511 INFO [S] 0x000F 0x8001 0x00000000
2015-10-09 11:10:00,733 INFO [R] 0x800F 0x8001 0x00000000
```

7.1.15 Command ID: 0x0101

Description: Sets the TDC control register.

Payload:

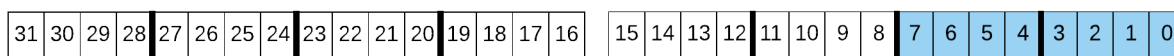


Fig. 7.9: Payload for TDC control register

```
Starting from the least significant bit (LSB)
(7:0)    TDC command
(31:8)   Not used
```

Available TDC commands:

Reset	0x80
Calibrate	0x02
Run	0x04

Examples:

Set the TDC control register to reset in detector board 5:

```
$ openpet -c 0x0101 5 0x00000080
```

Set the TDC control register to calibrate in detector board 5:

```
$ openpet -c 0x0101 5 2
```

Set the TDC control register to run in detector board 5:

```
$ openpet -c 0x0101 5 4
```

7.1.16 Command ID: 0x0102

Description: Gets the TDC control register.

Payload: None e.g. 0 or any value.

Example:

Reads the current TDC control register from detector board 5, e.g., run = 1:

```
$ openpet -c 0x0102 5 0xDEADFEED

2015-10-09 11:10:00,528 INFO [S] 0x0102 0x0005 0xDEADFEED
2015-10-09 11:10:00,732 INFO [R] 0x8102 0x0005 0x00000001
```

7.1.17 Command ID: 0x0103

Description: Resets ADC configuration. Commands Detector Board(s) to set ADC registers to OpenPET default values.

Payload: None e.g. 0 or any value.

Example:

Resets ADC registers on detector board 3 to default OpenPET values:

```
$ openpet -c 0x0103 3 0xDEADFEED

2015-10-09 11:10:00,528 INFO [S] 0x0103 0x0003 0xDEADFEED
2015-10-09 11:10:00,732 INFO [R] 0x8103 0x0003 0x00000000
```

7.1.18 Command ID: 0x0104

Description: Writes ADC register. See [ADS5282 datasheet](#) for valid register maps.

Payload:

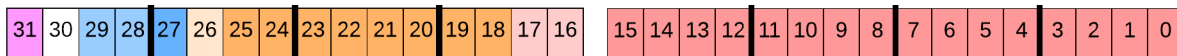


Fig. 7.10: Payload to write ADC register

```
Starting from the least significant bit (LSB)
(15:0)  ADC data (see ADS5282 datasheet, page 17)
(17:16) Reserved
(25:18) ADC register address (see ADS5282 datasheet, page 17)
(26)    Reserved
(27)    ADC chip address (0 for channel 0-7; 1 for channel 8-15)
(29:28) Reserved
(30)    Not used
(31)    Broadcast flag, i.e., run on all ADC chips
```

Note:

The ADC address and ADC data for the corresponding DB channel when setting the ADC gain. Each gain is set using 4 bits ranging from 0-12 dB. The gain has to be set on four channels at a time (i.e., 16-bit ADC data).

ADC Chip Address	ADC Address	ADC Data (MSB to LSB)
0x0	0x2A	Channel 3 to 0
0x0	0x2B	Channel 4 to 7
0x1	0x2A	Channel 11 to 8
0x1	0x2B	Channel 12 to 15

Examples:

Sets the ADC gain to 6 dB for channels 4 to 7 on the Detector Board in slot 3 without broadcasting:

```
$ openpet -c 0x0104 0x0003 0x00AC6666
```

Payload breakdown (LSB to MSB):

- 0x00AC6666 (=0 0 00 0 0 00101011 00 0110011001100110)
- Bits 15-0: 0110011001100110 (see [ADS5282 datasheet](#), page 17)
- Bits 17-16: Reserved
- Bits 25-18: 0x2B indicates for channel 4 to 7
- Bits 26: Reserved
- Bits 27: 0 indicates for channel 4 to 7
- Bits 29-28: Reserved
- Bits 30: Not used
- Bits 31: 0 indicates not a broadcast command

Sets the ADC gain to 8 dB for channels 8 to 11 on the Detector Board in slot 5 with broadcasting:

```
$ openpet -c 0x0104 0x0005 0x88A88888
```

Payload breakdown (LSB to MSB):

- 0x88A88888 (=1 0 00 1 0 00101010 00 1000100010001000)
- Bits 15-0: 1000100010001000 (see [ADS5282 datasheet](#), page 17)
- Bits 17-16: Reserved
- Bits 25-18: 0x2A indicates for channel 8 to 11
- Bits 26: Reserved
- Bits 27: 1 indicates for channel 8 to 11
- Bits 29-28: Reserved
- Bits 30: Not used
- Bits 31: 1 indicates broadcast command

7.1.19 Command ID: 0x0105

Description: Resets DAC configuration. Commands Detector Board(s) to set DAC registers to OpenPET default values.

Payload: None e.g. 0 or any value.

Example:

Resets DAC registers on detector board 3 to default OpenPET values:

```
$ openpet -c 0x0105 3 0xDEADFEED

2015-10-09 11:10:00,528 INFO [S] 0x0105 0x0003 0xDEADFEED
2015-10-09 11:10:00,732 INFO [R] 0x8105 0x0003 0x00000000
```

7.1.20 Command ID: 0x0106

Description: Writes DAC register. See [DAC LTC2634 datasheet](#) for valid register maps.

Payload:



Fig. 7.11: Payload to write DAC register

```
Starting from the least significant bit (LSB)
(9:0)   DAC data (see LTC2634 datasheet, page 20)
(12:10) Reserved
(16:13) DAC address (see LTC2634 datasheet, page 20)
(20:17) DAC command (see LTC2634 datasheet, page 20)
(22:21) Reserved
(26:23) DAC chip address
(28:27) DAC type (00=energy, 01=timing, 10=reserved, 11=all)
(30:29) Not used
(31)   Broadcast flag, i.e., run on all DAC chips for a given type
```

Note:

DAC Command=0x3 to set DAC voltage (10-bit data)

Type	Bits 28-27	DAC Full Scale
Energy	00	4.096 V
Timing	01	2.500 V
Reserved	10	2.500 V

The DAC address and DAC chip select for the corresponding DB channel when setting the energy and timing DAC. Because a chip has four DAC, a single DAC voltage can be set to four channels with one command.

Channel	DAC Address	DAC Chip Address
0	0x0	0x0
1	0x1	0x0
2	0x2	0x0
3	0x3	0x0
4	0x0	0x1
5	0x1	0x1
6	0x2	0x1
7	0x3	0x1
8	0x0	0x2
9	0x1	0x2
10	0x2	0x2
11	0x3	0x2
12	0x0	0x3
13	0x1	0x3
14	0x2	0x3
15	0x3	0x3
0-3	0xF	0x0
4-7	0xF	0x1
8-11	0xF	0x2
12-15	0xF	0x3

- The timing DAC is usually set to a low threshold to obtain the best timing.
- The energy DAC determines whether readout is initiated, and so is usually set to a higher threshold to reduce noise triggers (see *16-Channel Detector Board* (page 48)).

Examples:

Sets DAC energy threshold to +1.150V for channel 4 on the Detector Board in slot 3 without broadcasting:

```
$ openpet -c 0x0106 0x0003 0x00860120
```

Payload breakdown (LSB to MSB):

- 0x000860120 (=0 00 00 0001 00 0011 0000 000 0100100000)
- Bits 9-0: 0100100000 (1024*1.15/4.096)
- Bits 12-10: Reserved
- Bits 16-13: 0x0 (channel 4 in chip select 0x1)
- Bits 20-17: 0x3 (see [DAC LTC2634 datasheet](#), page 20)
- Bits 22-21: Reserved
- Bits 26-23: 0x1 indicates this is the chip where channel 4 resides
- Bits 28-27: 00 indicates that the DAC type is energy
- Bits 30-29: Not used
- Bits 31: 0 indicates not a broadcast command

Sets DAC timing threshold to +0.5V for channel 12 on the Detector Board in slot 5 with broadcasting:

```
$ openpet -c 0x0106 0x0005 0x898600CD
```

Payload breakdown (LSB to MSB):

- 0x898600CD (=1 00 01 0011 00 0011 0000 000 0011001101)
- Bits 9-0: 0011001101 ($1024 * 0.5 / 2.5$)
- Bits 12-10: Reserved
- Bits 16-13: 0x0 (channel 12 in chip select 0x3)
- Bits 20-17: 0x3 (see [DAC LTC2634 datasheet](#), page 20)
- Bits 22-21: Reserved
- Bits 26-23: 0x3 indicates this is the chip where channel 12 resides
- Bits 28-27: 01 indicates that the DAC type is timing
- Bits 30-29: Not used
- Bits 31: 1 indicates a broadcast command

7.1.21 Command ID: 0x0107

Description: Writes Sawtooth pulse(s). Commands DAC(s) to send sawtooth pulse(s) for a given duration of time.

Payload:

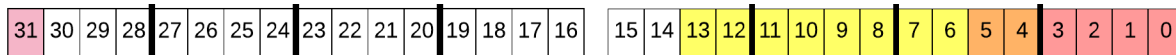


Fig. 7.12: Payload to write sawtooth pulse

```
Starting from the least significant bit (LSB)
(3:0)   DAC chip address
(5:4)   DAC type
(13:6)  Number of sawtooth pulses to send
(30:14) Not used
(31)    Broadcast flag, i.e., run on all DAC chips for a given type
```

Note: The time it takes to execute and finish this command depends on the number of sawtooth pulses you request. It is highly advised to send this command as an asynchronous command (non-blocking) by setting the MSB on CMD ID to '1'.

Example:

Command DAC to send 100 sawtooth pulses to detector board 3. DAC type is 00 which is energy, and chip address is 0x2. Broadcast flag is set to 1, and the CMD ID MSB is set to 1 for asynchronous command:

```
$ openpet -c 0x8107 0x0003 0x80001902
```

7.1.22 Command ID: 0x0108

Description: Sets a firmware threshold level to trigger on.

Payload:



Fig. 7.13: Payload to set firmware trigger threshold

```

Starting from the least significant bit (LSB)
(11:0)   Threshold value
(15:12)  Reserved
(17:16)  Mode (00=off, 01=on)
(31:18)  Not used

```

Note: The minimum and maximum threshold values correspond to the minimum and maximum of the ADC signal which is 2Vpp. To set at the 0V DC, set the MSB of the threshold value to 1.

Example:

Sets the firmware threshold level to 0V and mode 'on' for detector board 5:

```
$ openpet -c 0x0108 0x0005 0x00010800
```

7.1.23 Command ID: 0x0109

Description: Gets the firmware trigger threshold.

Payload: None e.g. 0 or any value.

Example:

Retrieves the firmware trigger threshold previously set on detector board 5:

```

$ openpet -c 0x0109 0x0005 0xDEADFEED

2015-10-09 11:10:00,528 INFO [S] 0x0109 0x0005 0xDEADFEED
2015-10-09 11:10:00,732 INFO [R] 0x8109 0x0005 0x00010800

```

OpenPET Control and Analysis Tools (OpenPET CAT)

OpenPET Control and Analysis Tools (OpenPET CAT) are data acquisition and analysis software for the OpenPET electronics based on the ROOT framework. OpenPET CAT utilizes the object-oriented design in providing basic utilities, control and analysis tools for the OpenPET electronics. Using OpenPET CAT requires installing the ROOT package (<http://root.cern.ch>). Using OpenPET CAT is optional; users can run the executables “openpet” (see Commands section) to configure the system and to acquire data directly.

8.1 Installing ROOT

Go to <http://root.cern.ch/drupal/content/downloading-root> for instructions on downloading and installing ROOT. Although ROOT is supported on many platforms, OpenPET only supports the Windows version of ROOT using the Microsoft Visual C++ compiler. Current OpenPET CAT supports ROOT version 5.34/30 and Microsoft Visual C++ 2010. If you plan to develop and compile OpenPET CAT codes, you also need to install Microsoft Visual C++ 2010 Express, which can be downloaded at no cost from <http://www.visualstudio.com/downloads/download-visual-studio-vs>, and cygwin.

8.2 Using OpenPET CAT

8.2.1 Configuration Files

Defining an OpenPET system starts with a system configuration file as shown in Fig. 8.1.

The system configuration file defines:

- **Command Engine Type:** “0” for USB and “1” for Ethernet.
- **Command Engine Id:** “0”
- **Acquisition Engine Type:** “0” for USB and “1” for Ethernet.
- **Acquisition Engine Id:** “0”
- **Configuration Files Directory:** The directory path to the remaining configuration files.
- **MB Setup Files Prefix:** The file prefix to the Multiplexer Board configuration files.
- **MB Configuration:** The multiplexer board address (0 to 7).

Note: Any comments can be inserted between the “***end ...” and the next “***...” tags.

```
# This file gives the hardware configuration parameters for
# the specified scanner.

***Command Engine Type***
# 0=usb, 1=ethernet
0
***end Command Engine Type***

***Command Engine Id***
0
***end Command Engine Id***

***Data Engine Type***
# 0=usb, 1=ethernet
0
***end Data Engine Type***

***Data Engine Id***
0
***end Data Engine Id***

***Configuration Files Directory***
C:/Users/seng/Documents/OpenPET/repocat/examples
***end Configuration Files Directory***

***MB Setup Files Prefix***
MB
***end MB Setup Files Prefix***

***MB Configuration***
# MB Address
0
***end MB Configuration***
```

Fig. 8.1: Example system configuration for a small system.

Following the system configuration files, there are three more configuration files: 1) Multiplexer Board configuration file; 2) Detector Unit configuration file; and 3) Detector Board configuration file. For example, in Fig. 8.1, there is one multiplexer board with address 0 (note that in the small system, there are no physical MB board, so a dummy MB configuration file with address 0 is used). The location and prefix of this file is specified in “***Configuration Files Directory***” and “***MB Setup Files Prefix***” respectively. In this example, a file MB0.txt, as shown in Fig. 8.2, would reside in the directory C:\Documents and Settings\seng\My Documents\OpenPET\software\small_system. MB0.txt shows that there is a Detector Unit with address 0, it is connected to MB0, and the configuration file for this Detector Unit is MB0_DU0.txt as shown in Fig. 8.3. MB0_DU0.txt shows that the Detector Board type is the 16-Channel DB for each of the two DBs with addresses 0 and 2 in this Detector Unit. The configuration files for these Detector Boards are MB0_DU0_DB0.txt and MB0_DU0_DB2.txt. For example, MB0_DU0_DB0.txt is shown in Fig. 8.4, which include all the DAC threshold and ADC gain settings for the 16-Channel DB.

```
# This file gives the hardware configuration parameters for
# the Multiplexer Board.

***DU Setup Files Prefix***
MB0_DU
***DU Setup Files Prefix***

***DU Configuration***
# DU Address
0
***end DU Configuration***
```

Fig. 8.2: Example Multiplexer Board configuration file.

```
# This file gives the hardware configuration parameters for
# the Detector Unit.

***DB Setup Files Prefix***
MB0_DU0_DB
***end DB Setup Files Prefix***

***DB Configuration***
# Type Address (Type: 0=16-Ch DB)
0      0
0      2
***end DB Configuration***
```

Fig. 8.3: Example Detector Unit configuration file.

```
# This file gives the hardware configuration parameters for
# the Detector Board.

***Channel Enable***
# Channel Enable
0      1
1      1
2      1
3      1
4      1
5      1
6      1
7      1
8      1
9      1
10     1
11     1
12     1
13     1
14     1
15     1
***end Channel Enable***

***Timing Threshold***
# Channel DAC(V)
0      0.1
1      0.1
2      0.1
3      0.1
4      0.1
5      0.1
6      0.1
7      0.1
8      0.1
9      0.1
10     0.1
11     0.1
12     0.1
13     0.1
14     0.1
15     0.1
***end Timing Threshold***

***Energy Threshold***
# Channel DAC(V)
0      0.05
1      0.05
2      0.05
3      0.05
4      0.05
5      0.05
6      0.05
7      0.05
8      0.05
9      0.05
10     0.05
11     0.05
12     0.05
13     0.05
14     0.05
15     0.05
***end Energy Threshold***

***Reserved Voltage***
# DAC(V)
0.0
0.0
***end Reserved Voltage***

***ADC Baseline Voltage***
# DAC(V)
0.0
```

Fig. 8.4: Example Detector Board configuration file.

8.2.2 OpenPET CAT Library and Macros

A library of C++ classes and ROOT macros are implemented to configure the system, acquire data and analyze the data. The OpenPET CAT software package can be downloaded from the OpenPET website (<http://openpet.lbl.gov>) under the “Downloads” menu. In the ‘lib’ directory, there is a library file called libopenpetcat.dll, which has to be copied to the directory <installed ROOT directory>/bin (i.e., \$ROOTSYS/bin). In the ‘examples’ directory, there are macros and configuration files to help you get started. Three key macros are described below:

1. `acquireData.C(int acqTimeInSec, int initSystem=1)` This macro configures the system using configuration files and acquires a raw data file. Inputs (arguments to the function):

- `acqTimeInSec` - acquisition time in seconds

- `initSystem` - configures the system (by default set to 1)

Inputs (edit in the macro file):

- `acqMode` - see command ID 0x0003
- `dataFormat` - currently not implemented yet
- `numADCsample` - 0 to 512; see command ID 0x0005
- `numPreTrigSample` - 0 to 16; see command ID 0x0005
- `numTrigWindowSample` - 0 to 16; see command ID 0x0005
- `configFile` - system configuration file
- `filename` - raw data filename (the date will be prepended to this name)

Example 1: `root> .x acquireData.C(100)`

This macro initializes the system using the configuration files and acquires data for 100 s.

Example 2: `root> .x acquireData.C(100, 0)`

This macro acquires data for 100 s without initializing the system.

2. `displayWaveform.C` This macro displays the acquired waveforms event-by-event. Inputs (edit in the macro file):

- `numSkippedEvent` - number of events to be skipped
- `evtTimeWindow` - not supported yet
- `dataFilename` - filename of the raw data file

Example: `root> .x displayWaveform.C`

3. `analyzeRawData.C` This macro reads the waveforms event-by-event and integrates each waveform to calculate its energy. It also outputs a root file containing an ntuple with the energy and tdc values for every channel. This macro can only be used when only one 16-channel Detector Board is in the system. Inputs (edit in the macro file):

- `numSkippedEvent` - number of events to be skipped
- `evtTimeWindow` - not supported yet
- `startBaselineBin` - start bin for calculating the baseline of the waveform
- `endBaselineBin` - end bin for calculating the baseline of the waveform
- `startIntegratingBin` - start bin for calculating the energy of the waveform
- `endIntegratingBin` - end bin for calculating the energy of the waveform
- `dataFilename` - filename of the raw data file
- `rootFilename` - output ROOT filename

Example: `root> .x analyzeRawData.C`

8.3 OpenPET CAT Graphic User Interface

8.3.1 Introduction

The OpenPET CAT Graphic User Interface (GUI) is a user interface for the OpenPET CAT data acquisition framework. It was developed so that users may conveniently use the OpenPET CAT software to gather and analyze data from the OpenPET system. There are two user interfaces: the OpenPET Control and Analysis - Data Acquisition

(openpet_cat_acq.exe) and the OpenPET Control and Analysis - Root Analyzer (openpet_cat_ana.exe). The first one is used for configuring the system, gathering data, and generating the ROOT file. The second one is used for analyzing the ROOT file.

8.3.2 OpenPET Control and Analysis - Data Acquisition

Once the data acquisition application is launched, there are two screens that will appear. One will be the user interface titled “OpenPET Control and Analysis Tools - Data Acquisition,” and the other is a status window entitled “ROOT session.” Fig. 8.5 shows this initial display.

The Data Acquisition GUI has four functionalities associated with it. First, it can initialize the system based on the system configuration files discussed in *Configuration Files* (page 79). It also allows the user to acquire data in the Acquire Data tab and display the waveform events in the Display Waveform tab. In addition, the data file may be converted into a ROOT file in the Generate ROOT File tab. These four functionalities are discussed in detail below.

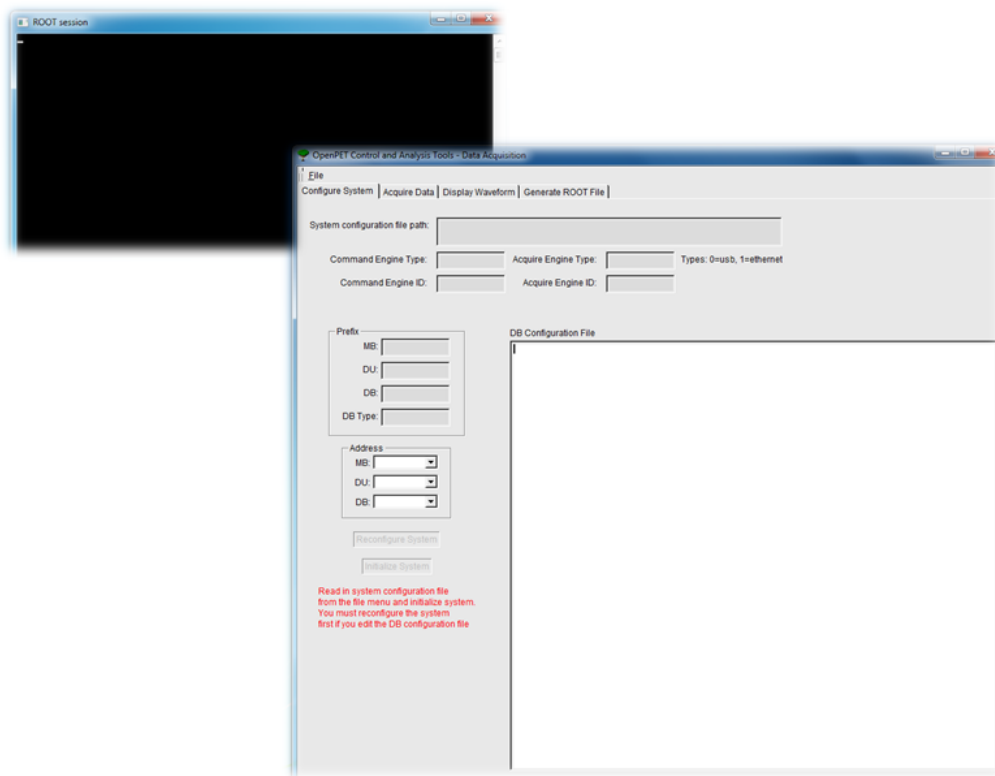


Fig. 8.5: Data Acquisition GUI Initial Display

Configure System

The initial screen of the user interface is the system configuration window (Fig. 8.6). This tab will display all hardware configuration data loaded from the system configuration file discussed in *Configuration Files* (page 79). The top section displays the engine settings and configuration file directory. The bottom section displays the prefixes and

address lists of Multiplexer Boards, Detector Units, and Detector Boards along with an editable text window for the Detector Board configuration file. There is also the Reconfigure System button and Initialize System button. The Reconfigure System button reconfigures the system should any values in the Detector Board configuration file be changed by the user. The Initialize System button initializes the system. Both buttons are disabled until a system configuration file is loaded.

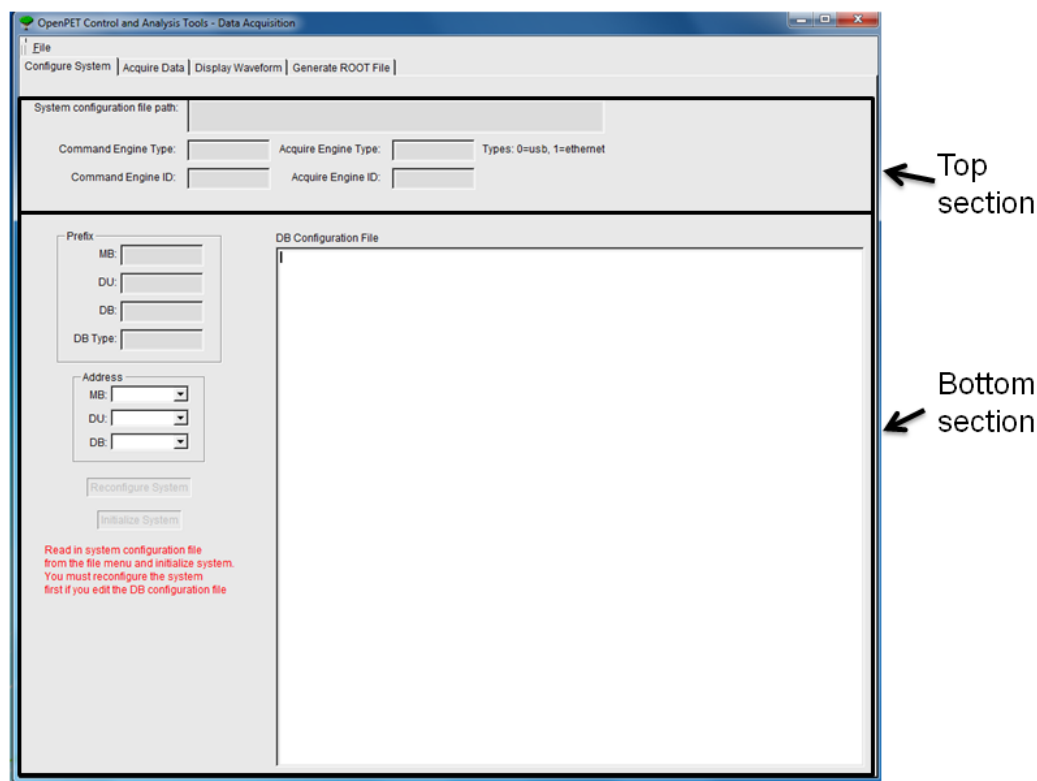


Fig. 8.6: Sections of the Configuration tab

To load a system configuration file, click on the File menu in the top left hand corner and select Read System Configuration File. This will open a dialog window allowing the user to select the desired system configuration file as shown in Fig. 8.7. Once the file is selected, the system will configure. The user may check the status of the configuration process by reading the ROOT session window. An example of a completed system configuration is shown in Fig. 8.8.

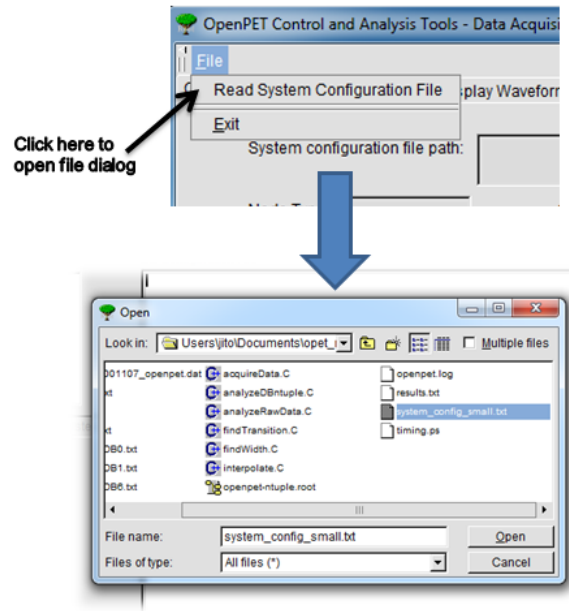
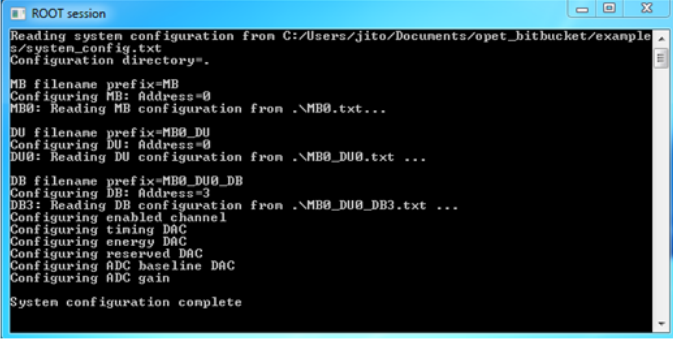


Fig. 8.7: Opening system configuration file

A screenshot of a ROOT session window titled "ROOT session". The window has a standard Windows-style title bar with minimize, maximize, and close buttons. The main content area is a black terminal window with white text. The text shows the process of reading system configuration from a file at "C:/Users/jito/Documents/opet_bitbucket/example/s/system_config.txt". It then proceeds to configure various components: MB (Multiplexer Board), DU (Detector Board), and DB (Detector Board). The configuration includes reading configuration files from the local directory (e.g., ".\\MB0.txt", ".\\MB0_DU0.txt", ".\\MB0_DU0_DB3.txt") and setting parameters like "enabled channel", "timing DAC", "energy DAC", "reserved DAC", "ADC baseline DAC", and "ADC gain". The session concludes with the message "System configuration complete".

```
ROOT session
Reading system configuration from C:/Users/jito/Documents/opet_bitbucket/example
s/system_config.txt
Configuration directory=.

MB filename prefix=MB
Configuring MB: Address=0
MB0: Reading MB configuration from .\\MB0.txt...

DU filename prefix=MB0_DU
Configuring DU: Address=0
DU0: Reading DU configuration from .\\MB0_DU0.txt ...

DB filename prefix=MB0_DU0_DB
Configuring DB: Address=3
DB3: Reading DB configuration from .\\MB0_DU0_DB3.txt ...
Configuring enabled channel
Configuring timing DAC
Configuring energy DAC
Configuring reserved DAC
Configuring ADC baseline DAC
Configuring ADC gain

System configuration complete
```

Fig. 8.8: ROOT Session Example

After the system has been configured, the top section will be filled with the appropriate values along with the configuration file path. Also, the configuration file of the first Detector Board of the first Multiplexer Board will appear. The MB, DU, and DB addresses shown specify the address of the configuration file being displayed. To change, select the desired MB, DU, and DB address combination from the dropdown lists to show that particular Detector Board's configuration file.

If the user is satisfied with the current configuration at this point, the user may initialize the system by clicking the Initialize System button. If not, the user can change values in a particular Detector Board's configuration file by selecting the correct MB address, DU address, and DB address to open the desired configuration file as shown in [Fig. 8.9](#). As the user makes changes, the configuration file is saved automatically. However, to have these changes take effect in the system, the system must be reconfigured by clicking the Reconfigure System button. Then the system may be initialized.

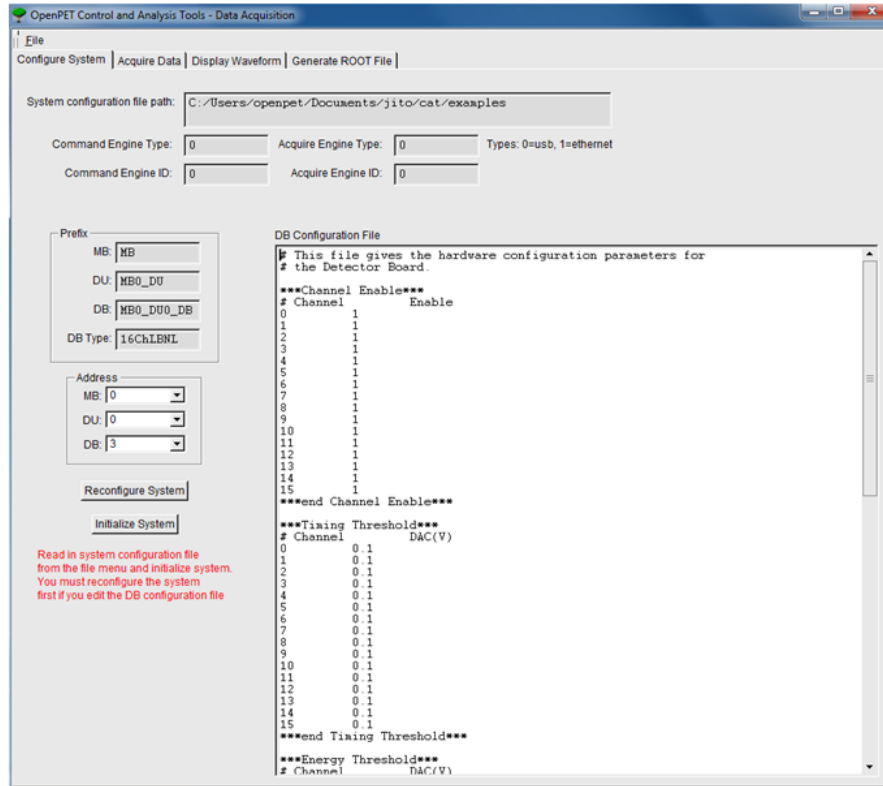


Fig. 8.9: Detector Board configuration file

Again, the user can observe the status of the system by reading the terminal window. The command and response IDs discussed in [Commands](#) (page 59) will be printed there. Therefore, the user can determine if the system was correctly initialized.

Acquire Data

Once the system has been initialized, the Acquire Data button will now activate so the user can acquire data. There are four parameters the user must set which are the following (shown in [Fig. 8.10](#)):

- Data Mode - see command ID 0x2201
- Data Format - see command ID 0x2203 for configuring oscilloscope mode
- Number of ADC samples - 1 to 241 (see command ID 0x2203)
- Acquisition time - acquisition time in seconds

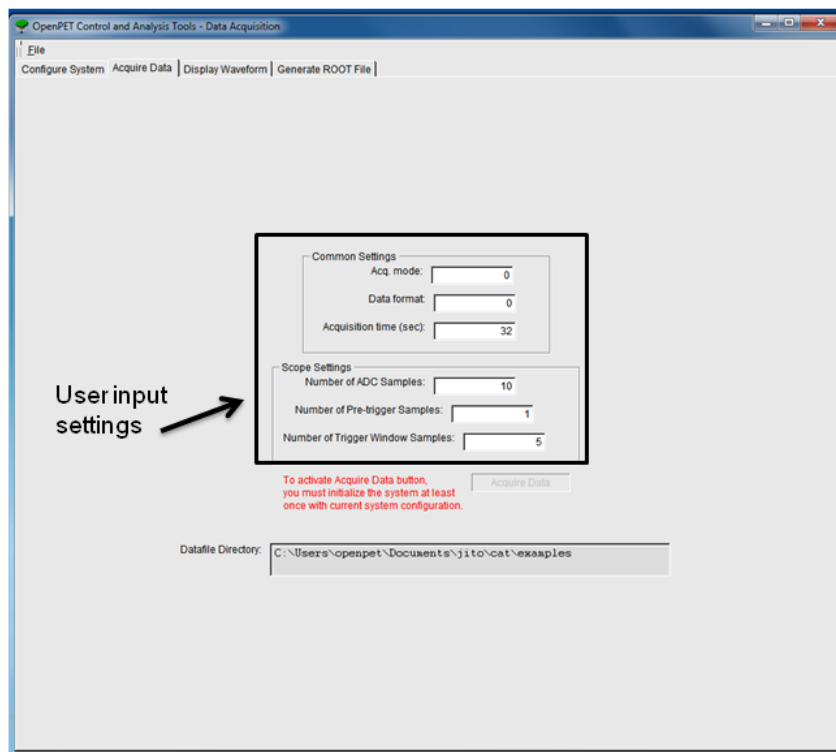


Fig. 8.10: Acquire data window

The previous values that were used will be set as default values.

The data file will be saved under the current working directory, which is displayed at the bottom of the window. The filename itself will have the format `yyyymmdd_hhmmss_openpet.dat` where the current date and time are used as the prefixes.

Example: `20140717_090251_openpet.dat` – Data file stored on July 17, 2014 at 9:02:51AM

Once the parameters are set, the user may click the Acquire Data button to begin retrieving data.

Display Waveform

Fig. 8.11 shows the Display Waveform tab. The functionality of this window is the same as the `displayWaveform` macro in *OpenPET CAT Library and Macros* (page 81). The user may load the desired data file to display the acquired waveforms event by event. The window is arranged vertically in four main sections. The top section is the user input settings. These settings are required to analyze the data file and are set to be the same values as in the Acquire Data tab. The next section is the data file section, which includes the button to load a data file and a field that will display the name of the current data file open. Next is the event section where the address and event number of the waveform currently displayed are shown, along with the button to cycle to the next event. The fourth section is the canvas where the waveforms will be displayed.

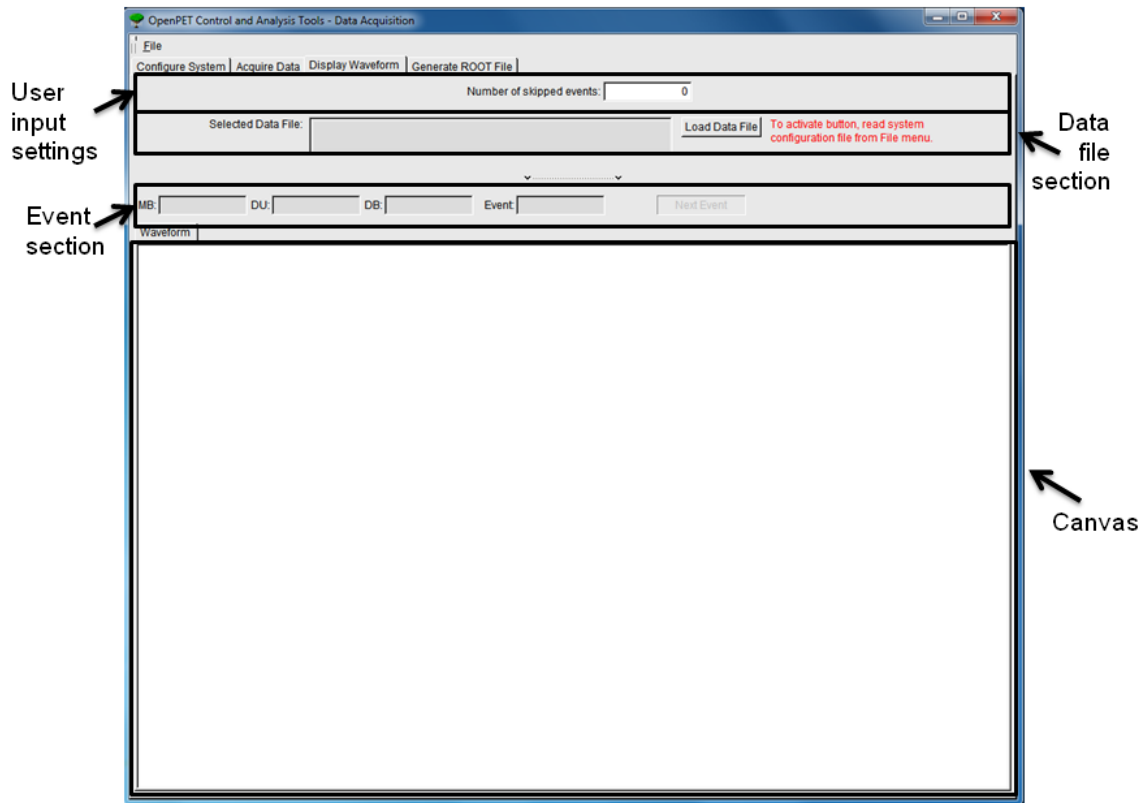


Fig. 8.11: Initial Display Waveform tab

To activate the Load Data File button, the user must first configure the system with the same configuration used to gather the data file. If a different system configuration file has already been loaded, the user can read in the correct system configuration file to match the configuration used with that data file. In either case, the steps to load the system configuration file are equivalent to the instructions in *OpenPET Control and Analysis - Data Acquisition* (page 83) and can be executed while in the Display Waveform tab. The user can load the data file by clicking on the Load Data File button. A file dialogue will appear. Select the desired file and open it. Once opened, the first event waveforms will appear in the canvas, and the address and event number will appear above it. The Next Event button will also activate at this time.

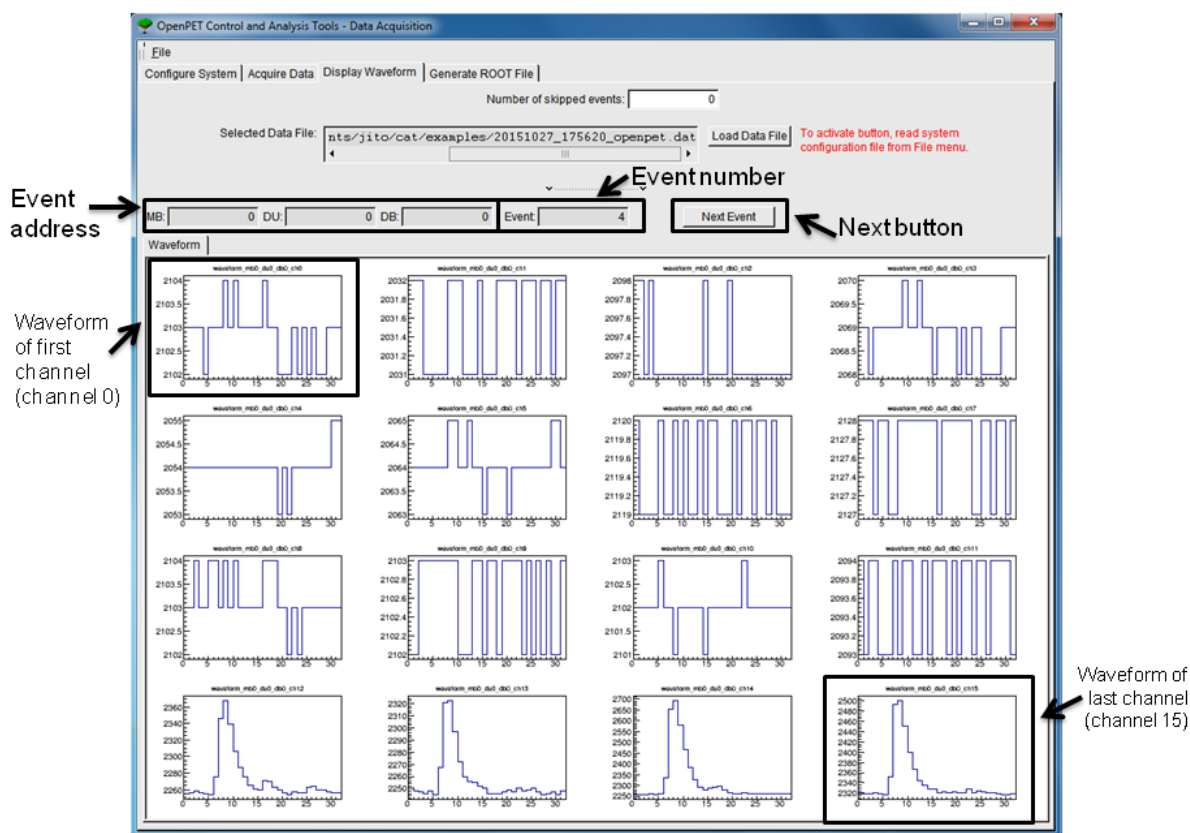


Fig. 8.12: Display Waveform

The waveforms are displayed by channel from left to right, starting in the top left corner. As shown in Fig. 8.12, the top left waveform is channel 0, the next one to the right is channel 1, and so on with the last channel on the bottom right corner. The address (MB, DU, DB) of the event currently displayed is shown above the canvas on the left along with the current event number. To the right of the address and event number is the Next Event button. Clicking on the Next Event button will display the next event waveforms.

In Fig. 8.12, the number of skipped events is set to the default zero. However, if the user does not want to start at the beginning of the file, the user may skip to a specific event number. For example, if the user wants to skip 10 events, the user would input ten as the desired number of skipped events and load the data file again. The opening waveforms would then display event ten as shown in Fig. 8.13.

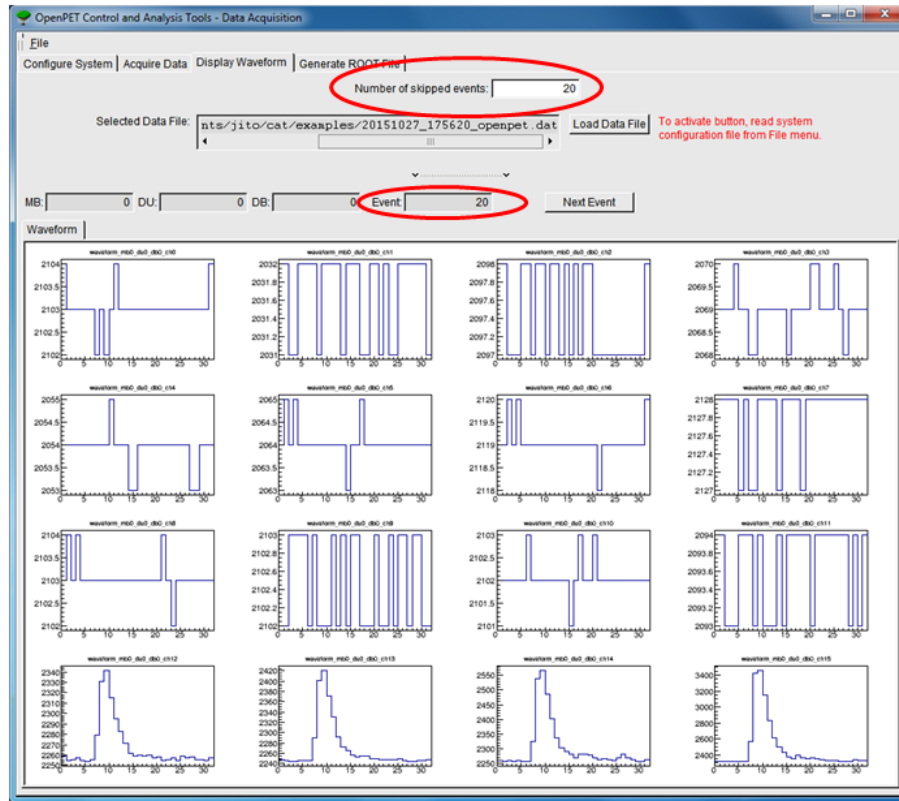


Fig. 8.13: Example highlighting number of skipped events

Generate ROOT File

The fourth tab allows the user to generate a ROOT file from the data file. This window is divided into three sections as shown in Fig. 8.14. The first section is the user input settings that specifies the data parameters. The second section is the data file section. It contains a text box that will display the file path of the loaded data file and the Load Data File button so the user can upload the data file. The final section contains the ROOT file parameters which are bin parameters required for calculating energy histograms that need to be set by the user. The previous values used are set as the default values.

To generate the ROOT file, the same system configuration file used in gathering the data must be currently loaded before loading the data file. Also, the user input settings must be the same as those used for gathering data. The default values are the same current values in the Acquire Data tab. Now, the user may click on the Load Data File button and a file dialogue will appear, allowing the user to select the desired file. Once the file is selected and the user is satisfied with the ROOT file parameters, he may click on the Generate ROOT File button. The button will not activate until a data file is loaded. Another file dialogue will appear, and the user can navigate to the directory where he would like to save the ROOT file. Then he must type in the filename **ending with the file extension “root”**. Click save and the ROOT file will begin generating. The sequence of steps is outlined in Fig. 8.15.

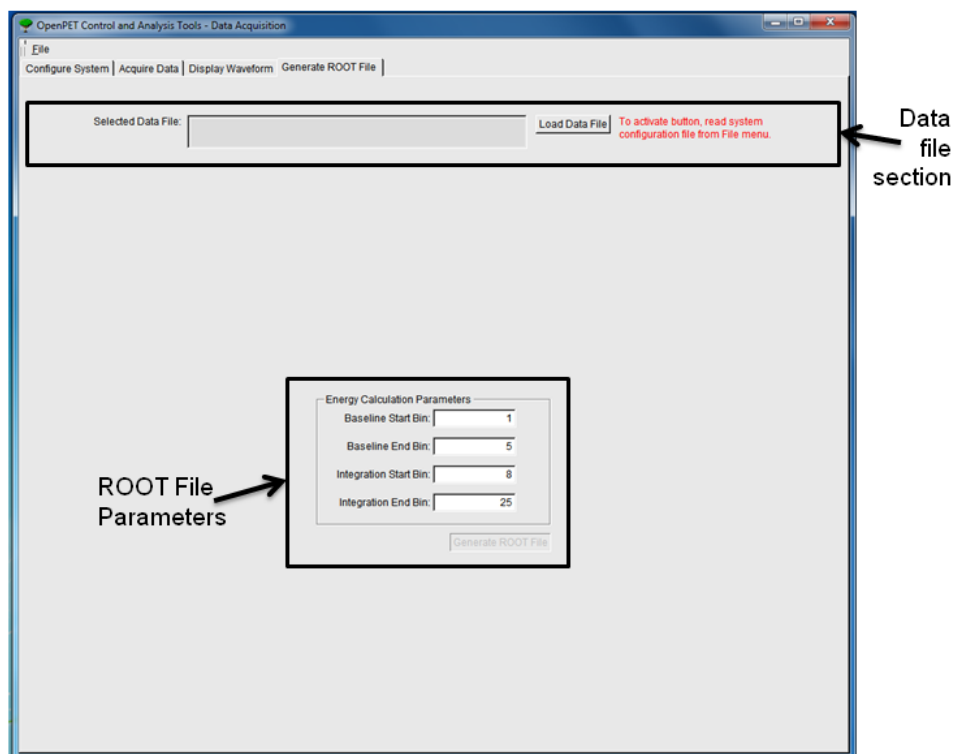


Fig. 8.14: Generate ROOT File tab

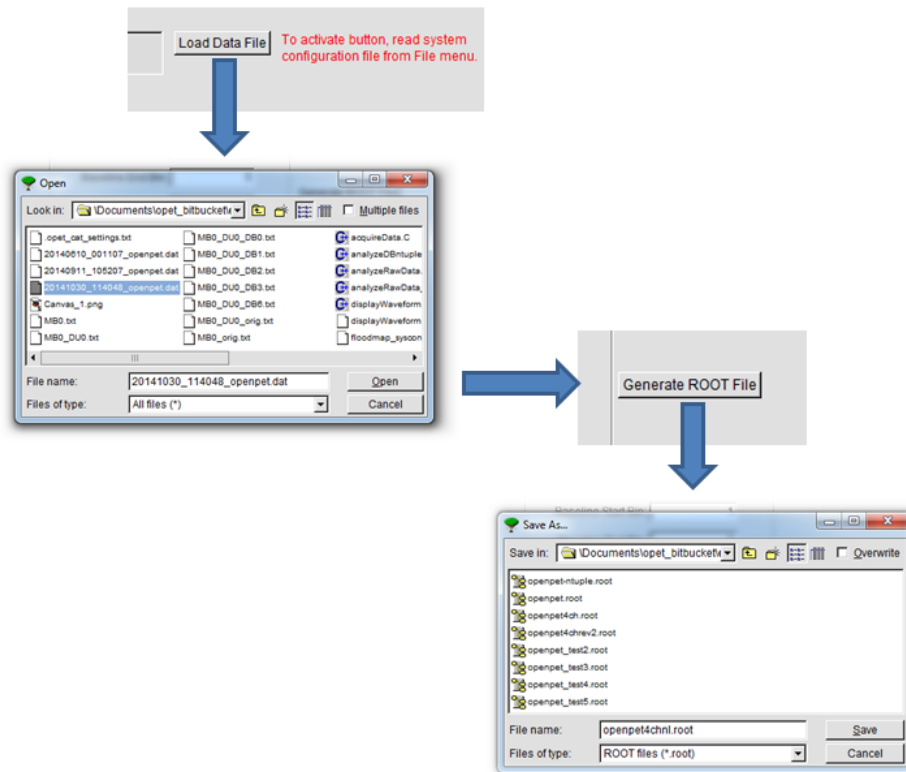
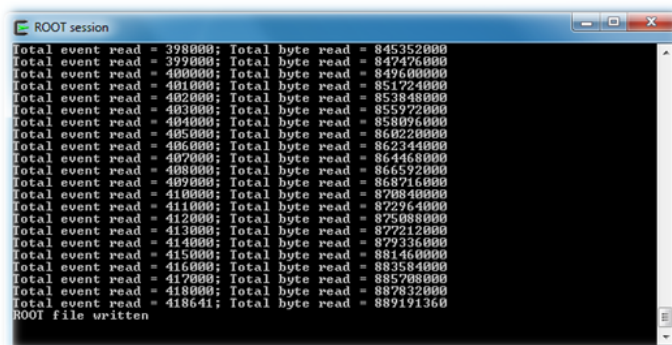


Fig. 8.15: Sequence of steps for saving a ROOT file

The status of the ROOT file may be checked by looking at the ROOT Session window. The window will print out every one thousand events read and the total bytes read. The ROOT file is completed when the statement "ROOT file written" is printed. An example is illustrated in Fig. 8.16.



```
ROOT session
Total event read = 398000; Total byte read = 845352000
Total event read = 399000; Total byte read = 847476000
Total event read = 400000; Total byte read = 849600000
Total event read = 401000; Total byte read = 851724000
Total event read = 402000; Total byte read = 853848000
Total event read = 403000; Total byte read = 855972000
Total event read = 404000; Total byte read = 858096000
Total event read = 405000; Total byte read = 860220000
Total event read = 406000; Total byte read = 862344000
Total event read = 407000; Total byte read = 864468000
Total event read = 408000; Total byte read = 866592000
Total event read = 409000; Total byte read = 868716000
Total event read = 410000; Total byte read = 870840000
Total event read = 411000; Total byte read = 872964000
Total event read = 412000; Total byte read = 875088000
Total event read = 413000; Total byte read = 877212000
Total event read = 414000; Total byte read = 879336000
Total event read = 415000; Total byte read = 881460000
Total event read = 416000; Total byte read = 883584000
Total event read = 417000; Total byte read = 885708000
Total event read = 418000; Total byte read = 887832000
Total event read = 418641; Total byte read = 889191360
ROOT file written
```

Fig. 8.16: Generating ROOT File Session

8.3.3 OpenPET Control and Analysis - ROOT Analyzer

Once the ROOT Analyzer application is launched, two screens will appear. One will be the user interface titled “OpenPET Control and Analysis Tools - ROOT Analyzer,” and the other will be a status window entitled “ROOT session.” Fig. 8.17 shows the initial startup of the ROOT Analyzer.

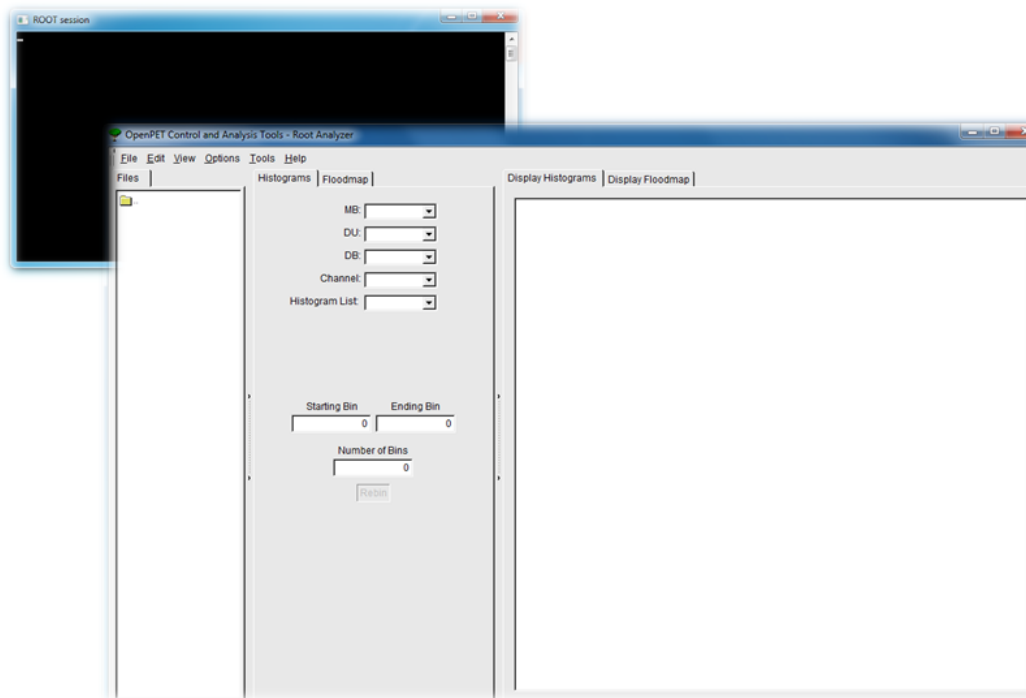


Fig. 8.17: ROOT Analyzer GUI Initial Display

The ROOT Analyzer GUI has two main functionalities. The first is displaying a variety of histograms such as detector board hitmap histograms and energy, baseline, and TDC histograms for specific channels in the Display Histograms tab. Secondly, for block sensors, the floodmap may be displayed in the Display Floodmap tab. These two functionalities are discussed in detail below.

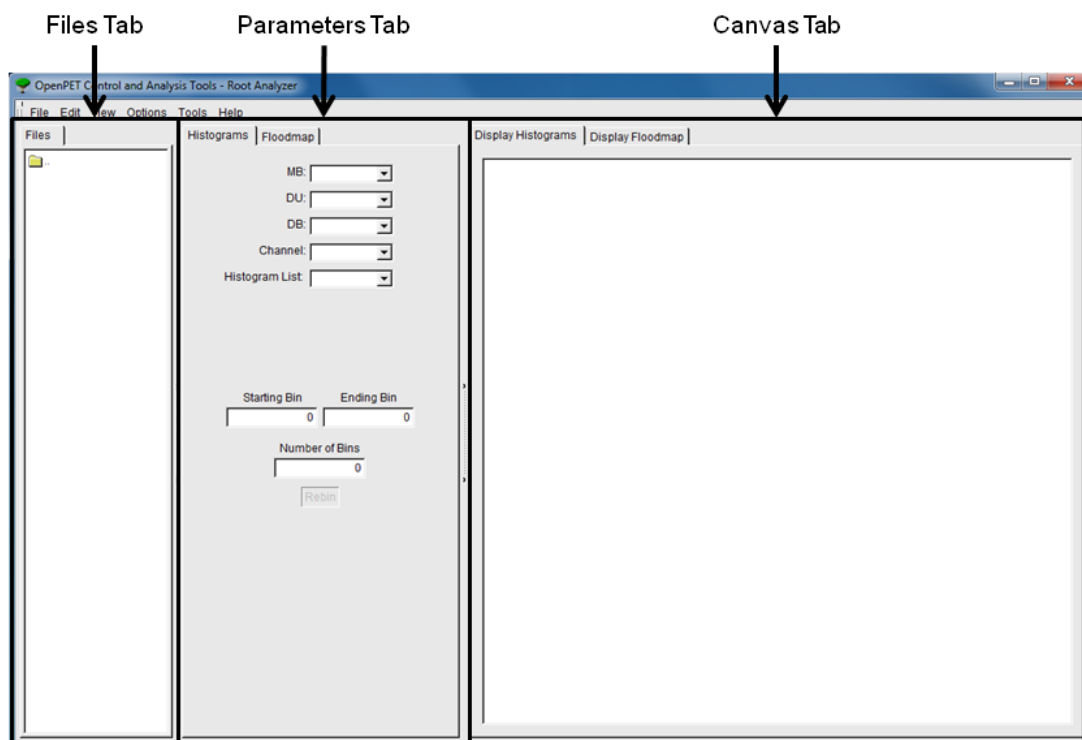


Fig. 8.18: ROOT Analyzer window layout

The window is laid out in three tab panels (Fig. 8.18). On the left is the Files tab which allows the user to open a ROOT file without opening a file dialogue. The middle panel is the Parameters Tab which has two tabs namely Histograms and Floodmap. On the right is the Canvas Tab which also has two tabs called Display Histograms and Display Floodmap. The middle Histograms tab contains parameters to display histograms on the Display Histograms canvas and the middle Floodmap tab contains parameters to display the floodmap on the Display Floodmap canvas. Each middle tab is linked to display its associated canvas and vice versa. For example, the user cannot select the middle Floodmap tab and be showing the Display Histograms canvas.

Before any analysis can be displayed, a ROOT file must be loaded first (Fig. 8.19). There are two different ways a user can open a ROOT file. The first method is by using the Files tab on the left side of the window. The user can navigate to the desired ROOT file by double clicking on folder icons to reach the desired directory. The folder titled “..” moves up one directory. The Files tab only displays directories or ROOT files. The second method to open a ROOT file is by using the File menu. To do so, the user must click on the File menu at the top left and select Read ROOT File. This will open a file dialogue displaying only ROOT files. The user can then select the desired ROOT file to open.

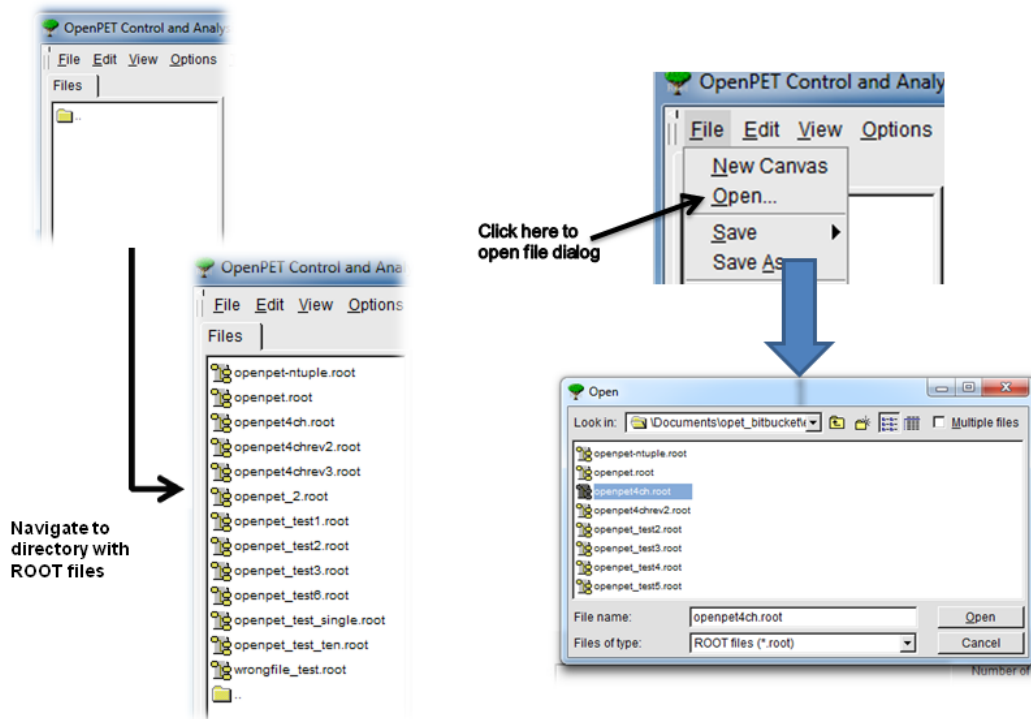


Fig. 8.19: Opening ROOT File - on the left is the first method mentioned and on the right is the second method.

Display Histograms

Once the ROOT file is opened, the hit histogram of the first Detector Board from the first Detector Unit of the first Multiplexor Board will be displayed as shown in Fig. 8.20. To see a different detector board, click the black arrows on the right side of the MB, DU, and DB address boxes to open a drop down list so that the desired MB, DU, and DB addresses may be selected. The first DB histograms for any given MB and DU combination is always displayed first until a different one is selected by the user. To display a different histogram, use the dropdown Histogram list to select the desired option as illustrated in Fig. 8.20.

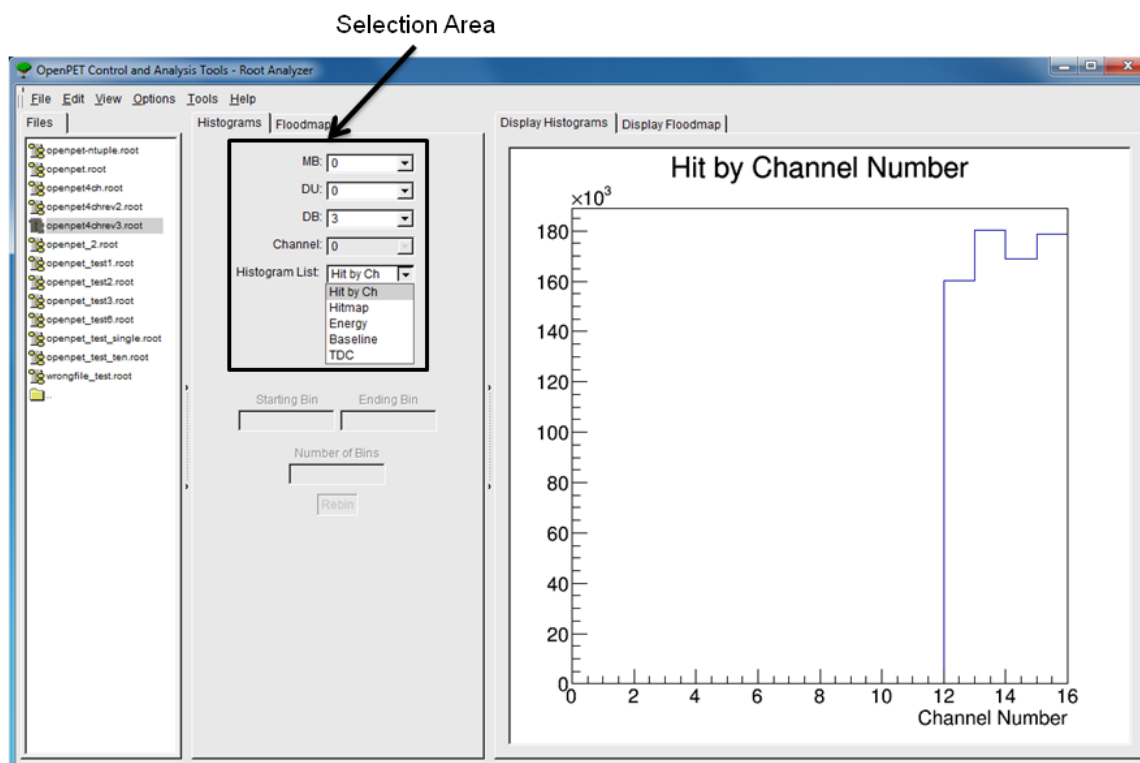


Fig. 8.20: Displaying First DB Hit

When viewing the energy, baseline, or TDC histogram, the user can change which channel to display by clicking on the black arrow next the channel number box. This will open a drop down menu from which the desired channel can be selected. The user can also adjust the binning of the histogram by clicking the Rebin button after changing the starting bin value, ending bin value, and the number of bins. An example is shown below in [Fig. 8.21](#).

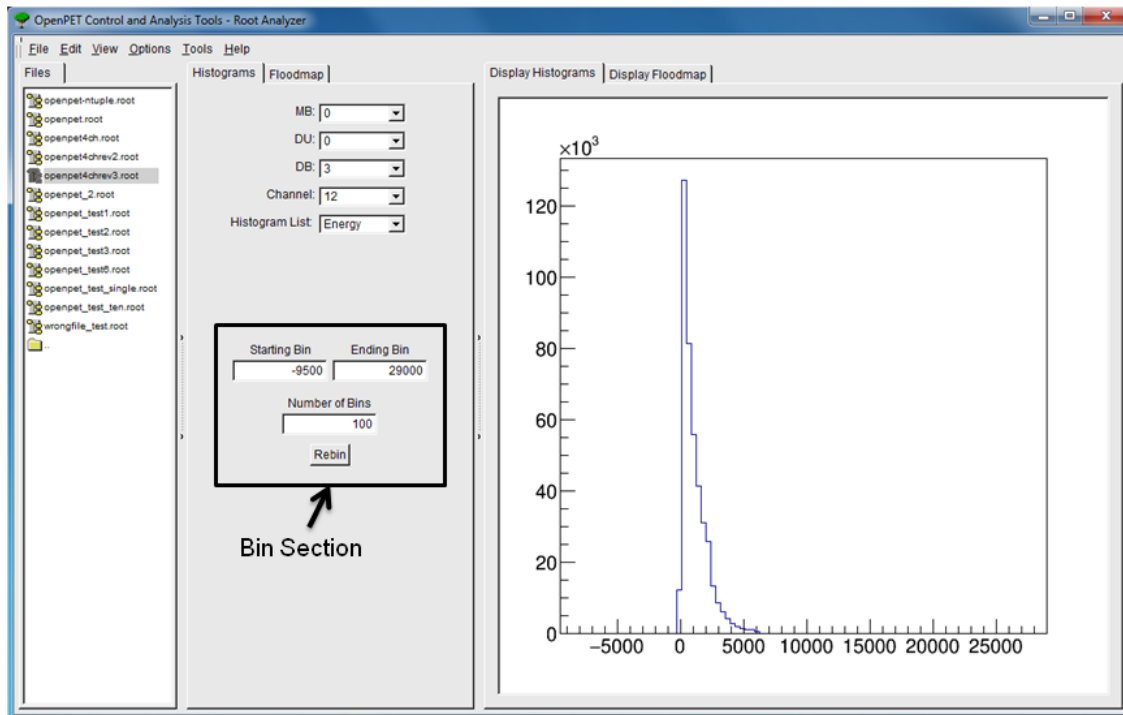


Fig. 8.21: Displaying DB Energy Histogram

Display Floodmap

For block detectors, the second pair of parameter and canvas tabs generates a floodmap for a user-defined combination of channels. The parameter Floodmap tab will be described top to bottom based on Fig. 8.22. At the top is the Address Section. This section contains the address of the current detector board along with the channels selected to generate the floodmap. There are four channel drop-down lists, one for each channel of the block detector. The channel names A through D correspond to the A through D values in the Equation Section. The Address Section also contains the Display Floodmap button. Next is the Bin Section which allows the user to rebin the histogram. The last section at the bottom is the Equation Section which shows the equations used to generate the floodmap.

To display a floodmap, the user must select the correct detector board if not already selected. After that, the user must determine which channels are the correct channels corresponding to variables A through D in the equations. Once the channels are selected, click on the Display Floodmap button. An example of a floodmap display is shown in Fig. 8.23. The floodmap can also be rebinned. The user can input the numbers for rebinning the X or Y values in the Bin Section and click the Rebin button.

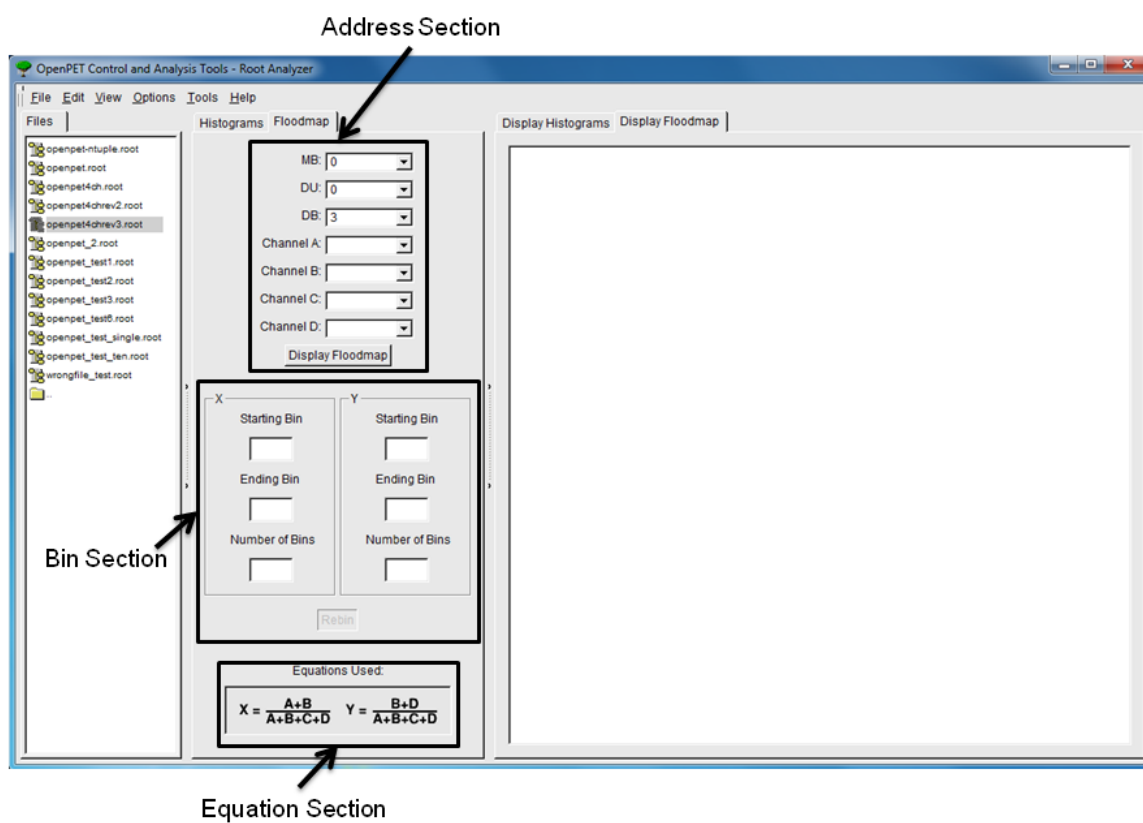


Fig. 8.22: Initial Floodmap Window

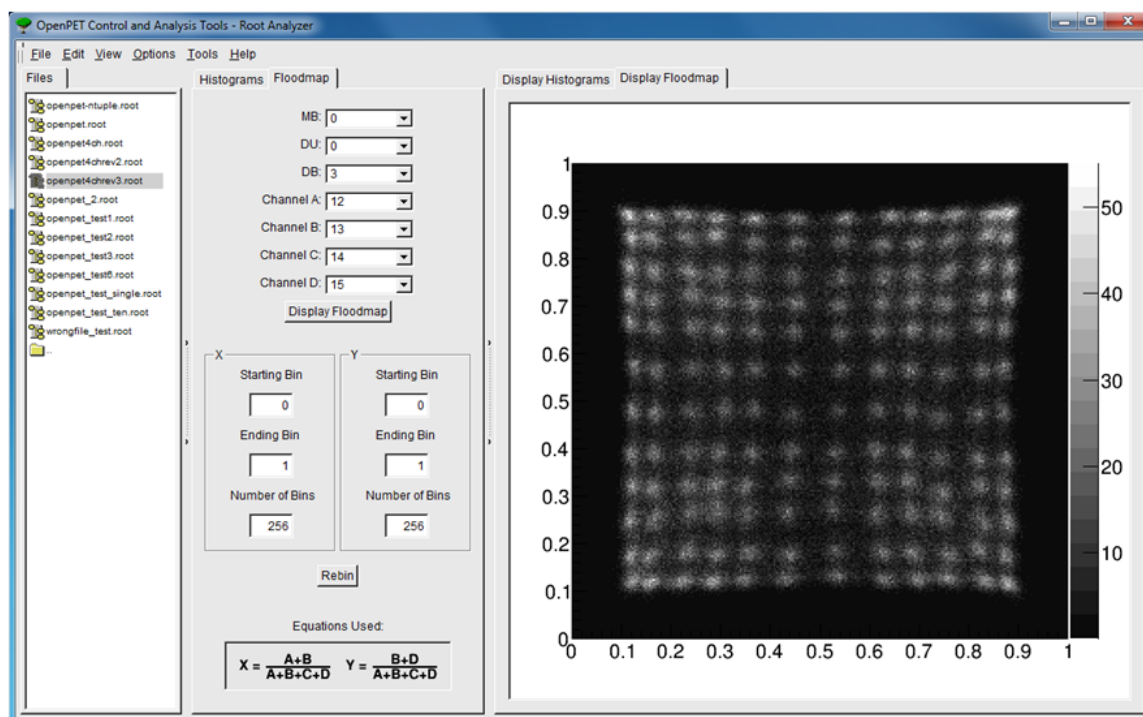


Fig. 8.23: Floodmap Example

Acknowledgements

Initial funding for the LBNL portion of this work was supported by the Director, Office of Science, Office of Biological and Environmental Research, Biological Systems Science Division of the U.S. Department of Energy under Contract No. DE-AC02-05CH11231. Subsequent work is supported by the National Institutes of Health of the US Department of Health and Human Services under grant R01 EB016104.

Reference to a company or product name does not imply approval or recommendation by the University of California or the U.S. Department of Energy to the exclusion of others that may be suitable.

10.1 Appendix 1: Default Values

10.1.1 Firmware Defaults (found in fw directory)

- Acquisition mode = 0x0 = idle
- Acquisition mode settings = 0x0
- Acquisition mode action = 0x0 = reset
- Trigger Mask = 0xFFFFFFFF = all channels enabled
- Debug ports are disabled
- SupportBoard Node Type = CDUC
- Main CLK Frequency = 80 MHz (Not programmable)
- Slice CLK Frequency = Main CLK Freq / 8 = 80/8 = 10 MHz = 100 ns
- QuickUSB Bus Width = 16 bits
- Number of QuickUSB packets (write cycles) to complete a single OpenPET command = 5
- QuickUSB output FIFO depth = 16 * 256. Should be more than 128. The larger the better.
- Total Number of Scope mode samples (DB fifo depth) = 256
- Number of Channels per ADC chip = 8
- Number of ADC chips = 2
- ADC resolution = 12
- TDC resolution = 19
- TDC core is enabled
- Scope mode core is enabled
- Singles mode core is enabled

10.1.2 Software Defaults (found in hostpc directory)

- QuickUSB Data bus width = 16 # QuickUSB Data bus width
- Number of QuickUSB packets (write cycles) to complete a single OpenPET command = 5

- Source Address = 0x4000
- QuickUSB Settings default values except for the following registers:
 - Register 1 = 0x0001, bus width = 16-bits
 - Register 2 = 0xC000, address bus is disabled not and incremented
 - Register 3 = 0x0002, IFLCK is disabled, polarity is normal, GPIF master mode
 - Register 5 = 0x8010, Allow high speed, CPU running at 48MHz, CLKOUT is disabled
- QuickUSB streaming call back timeout = 50 ms
- QuickUSB streaming buffers = 8
- QuickUSB streaming buffer size = 1 MB (this is the max)
- Default data output file name is strftime(“%Y%m%d-%H%M%S”) + “.openpetd”
- Data output file compression is disabled
- Number of command retries = 20
- Timeout between retries = 200 ms
- Acquisition time = 10 seconds
- Scope trigger window = 5
- Scope samples before trigger = 6
- Scope number of samples = 32
- Scope format = 0
- DAC threshold = 200 mv
- Process data before writing to file = true
- Acquisition mode = 0 = idle
- Acquisition mode settings = 0 = none

10.1.3 Embedded Software Defaults (found in sw directory)

- SupportBoard JTAG UART has enable_jtag_uart_ignore_fifo_full_error = true
- DetectorBoard JTAG UART is disabled
- Number of CLK ticks in order to complete a single software-firmware command = 4096
- SupportBoard SPI interface command timeout = 200 ms
- SupportBoard SPI interface retries before giving up on slave = 8

10.1.4 NIOS Peripheral Hardware Defaults (found in sw directory)

- ADC defaults are equal to datasheet defaults (i.e., gain=0 dB) except register 0x42 is set to 0x8005, i.e., a differential input clock is used on the ADC instead of singled ended.
- Timing DAC defaults are equal datasheet defaults except DAC level is set to 122.1 mV
- Energy DAC defaults are equal datasheet defaults except DAC level is set to 200 mV
- Ramp DAC defaults are equal datasheet defaults except DAC level is set to 0 mV

10.2 Appendix 2: Troubleshooting Diagnostics

Using the command sequence listed in *Example System Setup & Data Acquisition of a Small System in Oscilloscope Mode* (page 34) is a simple way to test if your OpenPET system has been properly set up to take data. However, if you are not properly acquiring data, further diagnosis will be necessary. This appendix outlines three basic tests for troubleshooting your OpenPET system. Data analysis can be performed with the provided HostPC Python Software, which is available on the [OpenPET website](#). You can also develop your own analysis software or use the OpenPET Control and Analysis optional software tool (see *OpenPET Control and Analysis Tools (OpenPET CAT)* (page 79)).

10.2.1 Test With Internal Trigger

The first step in troubleshooting is to discern if the basic support board and detector board digital command and data communication chain is functioning properly. This test does not require a signal input since the detector board generates the counter data input internally.

First, you need to set up the OpenPET hardware and install the firmware and software by following the instructions detailed in *Getting Started - Small System Oscilloscope Mode* (page 19). After that, **POWER CYCLE** your chassis. You can then configure the system and acquire test communication data. An example command sequence for a single detector board in slot 3 is shown below.

Step 1 - Ping the detector board in slot 3:

```
$ openpet -c 1 3 0
2015-09-01 12:30:00,529 INFO [S] 0x0001 0x0003 0x00000000
2015-09-01 12:30:00,733 INFO [R] 0x8001 0x0003 0x00000000
```

If you get an error at this step, check the following:

1. Are the detector board and support board programmed correctly?
2. Did you reboot your chassis?
3. Are the detector board slots correct? Remember, slots are numbered 0 to 7.

If the error persists after these things have been checked, the detector board may be physically damaged.

Step 2 - Configure ADC to send out fixed pattern output “PAT SYNC”:

```
$ openpet -c 0x0104 3 0x81140002
```

Step 3 - Change mode to Scope and configure its settings to 16 samples, 0 samples before trigger, and a trigger window of 2:

```
$ openpet -c 3 0x8003 1
$ openpet -c 5 0x8003 0x02000100
```

Step 4 - Bypass hardware trigger circuitry and use internal firmware trigger logic using a very small threshold:

```
$ openpet -c 0x108 0x8003 0x00010001
```

Step 5 - Acquire data for 5 seconds:

```
$ openpet -a 5
2015-10-23 11:41:34,368 INFO Starting qusb data acquisition...
2015-10-23 11:41:39,861 INFO 0.127s remaining.
2015-10-23 11:41:40,000 INFO QUSB rate is 41.048 MB/s
2015-10-23 11:41:40,000 INFO Stopping qusb data stream...
```

```
2015-10-23 11:41:40,000 INFO QUSB data stream has stopped.
2015-10-23 11:41:40,000 INFO Stopping qusb data queue [0]...
2015-10-23 11:41:40,000 INFO QUSB data queue has stopped.
```

A unique file name 20151023-114134.openpetd is created. The file size should be around 200 MB. Now use the plot.py script to view the data:

```
$ python plot.py 20151023-114134.openpetd
```

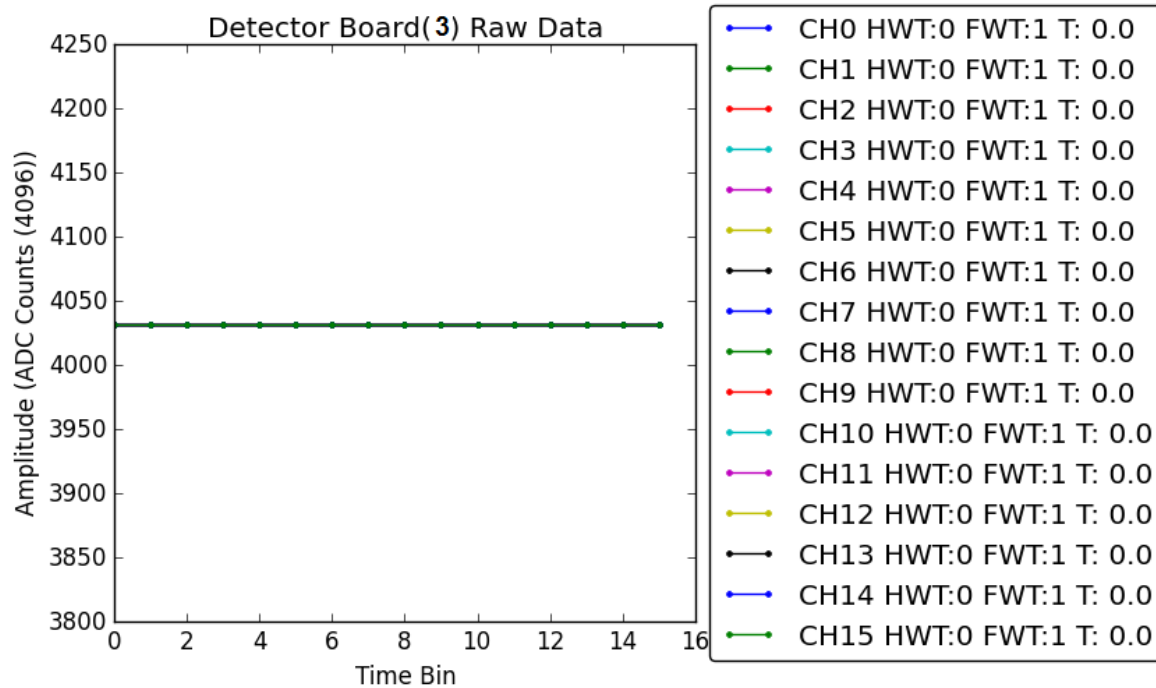


Fig. 10.1: ADC PAT SYNC pattern plot

10.2.2 Test With External Trigger

For this next test, you need to set up the OpenPET hardware and install the firmware and software by following the instructions detailed in *Getting Started - Small System Oscilloscope Mode* (page 19). After that, **POWER CYCLE** your chassis. You can then configure the system and acquire test communication data using the *Simple Example* (page 35) for a single detector board in slot 3.

10.2.3 Test Trigger

If you have confirmed that your digital communication chain and analog circuitry are functioning properly with the two tests described above, then you need to check your detector signals and trigger. Specifically, you should check your trigger mask and DAC settings to make sure that your trigger thresholds are set properly for the input signals from your detector module (see *Command ID: 0x0009* (page 69) and *Command ID: 0x000A* (page 70) for more details on how to set and read the trigger mask).

Errata

For an up-to-date list see the *issues* page in Bitbucket repository: [SupportBoard](#), [DetectorBoard](#), and [Software](#).

11.1 Firmware

11.1.1 SupportBoard

11.1.2 DetectorBoard

- High Performance TDC is disabled by default. Rebuild firmware with `g_en_tdc : boolean := true;` to enable it.

11.2 Embedded Software

11.2.1 SupportBoard

Main FPGA

IO FPGAs

11.2.2 DetectorBoard

11.3 Software

11.4 Hardware

- DetectorBoard front panel LEDs: CLK OK is swapped with V OK.

Index

- `genindex`

Symbols

16-Channel Detector Board, [47](#)

A

Altera Tools, [24](#)

B

Bus IO, [46](#)

C

Coincidence Unit, [10](#)

Coincidence Unit Controller, [10](#)

Commands, [57](#)

D

Data Acquisition, [34](#)

Data Description and Arrangement, [36](#)

Detector Board, [44](#)

Detector Unit, [9](#)

Downloading Software & Firmware, [21](#)

E

Errata, [108](#)

F

Firmware & Software Structures, [16](#)

G

Getting Started, [18](#)

I

Installing OpenPET Firmware & Software, [27](#)

L

Large System, [14](#)

O

OpenPET CAT, [78](#)

Q

QuickUSB, [22](#)

R

Release Notes, [1](#)

S

Small System, [12](#)

Standard System, [12](#)

Support Board, [51](#)

Support Crate, [9](#)

U

USB-Blaster, [25](#)