

**The OpenPET Firmware and Software Programmer Guide**

Qiyu Peng ([qpeng@lbl.gov](mailto:qpeng@lbl.gov))

Version 1.0

Contents

[Table of Figures 4](#_Toc358895204)

[Abbreviations 6](#_Toc358895205)

[1. System Hierarchy 7](#_Toc358895206)

[*1.1* *Introduction* 7](#_Toc358895207)

[*1.2* *Standard and compact system configurations* 7](#_Toc358895208)

[*1.3* *System hardware, firmware and software structures* 12](#_Toc358895209)

[*1.3.1* *System hardware structure* 12](#_Toc358895210)

[*1.3.2* *System firmware and software structure* 14](#_Toc358895211)

[*1.4* *System addressing strategies* 15](#_Toc358895212)

[*1.4.1* *System configuration profile* 16](#_Toc358895213)

[*1.4.2* *Node type register and absolute address register.* 16](#_Toc358895214)

[*1.4.3* *Offspring configuration profile* 19](#_Toc358895215)

[*1.4.4* *Firmware and software version control registers* 20](#_Toc358895216)

[*1.4.5* *Absolute address assignment strategy* 21](#_Toc358895217)

[2. System commands and responses 23](#_Toc358895218)

[*2.1* *System commands and responses* 23](#_Toc358895219)

[*2.2* *Format of system commands and responses* 23](#_Toc358895220)

[*2.3* *Definition of 16-bit command/response ID word* 23](#_Toc358895221)

[*2.4* *Commands/responses for CUC, MB, DB, CDUC nodes* 24](#_Toc358895222)

[*2.4.1* *Low level hardware devices control commands* 24](#_Toc358895223)

[*2.4.2* *System configuration commands.* 25](#_Toc358895224)

[*2.4.3* *High-level application-specified commands.* 25](#_Toc358895225)

[3. List mode data 27](#_Toc358895226)

[*3.1* *Event words and status words* 27](#_Toc358895227)

[*3.2* *Event words* 27](#_Toc358895228)

[*3.2.1* *Event Data Paths* 27](#_Toc358895229)

[*3.2.2* *Event Data length* 28](#_Toc358895230)

[*3.2.3* *Event Data formats* 29](#_Toc358895231)

[*3.3* *Status words* 39](#_Toc358895232)

[*3.3.1* *General information* 39](#_Toc358895233)

[*3.3.2* *General definition of a status word* 40](#_Toc358895234)

[*3.3.3* *OpenPET standard status words* 41](#_Toc358895235)

[*1.5.1* *Reserved status words and user-defined status words* 46](#_Toc358895236)

[*3.4* *Coding and decoding of list mode data* 46](#_Toc358895237)

[*3.4.1* *Redundancy and integrity of list mode data* 46](#_Toc358895238)

[*3.4.2* *Summary of event data formats* 47](#_Toc358895239)

[*3.4.3* *Algorithm to decode list mode data file* 50](#_Toc358895240)

[4. System software framework models 51](#_Toc358895241)

[*4.1* *Host computer software* 51](#_Toc358895242)

[*4.1.1* *Host computer software functions* 51](#_Toc358895243)

[*4.1.2* *Host computer software model* 52](#_Toc358895244)

[*4.2* *CUC/MB/DUC/CDUC software* 54](#_Toc358895245)

[*4.3* *An example (how host computer and CUC/MB/DUC/CDUC NIOS work together)* 56](#_Toc358895246)

[5. System firmware (FPGA) framework models 60](#_Toc358895247)

[*5.1* *CUC/MB/DUC/CDUC firmware (FPGA) framework model* 60](#_Toc358895248)

[*5.2* *DB (FPGA) framework model* 62](#_Toc358895249)

[6. OpenPET Programming Tools / Environment 63](#_Toc358895250)

[*6.1* *Collaboration through Github* 63](#_Toc358895251)

[*6.2* *OpenPET Programming Code Styles* 63](#_Toc358895252)

[*6.2.1* *C Programming Code Styles* 64](#_Toc358895253)

[*6.2.2* *VHDL Programming Code Styles* 70](#_Toc358895254)

[Appendix 1 The OpenPET System Boot-up Sequences 77](#_Toc358895255)

[References 78](#_Toc358895256)

# Table of Figures

[Figure 1 Tree network topology of OpenPET system. A CUC which supports 8 MBs is housed in a crate with power supply. The assembled unit is called coincidence unit. A DUC which supports 8 DBs is also housed in a crate with power supply. The assembled unit is called detector unit. One coincidence unit fans out up to 64 detector units. The hardware of CUC and DUC are the same (SB). The differences are the firmware and software configurations. 8](#_Toc358895257)

[**Figure 2** The compact system configuration. A CDUC performs functions of both CUC and DUC. 9](#_Toc358895258)

[**Figure 3** The simplified standard system configuration without MB. 10](#_Toc358895259)

[**Figure 4** Mixed system configurations don’t conflict with the standard OpenPET protocol. But they are not recommended. 11](#_Toc358895260)

[**Figure 5** standard system hardware structure. 12](#_Toc358895261)

[**Figure 6** System firmware and software structure. 15](#_Toc358895262)

[**Figure 7** Node type register and absolute address register for (a) CUC, (b) MB board, (c) DUC board and (d) DB board. “X” means not used. “x” mean 0 or 1. 17](#_Toc358895263)

[**Figure 8** Node type register and absolute address register values some nodes (CUC, MB3, DUC3 and DB2) in a system with (a) standard (b) compact and (c) simplified standard configuration. 19](#_Toc358895264)

[**Figure 9** Thehost PC polls the nodes one by one to read offspring configuration profiles and establish system configuration profile. Note that the host PC acquires the absolute addresses of all offspring nodes of a parent node through its “offspring configuration profile”. So the all the commands sent here are point-to-point commands with absolute addresses. 22](#_Toc358895265)

[**Figure 10** Format of system commands and responses 23](#_Toc358895266)

[**Figure 11** The definition of the 16-bit command/response ID word 24](#_Toc358895267)

[**Figure 12** Event data path for standard configuration 27](#_Toc358895268)

[**Figure 13** Event data path for compact configuration 28](#_Toc358895269)

[**Figure 14** Event data path for simplified standard configuration 28](#_Toc358895270)

[**Figure 15** Data Length combination 0 (32🡪 32) 28](#_Toc358895271)

[**Figure 16** Data Length combination 1 (32🡪 64) 29](#_Toc358895272)

[**Figure 17** Data Length combination 2 (64🡪 64) 29](#_Toc358895273)

[**Figure 18** Data Length combination 3 (64🡪 64) 29](#_Toc358895274)

[**Figure 19** Format of the 32-bit single event data 30](#_Toc358895275)

[**Figure 20** Demonstrations of how the addresses are added to the event data when it transmitted from DB to CUC. 31](#_Toc358895276)

[**Figure 21** Format of the 32-bit raw ADC data. 33](#_Toc358895277)

[**Figure 22** Demonstrations of how the addresses are added to the event data when it transmitted from DB to CUC. 34](#_Toc358895278)

[**Figure 23** The format of the 64-bit single event data. 36](#_Toc358895279)

[**Figure 24** The format of the 64-bit coincidence event data. 38](#_Toc358895280)

[**Figure 25** The format of the 32-bit coincidence event data. 39](#_Toc358895281)

[**Figure 26** The format of the 64-bit status word. 40](#_Toc358895282)

[**Figure 27** The format of event data format/timer word. 42](#_Toc358895283)

[**Figure 28** The format of Coincidence Event rate word. 43](#_Toc358895284)

[**Figure 29** The format of single event rate word. 45](#_Toc358895285)

[**Figure 30** In “event data format/timer word”, bit 56 and bit 53 to 48 (7 bits) are used to define event data.. 47](#_Toc358895286)

[**Figure 31** flowchart to decode a list mode data file. 50](#_Toc358895287)

[**Figure 32** The framework model for the host computer software. 53](#_Toc358895288)

[**Figure 33** The framework model for the CUC/MB/DUC/CDUC software. 55](#_Toc358895289)

[**Figure 34** The command message package, an example. 56](#_Toc358895290)

[**Figure 35** The response message package, an example. 57](#_Toc358895291)

[**Figure 36** CUC/MB/DUC/CDUC firmware (FPGA) framework model. 60](#_Toc358895292)

[**Figure 37** The DB firmware framework model. 62](#_Toc358895293)

[**Figure 38** C code program files style. 64](#_Toc358895294)

[**Figure 39** Program file prolog format. 67](#_Toc358895295)

[**Figure 40** function prolog format. 69](#_Toc358895296)

[**Figure 41** The OpenPET System Boot-up Sequence. 77](#_Toc358895297)

# Abbreviations

SB: Support Board

DB: Detector Board

MB: Multiplexer Board (also called high-speed data transceiver board)

CU: Coincidence Unit

DU: Detector Unit

CDU: Coincidence/Detector Unit

CUC: Coincidence Unit Controller

DUC: Detector Unit Controller

CDUC: Coincidence/ Detector Unit Controller

FPGA: Field-programmable gate array

NIOS II: A 32-bit embedded-processor architecture designed specifically for the Altera FPGAs

VHDL: VHSIC hardware description language

CRC: Cyclic redundancy check

FSM: Finite State Machine

# System Hierarchy

## *Introduction*

The OpenPET system is essentially a computer network with a tree topology. The OpenPET configuration strategy needs to fulfill the following two basic requirements:

1. Compatible with different detector modules (e.g. single analog channel addressing, single crystal address for conventional block detector and etc.);
2. Compatible with different system configurations (e.g. different size of systems shown in Figure 1, 2 and 3).

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

## *Standard and compact system configurations*

As shown in Figure 1, the OpenPET system adapts a tree network topology. The basic characteristics are:

1. Host computer is the central 'root' node (the top level of the hierarchy);
2. Host computer is connected to a CUC node that is one level lower in the hierarchy (i.e., the second level) with a point-to-point link.
3. The CUC node is connected to up to 8 MB nodes that are one level lower in the hierarchy (i.e., the third level) with point-to-point links.
4. The MB node is connected to up to 8 DUC nodes that are one level lower in the hierarchy (i.e., the fourth level) with point-to-point links.
5. The DUC node is connected to up to 8 DB nodes that are one level lower in the hierarchy (i.e., the fifth level) with point-to-point links.
6. The DB node supports 32 analog channels which can either be addressed individually or addressed in groups (e.g. 4 channels for one conventional block detector)



Figure 1 Tree network topology of OpenPET system. A CUC which supports 8 MBs is housed in a crate with power supply. The assembled unit is called coincidence unit. A DUC which supports 8 DBs is also housed in a crate with power supply. The assembled unit is called detector unit. One coincidence unit fans out up to 64 detector units. The hardware of CUC and DUC are the same (SB). The differences are the firmware and software configurations.

The system configuration shown in Figure 1 is standard system configuration. The OpenPET system can also be configured in a compact mode as shown in Figure 2. In compact mode, a SB is configured as a CDUC which interfaces with DBs and performs coincidence functions.



**Figure 2** The compact system configuration. A CDUC performs functions of both CUC and DUC.

A simplified standard system shown in Figure 3 is legal for now (before the MB is developed), but not recommended.



**Figure 3** The simplified standard system configuration without MB.

Mixed system configurations as shown in Figure 4 are not allowed.



(a)



(b)

**Figure 4** Mixed system configurations don’t conflict with the standard OpenPET protocol. But they are not recommended.

## *System hardware, firmware and software structures*

### *System hardware structure*

The standard system hardware structure is shown as follows.



**Figure 5** standard system hardware structure.

The host computer is equipped with USB/Ethernet/Fiber-Optic connection and high speed hard disk. It sends commands to the CUC through USB/Ethernet/ Fiber-Optic, read data from the CUC and save them into the hard disk in real time.

The CUC and DUC have the same hardware configurations. The core component is a FPGA which is configured as an embedded NIOS II micro processor. The NIOS II μ-processor is interfaced to register array which is implemented in the FPGA and peripheral devices include:

1. **Digital IO**

Through Main FPGA:

16 bits digital IO (4 bits direction control)

10 Bits LED bar

Two logic analyzer connectors (16+1 bits each)

Through IO FPGA 1:

16 bits digital IO (4 bits direction control)

10 Bits LED bar

One logic analyzer connector (16+1 bits each)

Through IO FPGA 2:

16 bits digital IO (4 bits direction control)

10 Bits LED bar

One logic analyzer connector (16+1 bits each)

1. **Memory**

Through Main FPGA:

* Flash memory

EPCS64 flash memory

64MB flash memory

* RAM

4MB SRAM

1GB DDR2-SO-DIMM SDRAM

Through IO FPGA 1:

* RAM

2MB SRAM

Through IO FPGA 2:

* RAM

2MB SRAM

1. **SD card**
2. **Miscellaneous devices**

Power monitor

Temperature sensors

Two RS232 interfaces

\* Note that the data/status/command communication interfaces (USB, Gb Ethernet, Fiber-Optic and high speed digital data communication ports) are not connected to the NIOS II μ-processor directly. NIOS II μ-processor talks to those communication interfaces through the register array implemented in the FPGAs.

The MB hardware configurations are not finally determined yet. But we are planning to use low cost Altera Arria V FPGA (about $100) which has 12 channels of high speed (6.375Gbps) transceivers. Before the MB is developed, we can design some simple interface board to connect the DU to the CU directly. Our firmware and software scheme ensures that the alternative method also works.

The DB hardware configuration can be divided into two parts: digital part and analog part. The digital part includes FPGA, DIO (digital IO, LED and logic analyzer connector), memory (EPCS64, SRAM), and digital communication interfaces (USB and high speed digital data communication ports). The analog part includes signal conditioning circuits, ADC, CFD or raising edge discriminator and TDC.

There is no NIOS II μ-processor implemented in the DB FPGA. The configuration and calibration of the DB boards are fully controlled by the DUC through the high speed parallel digital data and command communication ports.

### *System firmware and software structure*

Figure 6 shows the system firmware and software structure.

In summary, there are four types of software codes and seven types of firmware codes run in the whole system:

Software:

1. Software which run in Host computer

Function: top level system configuration and calibration, data acquisition and analysis.

1. Software which run in CUC NIOS II μ-processor;

Function: CUC SB board monitor and management, coincidence pair configuration;

1. Software which run in MB NIOS II μ-processor;

Function: MB board monitor and management, multiplexer configuration, high-speed transceiver;

1. Software which run in DUC NIOS II μ-processor;

Function: DUC SB board monitor and management, DB board configuration, calibration, monitor and management.

Firmware:

1. Firmware for CUC main FPGA (Altera Cyclone III EP3C40F780)

Function: CUC SB board monitor and management, coincidence pair configuration;

1. Firmware for CUC IO FPGA 1 and 2 (two identical FPGAs, Altera Cyclone III EP3C40F780)

Function: CUC command/status flow and high speed dataflow router;

1. Firmware for MB FPGA (Altera Arria V FPGA)

Function: MB monitor and management, multiplexer configuration, MB command/status flow and high speed dataflow router;

1. Firmware for DUC main FPGA (Altera Cyclone III EP3C40F780)

Function: DUC SB board monitor and management, DB board configuration, calibration, monitor and management.

1. Firmware for DUC IO FPGA 1 and 2 (two identical FPGAs, Altera Cyclone III EP3C40F780)

Function: DUC command/status flow and high speed dataflow router;

1. Firmware for DB FPGA

Function: ADC control, energy calculation and correction, crystal decoding, Energy threshold, TDC, time correction and etc;

\* Note: in our SB hardware design for CUC, DUC and CDUC , we adapted three cheap FPGAs (main FPGA, IO FPGA 1 and 2), instead of using one single expensive FPGA, to decrease the hardware cost.



**Figure 6** System firmware and software structure.

## *System addressing strategies*

To support system configurations with different sizes (as shown in Figure 1 and Figure 2), we adapt system hardware node addressing strategy as follows:

### *System configuration profile*

Host computer software maintains the whole system configuration using a software data structure called “system configuration profile”. The host computer can automatically indentify system configurations (e.g. different size of systems shown in Figure 1 and 2) according to the contents of the system configuration profile.

The system configuration profile includes the following information:

* A tree data structure of the whole system
* A data structure contains details of all nodes in the tree (node types, absolute addresses, connection status and etc.)
* Integrity of all the hardware devices connected to all the nodes

The definition of system configuration profile <add the definition here>.

### *Node type register and absolute address register.*

Each node has a node type registerand an absolute address register. Node type register is a 4-bit register hardwired (read only) by the node firmware. Node absolute address register is a readable and writable 12-bit register. The host PC assigns the absolute address for each node.

Definition of node type register:

0000: CUC node

0001: MB node

0010: DUC node

0011: DB node

0100: CDUC node (for small size system configuration)

0101~1110: unused

1111: reserved for host computer

Definition of absolute address register:

Bit 0~2: MB address

Bit 3~5: DUC address

Bit 6~8: DB address

Bit 9~11: unused

\* Note 1: node type register is hardwired in the FPGA firmware. So it can also be used to control the compilation of the FPGA firmware.

Note 2: node type code “1111” is reserved for host computer

Note 3: the absolute address register makes it possible to debug/address a node directly using a JTAG cable connected to that node.

Figure 7 demonstrates how to combined Node type register and absolute address register (16 bits in total) for system CUC, MB, DUC and DB node addressing.



**Figure 7** Node type register and absolute address register for (a) CUC, (b) MB board, (c) DUC board and (d) DB board. “X” means not used. “x” mean 0 or 1.

Figure 8 demonstrates the node type register and absolute address register values of some nodes in systems with standard, compact and simplified standard configurations.



(a)



(b)

 (c)

**Figure 8** Node type register and absolute address register values some nodes (CUC, MB3, DUC3 and DB2) in a system with (a) standard (b) compact and (c) simplified standard configuration.

### *Offspring configuration profile*

Each node maintains information of its direct offspring nodes using a software data structure called “offspring configuration profile”. The offspring configuration profile includes the following information:

* Offspring Connection Status
* Offspring Enable/disable Status
* Offspring’s node types and absolute addresses

The offspring connection status registers/variables are used to indicate how many nodes are in the system and how they are connected to the system.

* The CUC node uses an 8-bit offspring connection status **registers/variables** to record how many MB nodes are connected. For example, if there are 8 MBs plugged in slot 3 (Figure 1), the content of the 8-bit offspring connection status registers/variables is “11111111” ;
* The MB node uses an 8-bit offspring connection status **registers/variables** to record how many DUC nodes are connected. For example, if there is no DUC plugged in MB0 (Figure 1), the content of the 8-bit offspring connection status registers/variables in MB0 is “00000000” ;
* The DUC node uses an 8-bit offspring connection status **registers/variables** to record how many DB nodes are connected. For example, if there are 8 DBs plugged in DUC3 (Figure 1), the content of the 8-bit offspring connection status registers/variables register is “11111111” ;
* \* The DB node uses a 32-bit offspring connection status **register** to record how many analog input channels are connected. For example, if all 32 analog channels of DB2 are connected (Figure 1), the content of the 32-bit offspring connection status register is “11111111111111111111111111111111” ;

The default value of the offspring connection status registers/variables are 0.

\* Note: We can detect how many analog channels are connected by counting the events from those channels. If there is no event coming in from a channel, we know that channel is not connected (or broken).

The OpenPET system uses offspring enable/disable status registers/variables to enable or disable nodes in the system. When the value of an offspring enable/disable status register/variable is 0, the command, states and data stream from that offspring node will be ignored/disabled.

* The CUC node uses an 8-bit offspring enable/disable status registers/variables to enable/disable MB nodes. For example, there are 8 MBs plugged in slot 3 (Figure 1). MB1 is disabled if the content of the 8-bit offspring enable/disable status registers/variables is “11111101” ;
* The MB node uses an 8-bit offspring enable/disable status registers/variables to enable/disable DUC nodes;
* The DUC node uses an 8-bit offspring enable/disable status registers/variables to enable/disable DB nodes;
* The DB node uses a 32-bit offspring enable/disable status registers/variables to enable/disable analog channels. For example, if all 32 analog channels of DB2 are enabled (Figure 1), the content of the 32-bit offspring enable/disable status registers/variables is “11111111111111111111111111111111” ;

The default value of the offspring enable/disable status registers/variables are 1.

Offspring’s node types and absolute addresses are also recorded in the offspring configuration profile.

### *Firmware and software version control registers*

A 32 bits read-only register is used for firmware (16 bits) and software (16 bits) version control. The 16 bits firmware version register is hardwired by the node firmware. The 16 bits firmware version register is hardwired by the node firmware. The 16 bits software version register is a read-only constant value defined in the node software.

### *Absolute address assignment strategy*

When the system is reconfigured (or restarted), the host computer is able to acquire the information of new system configuration automatically and update the system configuration profile by polling the nodes one by one (Figure 9).

Step 1: assign absolute addresses to all nodes connected to the system

The procedure is as follows:

1. Host computer sends an “CUC/CDUC address assignment command” to the CUC or CDUC;
2. CUC or CDUC responses (after receiving commands from the host computer):
3. Polls all slots one by one to identify if the slots are occupied and what types of nodes are plugged in or connected to those slots;
4. Assigns and sends absolute addresses to all offspring one by one.
5. Updates its offspring configuration profile (after receiving responses from offspring);
6. Sends an acknowledge response to the host computer;
7. Responses from the offspring of CUC or CDUC (after accepting new absolute addresses from the CUC or CDUC):
8. Updates its absolute address register.
9. Sends an acknowledge response to its parent;
10. Polls all slots one by one to identify if the slots are occupied and what types of nodes are plugged in or connected to those slots;
11. Assigns and sends absolute addresses to all offspring one by one.
12. Updates its offspring configuration profile (after receiving responses from offspring).
13. The offspring of the offspring of CUC or CDUC repeat procedure (3) after accepting new absolute addresses from the offspring of CUC or CDUC, until the offspring configuration profile in all nodes are updated. Note that when a DB node receives a new absolute address from its parent, it only needs to update its absolute address register and send an acknowledge response to its parent.

Step 2: Host PC read offspring configuration profile from all nodes and establish system configuration profile

After step 1, the host PC sends commands to read offspring configuration profiles from all the nodes one by one, and establish system configuration profile (Figure 9). If required, the host PC can also disable nodes from the system by setting zeros to offspring enable/disable status registers/variables. As described before, the offspring enable/disable status registers/variables are used to disable nodes without physically removing them from the OpenPET system. When a bit of an offspring enable/disable status register/variable is set to zero, the node represented by that bit will be isolated from the OpenPET tree. All event data from that node will be dropped. However, the host computer will still have access to non-event data (such as registers, RAM and etc.). The offspring enable/disable status registers/variables make it possible to selectively read/disable event data from a certain analog channel (or channels) for debugging.



**Figure 9** Thehost PC polls the nodes one by one to read offspring configuration profiles and establish system configuration profile. Note that the host PC acquires the absolute addresses of all offspring nodes of a parent node through its “offspring configuration profile”. So the all the commands sent here are point-to-point commands with absolute addresses.

# System commands and responses

There are two types of bit flows in the system. One is upstream list mode data flow, which is always from offspring nodes to the parent nodes. One is bi-directional system commands and responses flow.

## *System commands and responses*

## *Format of system commands and responses*

The length of the system commands and responses varies from 8 bytes to 261 bytes, depended on the functions of specific commands/responses. The format is shown as follows:



**Figure 10** Format of system commands and responses

A command/response comprises of five parts:

* Command/response ID (16 bits) – what kind of command/response. The definition of Command/response ID will be described in more details later.
* Source address (16 bits) – where the command/response from. Source address includes the node type code (4 bits) and absolute address (12 bits) defined before. Note that the node type code of the host computer is “1111” and the host computer has no absolute address.
* Target address (16 bits) – where the command/response to. The definition is the same as the 16-bit source address.
* Data payload. The length of the data payload depends on the functions of specific commands/responses.
* CRC: Cyclic redundancy check

## *Definition of 16-bit command/response ID word*

The definition of the 16-bit command/response ID word is shown as follows:



**Figure 11** The definition of the 16-bit command/response ID word

* Bit 15: Command/response flag.

“0”: command (downstream from parent nodes to offspring nodes);

“1”: response (upstream from offspring nodes to parent nodes).

* Bit 14 to 11: Command/response node type. We will have four sets of commands/responses for four types of nodes.

0000: Command/response for CUC node

0001: Command/response for MB node

0010: Command/response for DUC node

0011: Command/response for DB node

0100: Command/response for CDUC node (for small size system configuration)

0101~1110: unused (users can define command/response by themselves).

1111: reserved for host computer

* Bit 10: Standard/user-defined command/response flag

“0”: Standard command/response;

“1”: User-defined command/response.

* Bit 9 to 0: detailed command/response ID

## *Commands/responses for CUC, MB, DB, CDUC nodes*

The commands/responses for CUC, MB, DB, CDUC nodes can be divided into three types: (1) Low level hardware devices control commands; (2) System configuration commands and High-level application-specified commands.

### *Low level hardware devices control commands*

Register read command set,

Register write command set,

\* Note that some hardware devices (such as ADCs, LEDs, temperature sensors, voltage monitors, current monitors) are mapped to the registers. So they can be controlled conveniently by reading/writing the registers.

RAM read command,

RAM write command,

Flash memory read command,

Flash memory command,

SD read command,

SD write command,

<more commands will be added to this list>

### *System configuration commands.*

System reset command,

Load/run firmware command,

Load/run software command,

System hardware version report command,

System firmware version report command,

System software version report command,

Address assignment command,

Node enable/disable command,

Hardware device integrity test command set

(A set of commands for all the hardware devices on the node)

<more commands will be added to this list>

### *High-level application-specified commands.*

Energy calculation and correction command set,

Energy window setting command set,

Crystal decoding command set,

TDC and time correction setting command set;

Data acquisition mode setting command set,

Coincidence setting command set

(Coincidence pairs setting, time window setting, data format, status word setting and etc.)

<more commands will be added to this list>

# List mode data

## *Event words and status words*

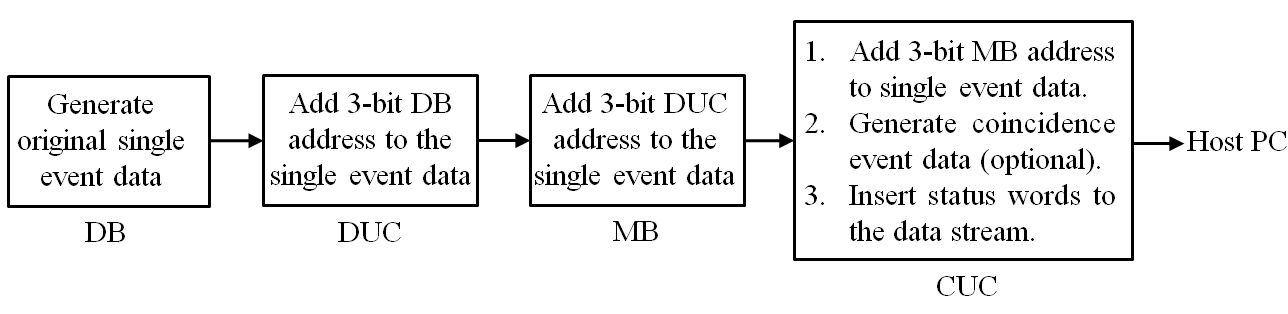
In general, there are two kinds of words in the OpenPET list mode data:

* **Event words**, include
* Single event words, and
* Coincidence
* **Status words** transmit information other than the event data to the host PC, such as:
* Event data format
* Time information
* Single or coincidence event rate
* System configuration and status
* Application-specific information such as patient bed position, transmission source position, system temperature, ECG signal, gate signal and etc.

## *Event words*

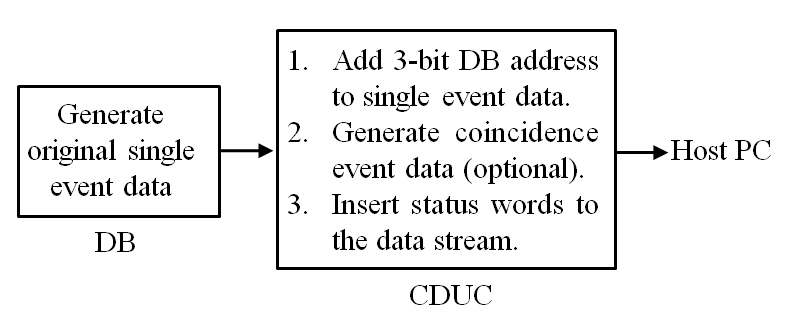
### *Event Data Paths*

The following figure shows how the event data is generated and transmitted from the DB to host PC. (1) DB generates the original single event data and transmits them to the DUC. (2) DUC adds 3-bit DB address to the event data, multiplexes them and transmits them to the MB. (3) MB adds 3-bit DUC address to the event data, multiplexes them and transmits them to the CUC. (4) CUC adds 3-bit MB address to the event data, multiplexes them, (optional) generates coincidence event data, inserts the status words to the data stream and transmits the data to the host PC.

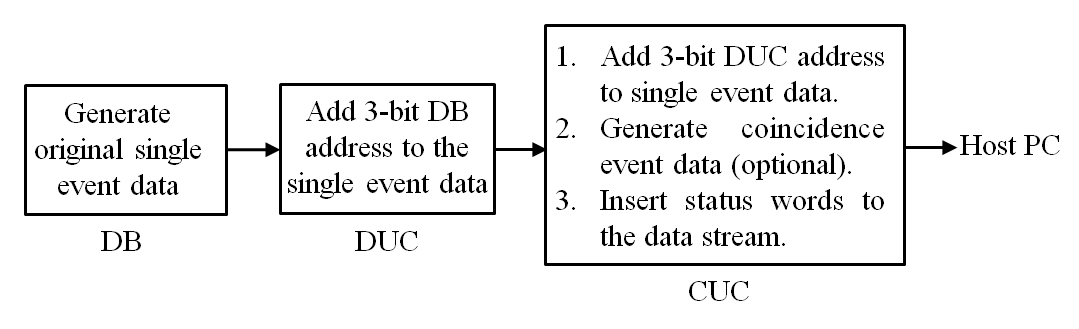


**Figure 12** Event data path for standard configuration

The OpenPET also supports a compact configuration as shown in Figure 2 and simplified standard configuration as shown in Figure 3. The diagrams of the event data path for those two configurations are shown as follows.



**Figure 13** Event data path for compact configuration



**Figure 14** Event data path for simplified standard configuration

Note that in the compact configuration, 3-bit DUC and 3-bit MB addresses are not required. Therefore, those 6 bits can be redefined by the users for payload data transmission. Similarly, in simplified standard configuration, 3-bit MB addresses are not required, and can be redefined by the users.

### *Event Data length*

To maximize the event rate and to meet the requirements for different applications, OpenPET standard uses the following event data length modes:

1. Event data from DB🡪DUC, DUC🡪 MB and MB🡪CUC: 32-bit mode or 64-bit mode.
2. Event data from CUC 🡪 host PC: 32-bit mode, 64-bit mode, or 128-bit mode.

Therefore, there are four possible data combinations of data lengths through DB to host PC :

1. **Data Length combination 0 (32🡪 32)**

The length of the data from DB🡪DUC, DUC🡪 MB and MB🡪CUC is 32 bits.

The length of the data from CUC 🡪 host PC is also 32 bits.

The interval between slice boundaries is also 8 system clocks (80MHz).

The maximum event rate is twice of that in combination 0.

DB

DUC

MB

CUC

Host PC

32 bits

32 bits

32 bits

32 bits

**Figure 15** Data Length combination 0 (32🡪 32)

1. **Data Length combination 1 (32🡪 64)**

The length of the data from DB🡪DUC, DUC🡪 MB and MB🡪CUC is 32 bits.

The length of the data from CUC 🡪 host PC is 64 bits.

The interval between slice boundaries is 8 system clocks (80MHz).

DB

DUC

MB

CUC

Host PC

32 bits

32 bits

32 bits

64 bits

**Figure 16** Data Length combination 1 (32🡪 64)

1. **Data Length combination 2 (64🡪 64)**

The length of the data from DB🡪DUC, DUC🡪 MB and MB🡪CUC is 64 bits.

The length of the data from CUC 🡪 host PC is also 64 bits.

The interval between slice boundaries is 16 system clocks (80MHz).

In this combination, more information payload can be transmitted from DB to host PC.

DB

DUC

MB

CUC

Host PC

64 bits

64 bits

64 bits

64 bits

**Figure 17** Data Length combination 2 (64🡪 64)

1. **Data Length combination 3 (64🡪 128)**

The length of the data from DB🡪DUC, DUC🡪 MB and MB🡪CUC is 64 bits.

The length of the data from CUC 🡪 host PC is 128 bits.

The interval between slice boundaries is 16 system clocks (80MHz).

In this combination, more information payload can be transmitted from DB to host PC.

DB

DUC

MB

CUC

Host PC

64 bits

64 bits

64 bits

128 bits

**Figure 18** Data Length combination 3 (64🡪 64)

The method to decode data file with different data length and data formats is described in next section. It is not recommended to mix event data with different data lengths and formats in the same data file.

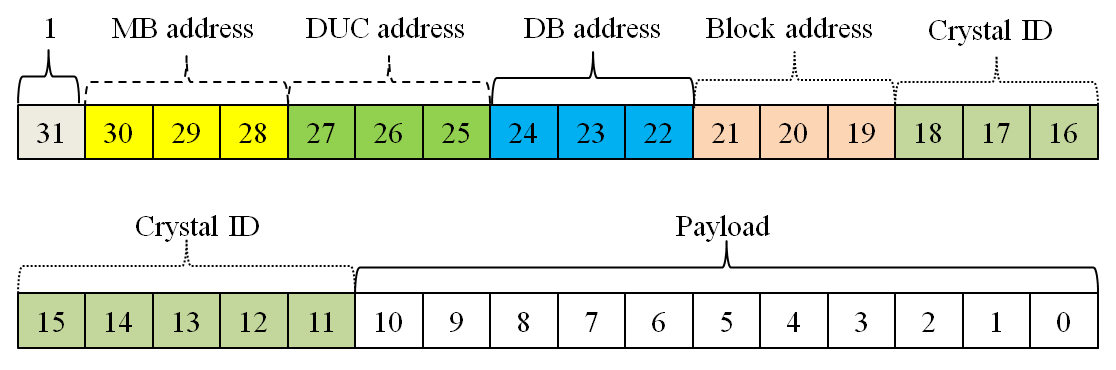
Note that combination 0 (32🡪 32) and combination 1 (32🡪 64) meet the requirements for majority of OpenPET applications. The purposes of combination 2 (64🡪 64) and combination 3 (64🡪 128) are to meet the requirements in some very rare and very unique OpenPET applications. So we suggest the users start from studying combination 0 (32🡪 32) and combination 1 (32🡪 64), and do not work on combination 2 (64🡪 64) and combination 3 (64🡪 128) until they find that combination 0 (32🡪 32) and combination 1 (32🡪 64) do not meet their requirements.

### *Event Data formats*

#### DB🡪DUC, DUC🡪 MB and MB🡪CUC

##### ***32-bit mode***

The following figure shows the format of the 32-bit single event data:



**Figure 19** Format of the 32-bit single event data

The definitions of the 32 bits are:

* **Bit 31**: Data valid bit. This bit has to be set to “1”. A parent node rejects any event data with a data valid bit set to “0”from its offspring nodes.
* **Bit 30 to 28**: 3 MB address bits. 3-bit MB addresses are assigned and added to the 32 bits event data in CUC. In a system without MB (compact configuration and simplified standard configuration), those 3 bits can be redefined as payload bits by the users.
* **Bit 27 to 25**: 3 DUC address bits. 3-bit DUC addresses are assigned and added to the 32 bits event data in MB (standard configuration) or CUC (simplified standard configuration). In a compact system, those 3 bits can also be redefined as payload bits by the users.
* **Bit 24 to 22**: 3 DB address bits. 3-bit DB addresses are assigned and added to the 32 bits event data in DUC or CDUC. Those 3 bits CANNOT be redefined.
* **Bit 21 to 19**: 3 block detector address bits. 3-bit block detector addresses are assigned in the DB when a single event is detected. A DB has 32 channels, and therefore it supports up to 8 conventional block detector modules. The users can redefine those 3 bits if they do not use conventional block detector modules.
* **Bit 18 to 11**: 8 crystal ID address bits. 8-bit crystal ID addresses are assigned in the DB when a single event is detected. The maximum crystal matrix supported in this format is 16 x16. The users can redefine those 8 bits if they do not use conventional block detector modules.
* **Bit 10 to 0**: 11 bits of payload data. 11-bit payload data are generated in the DB when a single event is detected. The detailed definitions of those 11-bit payload data will be described in the next section.

The following figure demonstrates how the addresses are added to the event data when it transmitted from DB to CUC.



**Figure 20** Demonstrations of how the addresses are added to the event data when it transmitted from DB to CUC.

The 11-bit payload data have definitions for five standard data mode (time, energy, test and etc.) as follows:

* **Time mode –** Bit 10~0, TDC data bits (11 bits, LSB: 50ps).
* **Energy mode –** Bit 10~0, energy data bits (11 bits).
* **Test mode1 –** Bit 10~0, test pattern data: 10101010101.
* **Test mode2 –** Bit 10~0, test pattern data: 01010101010.
* **Anger-logic flood map plot mode –** note that in this mode, the 8 crystal ID address is not needed. Therefore they are redefined for

-- Bit 18, not used.

-- Bit 17~9, Y value (9 bits).

-- Bit 8~0, X value (9 bits).

One disadvantage of the 32-bit single event data definition is that it only allows 11 bits of payload data. However, many bits defined in this protocol (marked in blue) can be redefined by the users for payload data transmission in different system configurations and applications. **The maximum payload data bits can be as much as 28 bits (compact configuration, not conventional block detector).**

The six possible user-defined modes are listed as follows:

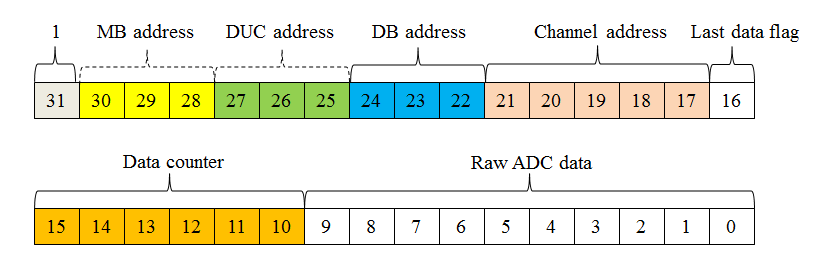
* **User-defined mode 1 –** 11 bits of payload, but the definition of the payload bits are not one of the five standard definitions above mentioned.
* **User-defined mode 2 –** 14 bits of payload. This mode is for simplified standard system or compact system that has no MB nodes, and the 3 bits of MB addresses are redefined for payload data.
* **User-defined mode 3 –** 17 bits of payload. This mode is for compact system that has no MB and DUC nodes, and the 3 bits of MB addresses and the 3 bits of DUC addresses are redefined for payload data.
* **User-defined mode 4 –** 22 bits of payload. 3 block detector address bits and 8 crystal ID address bits are redefined for payload data.
* **User-defined mode 5 –** 25 bits of payload. This mode is for simplified standard system or compact system that has no MB nodes, and the 3 bits of MB addresses, 3 block detector address bits and 8 crystal ID address bits are redefined for payload data.
* **User-defined mode 6 –** 28 bits of payload. This mode is for compact system that has no MB and DUC nodes, and the 3 bits of MB addresses, the 3 bits of DUC addresses, 3 block detector address bits and 8 crystal ID address bits are redefined for payload data.

##### ***Raw ADC data mode (32-bit)***

Raw ADC data mode is essentially a special case of 32-bit data mode. There are two major differences between 32-bit raw ADC data mode and regular 32-bit data mode:

1. In 32-bit raw ADC data mode, a train of 32-bit data (instead of only one 32-bit data) needs to be transmitted. The timing of data train transmission crosses several slice boundaries, can cannot be interrupted. Therefore, when raw data train from a DB is being transmitted, it will (a) freeze the single event trigger in that DB, and (b) occupy the whole event data path from DB to CUC to prevent the data train from being interrupted, until the transmission of the whole data train is done.
2. In 32-bit raw ADC data mode, there is no need to assign 3 block detector address bits and 8 crystal ID address bits. Instead, 5 bits of channel addresses need to be assigned to address 32 analog channels in a DB board. Therefore, there are 17 bits of payload in 32-bit raw ADC data mode.

The following figure shows the format of the 32-bit raw ADC data:



**Figure 21** Format of the 32-bit raw ADC data.

The definitions of the 32 bits are:

* **Bit 31**: Data valid bit. This bit has to be set to “1”. A parent node rejects any event data with a data valid bit set to “0”from its offspring nodes.
* **Bit 30 to 28**: 3 MB address bits. 3-bit MB addresses are assigned and added to the 32 bits event data in CUC. In a system without MB (compact configuration and simplified standard configuration), those 3 bits can be redefined as payload bits by the users.
* **Bit 27 to 25**: 3 DUC address bits. 3-bit DUC addresses are assigned and added to the 32 bits event data in MB (standard configuration) or CUC (simplified standard configuration). In a compact system, those 3 bits can also be redefined as payload bits by the users.
* **Bit 24 to 22**: 3 DB address bits. 3-bit DB addresses are assigned and added to the 32 bits event data in DUC or CDUC. Those 3 bits can NOT be redefined.
* **Bit 21 to 17**: 5 channel address bits. 5-bit channel addresses are assigned in the DB when a single event is detected.
* **Bit 16**: last data flag bit. This bit keeps zeros until sending the last word of the raw data train. When a data train is sent to a node and there is no other data transmission on going, the data train will occupy that node. No other data will be able to plug in and interrupt current data train. In the last word of the raw data train, the last data flag bit (bit 16) will be set to “1”. The node will release the event data path to new raw data train after detecting “1” in the last data flag bit.
* **Bit 15 to 10**: 6 counter bits. Those 6 bits are used to count the raw ADC data (from 0 to 63). The maximum length of the raw data train is 64.
* **Bit 9 to 0**: 10 bits of raw ADC data. 10-bit raw ADC data are generated in the DB when a single event is detected.

The following figure demonstrates how the addresses are added to the event data when it transmitted from DB to CUC.



**Figure 22** Demonstrations of how the addresses are added to the event data when it transmitted from DB to CUC.

Note that there will be in maximum 64 raw ADC data loaded in the data train. In some applications, the users may need to collect some additional information with the raw ADC data (DOI, TOF and etc.). In that situation, the users can redefine the 10 payload bits in some of the 32 bits data in the data train. For example, one can load 63 raw ADC data in the first 63 32-bit data and load 10 bits of TDC data in the last 32-bit data.

Also note that since a data train has to occupy the whole event data path from DB to CUC, the CUC is not able to collect more than one event in any time to generate any coincidence event. In the last word of the raw data train, the last data flag bit (bit 16) will be set to “1”. The hardware nodes along the data path (DUC3, MB3 and CUC in figure 22) will release the event data path after detecting “1” in the last data flag bit.

The RAW ADC data train can be triggered in different modes as follows:

* **Single channel mode (“0000”) –** in this mode, all 32 analog channels are triggered independently. Only the raw data from the triggered ADC channel is loaded to the data train in each event data transmission. This mode is useful for debugging single channels of photo detectors.
* **Four-channel mode (“0001”)–** in this mode, 32 analog channels are divided into 8 groups (group1: ch0~3, group2: ch4~7, …, group8: ch28~31). Each group has 4 channels. Any of the 4 channels in a group can trigger an event transmission. Raw data from all 4 channels in that group will be loaded to the data train in each event data transmission. This mode is useful for debugging block detectors.
* **Sixteen-channel mode (“0010”)–** in this mode, 32 analog channels are divided into 2 groups (group1: ch0~15, group2: ch16~31). Any of the 16 channels in a group can trigger an event transmission. Raw data from all 16 channels in that group will be loaded to the data train in each event data transmission. This mode is useful when testing cross-talks between analog channels in 16-ch detector boards.
* **Thirty-two-channel mode (“0011”)–** in this mode, any of the 32 channels in a group can trigger an event transmission. Raw data from all 32 channels will be loaded to the data train in each event data transmission. This mode is useful when testing cross-talks between analog channels in 32-ch detector boards.
* **Reserved raw ADC modes (“0100” ~“1000”)**
* **User-defined raw ADC modes (“1000” ~“1111”)**

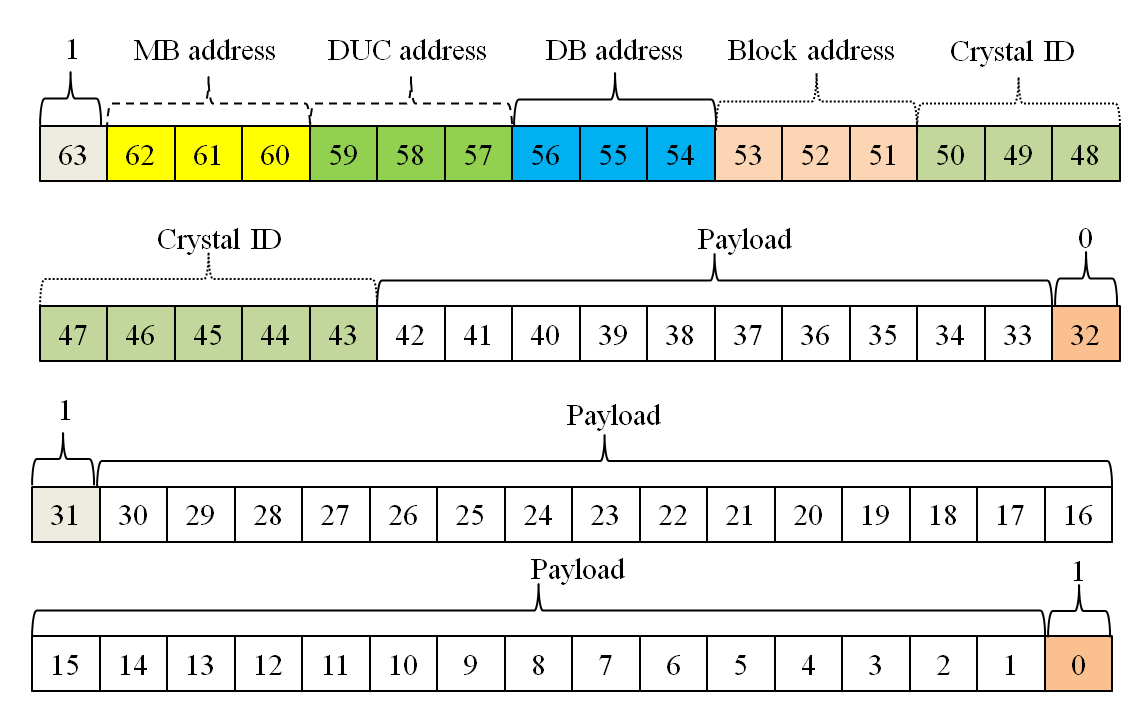
##### ***64-bit mode (not recommended)***

In the worst situation, the 32-bit single event data definition only allows 11 bits of payload data. To meet the requirements in applications which need large number of payload data bits, OpenPET standard also defines 64-bit single event data mode.

**Note that:**

1. In the 32-bit single event data mode, the interval between slice boundaries is 8 system clocks (80MHz). By contrast, the interval between slice boundaries is increased to 16 system clocks (80MHz) to enable transmission of extra 32 bits in the 64-bit single event data mode.
2. The 32-bit single event data mode meets the requirements for majority of OpenPET applications. We recommend NOT using 64-bit single event data mode to transmit data until inevitably necessary.

The following figure shows the format of the 64-bit single event data:



**Figure 23** The format of the 64-bit single event data.

The definitions of the 64 bits are very similar to that in the 32-bit format:

* **Bit 63**: Data valid bit. This bit has to be set to “1”. A parent node rejects any event data with a data valid bit set to “0”from its offspring nodes.
* **Bit 62 to 60**: 3 MB address bits. 3-bit MB addresses are assigned and added to the 32 bits event data in CUC. In a system without MB (compact configuration and simplified standard configuration), those 3 bits can be redefined as payload bits by the users.
* **Bit 59 to 57**: 3 DUC address bits. 3-bit DUC addresses are assigned and added to the 32 bits event data in MB (standard configuration) or CUC (simplified standard configuration). In a compact system, those 3 bits can also be redefined as payload bits by the users.
* **Bit 56 to 54**: 3 DB address bits. 3-bit DB addresses are assigned and added to the 32 bits event data in DUC or CDUC. Those 3 bits can NOT be redefined.
* **Bit 53 to 51**: 3 block detector address bits. 3-bit block detector addresses are assigned in the DB when a single event is detected. A DB has 32 channels, and therefore it supports up to 8 conventional block detector modules. The users can redefine those 3 bits if they do not use conventional block detector modules.
* **Bit 50 to 43**: 8 crystal ID address bits. 8-bit crystal ID addresses are assigned in the DB when a single event is detected. The maximum crystal matrix supported in this format is 16 x16. The users can redefine those 8 bits if they do not use conventional block detector modules.
* **Bit 42 to 33**: 10 bits of payload data. 10-bit payload data are generated in the DB when a single event is detected. The detailed definitions of those 10-bit payload data will be described in the next section.
* **Bit 32**: Flag bit for the first 32 bit of data. This bit has to be “0”.
* **Bit 31**: Data valid bit. This bit has to be set to “1” for the purposes of data synchronization and data file format decoding.
* **Bit 30 to 1**: 30 bits of payload data. 30-bit payload data are generated in the DB when a single event is detected. The detailed definitions of those 30-bit payload data will be described in the next section.
* **Bit 0**: Flag bit for the second 32 bit of data. This bit has to be “1”.

The 64-bit single event data format is mainly defined for users with special requirements in the payload data length. OpenPET define only one format for the 40 bits payload in the 64-bit single event data.

* **Standard 40-bit payload mode**

-- Bit 42~33, Bit 30~29 TDC data bits (12 bits, LSB: 25ps).

-- Bit 28~25, DOI (4 bits).

-- Bit 24~17, Energy value (8 bits).

-- Bit 16~9, Y value (8 bits).

-- Bit 8~1, X value (8 bits).

Similar to 32-bit single event data format, many bits defined in 64-bit single event data format (marked in blue) can be redefined by the users for payload data transmission in different system configurations and applications. The maximum payload data bits can be as much as 57 bits (compact configuration, not conventional block detector).

The three possible user-defined modes are listed as follows:

* **User-defined mode 1 –** 40 bits of payload, but the definition of the payload bits are not one of the five standard definitions above mentioned.
* **User-defined mode 2 –** 43 bits of payload. This mode is for simplified standard system or compact system that has no MB nodes, and the 3 bits of MB addresses are redefined for payload data.
* **User-defined mode 3 –** 46 bits of payload. This mode is for compact system that has no MB and DUC nodes, and the 3 bits of MB addresses and the 3 bits of DUC addresses are redefined for payload data.
* **User-defined mode 4 –** 51 bits of payload. 3 block detector address bits and 8 crystal ID address bits are redefined for payload data.
* **User-defined mode 5 –** 54 bits of payload. This mode is for simplified standard system or compact system that has no MB nodes, and the 3 bits of MB addresses, 3 block detector address bits and 8 crystal ID address bits are redefined for payload data.
* **User-defined mode 6 –** 57 bits of payload. This mode is for compact system that has no MB and DUC nodes, and the 3 bits of MB addresses, the 3 bits of DUC addresses, 3 block detector address bits and 8 crystal ID address bits are redefined for payload data.

#### CUC/CDUC🡪 host PC

##### ***Single event mode***

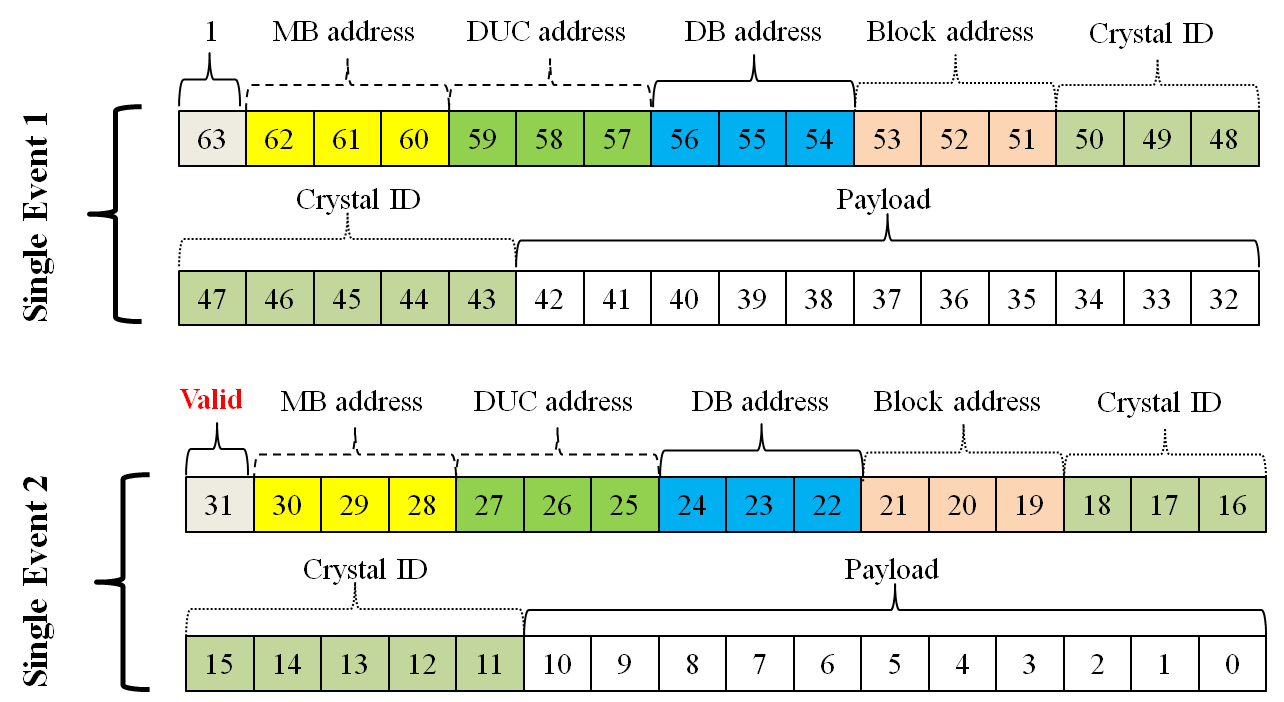
In single event data mode (including the 32-bit raw ADC data mode), CUC/CDUC node just assign the MB (DUC or DB, depends on the system configuration) addresses to the data, and directly send the data to host PC. Therefore the formats of the data from CUC/CDUC🡪 host PC are the exactly the same that those defined in previous section (DB🡪DUC, DUC🡪 MB and MB🡪CUC).

##### ***Coincidence event mode***

In coincidence event data mode, the CUC/CDUC node needs to combine two single event data to form a coincidence event data. In most situations, we almost directly combine two single event data to form the coincidence event data. For examples:

1. Two 32-bit single event data 🡪 one 64-bit coincidence event data

The following figure shows the format of the 64-bit coincidence event data.



**Figure 24** The format of the 64-bit coincidence event data.

Note that the only difference between the one 64-bit coincidence word and the original two 32-bit single event words is that:

* Bit 31: valid bit (originally always set to “1”) is now used to indicate valid coincidence data (“1”) or invalid coincidence data for random correction (“0”).

1. Two 64-bit single event data 🡪 one 128-bit coincidence event data

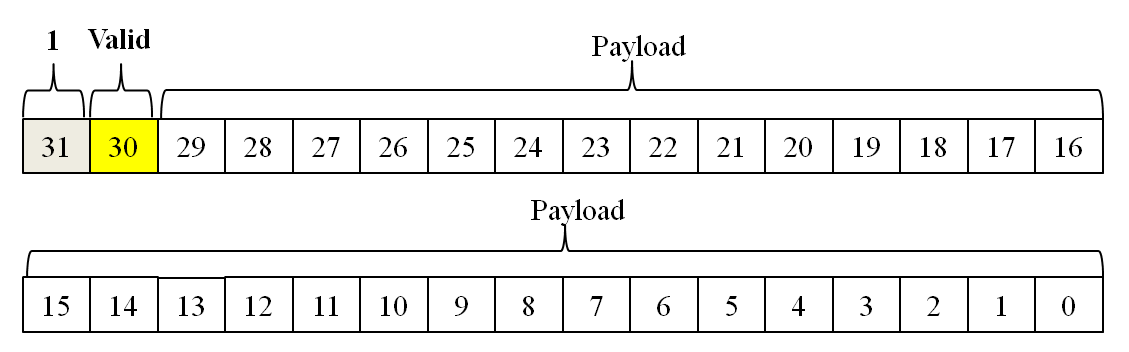
Similarly, the only difference between the one 128-bit coincidence word and the original two 64-bit single event words is that:

* **Bit 63**: Valid coincidence event bit (originally always set to “1”) is now used to indicate valid coincidence data (“1”) or invalid coincidence data for random correction (“0”).

In addition to the direct combination of two single event data to form a coincidence event data, we also allow the users to form shorter coincidence data:

1. Two 32-bit single event data 🡪 one 32-bit coincidence event data

The following figure shows the format of the 32-bit coincidence event data.



**Figure 25** The format of the 32-bit coincidence event data.

The definitions of the 32 bits are:

* **Bit 31**: Data valid bit. This bit has to be set to “1”.
* **Bit 30**: Valid coincidence event bit. This bit is used to indicate valid coincidence data (“1”) or invalid coincidence data for random correction (“0”).
* **Bit 29 to 0**: Payload bits (30 bits). In most situations, the payload bits contain two types of information: coincidence pair ID and time differences between the two single events (for time of flight system). The users can decide how to efficiently load the 30-bit payload data segment according to their detector and system designs.

1. Two 64-bit single event data 🡪 one 64-bit coincidence event data

Similarly, the definitions of the 64 bits are:

* **Bit 63**: Data valid bit. This bit has to be set to “1”.
* **Bit 62**: Valid coincidence event bit. This bit is used to indicate valid coincidence data (“1”) or invalid coincidence data for random correction (“0”).
* **Bit 61 to 0**: Payload bits (62 bits). In most situations, the payload bits contain two types of information: coincidence pair ID and time differences between the two single events (for time of flight system). The users can decide how to efficiently load the 62-bit payload data segment according to their detector and system designs.

## *Status words*

### *General information*

Status words are inserted to the data stream by the CUC/CDUC to transmit information other than the event data to the host PC.

There are many different types of status words that can be inserted to the data stream. In some situations, the users may want to insert some real time status words (highest priority) into the data stream. To avoid conflict (two status words want to be inserted to the data stream simultaneously), the following strategies are adapted when the status words circuits are implemented in the firmware design:

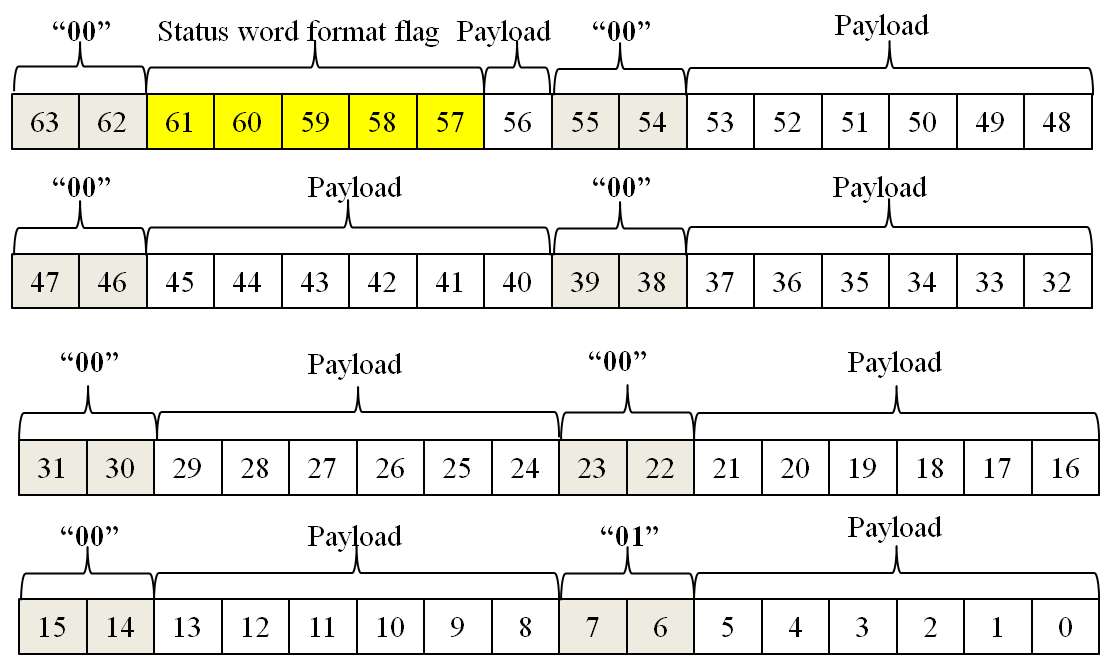
* Status words with fixed time intervals are allocated in different time points.
* The OpenPET system clock is 80MHz. The slice boundary clock is 10MHz (or 5MHz). So the maximum rate of status words is fixed to 5MHz (0.2µs time interval).
* Status words with fixed time intervals are only allowed to be inserted at the time points:

Format flag is a 5-bit “status word format flag” described below (up to 32 different types of status words can be defined in OpenPET).

* Real time status words with non-fixed time intervals can be inserted to the data stream at any time points. The time precision for insertion of those words is 0.2µs. Note that does NOT mean the time resolution of the information in those status words is limited to 0.2µs.

### *General definition of a status word*

The length of the status data is fixed to 64 bits. The following figure shows the general format of a 64-bit status word:



**Figure 26** The format of the 64-bit status word.

The definitions of the 64 bits are:

* **Bit 63 to 62**: Status word flag. Those two bits have to be set to “00”.
* **Bit 61 to 57**: Status word format flag (5 bits). Up to 32 different types of status words can be defined in OpenPET.

– “00000” Event data format word

* **Bit 56**: payload data (1 bit).
* **Bit 55 to 54**: Status word flag. Those two bits have to be set to “00”.
* **Bit 54 to 48**: payload data (7 bits).
* **Bit 47 to 46**: Status word flag. Those two bits have to be set to “00”.
* **Bit 45 to 40**: payload data (6 bits).
* **Bit 39 to 38**: Status word flag. Those two bits have to be set to “00”.
* **Bit 37 to 32**: payload data (6 bits).
* **Bit 31 to 30**: Status word flag. Those two bits have to be set to “00”.
* **Bit 29 to 24**: payload data (6 bits).
* **Bit 23 to 22**: Status word flag. Those two bits have to be set to “00”.
* **Bit 21 to 16**: payload data (6 bits).
* **Bit 15 to 14**: Status word flag. Those two bits have to be set to “00”.
* **Bit 13 to 8**: payload data (6 bits).
* **Bit 7 to 6**: Status word flag. Those two bits have to be set to “01”.
* **Bit 5 to 0**: payload data (6 bits).

There are 43-bit of payload data in total. Detailed definitions of those 43 bits are described in the next section.

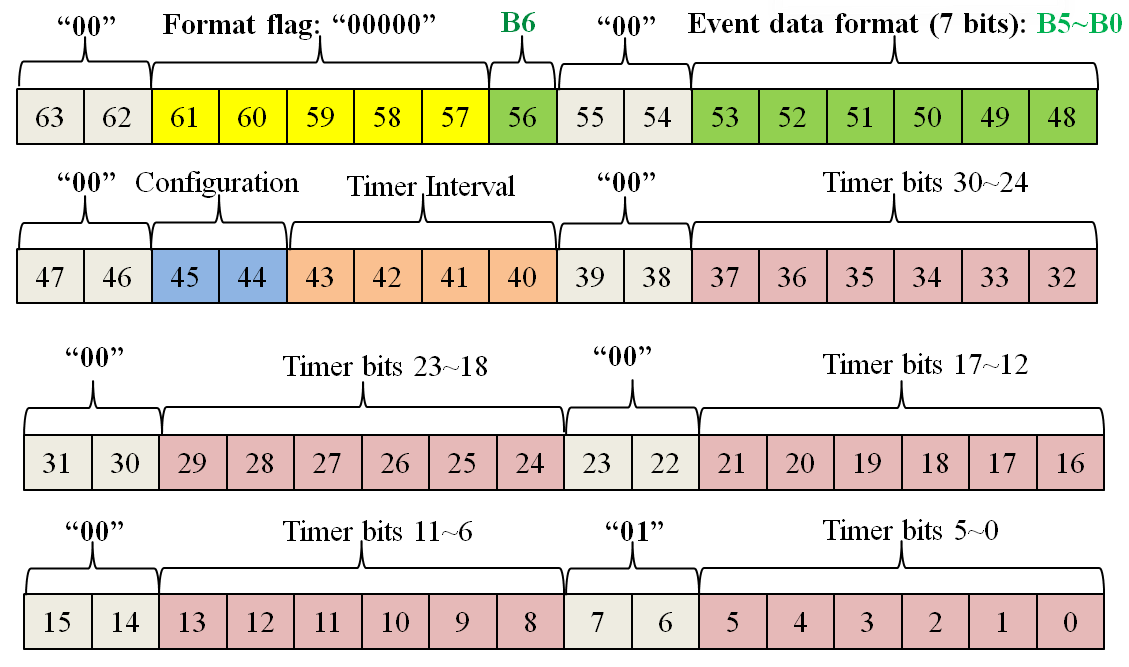
### *OpenPET standard status words*

OpenPET protocol defines three standard status words: event data format/timer word (00000), coincidence event rate word (00001) and single event rate word (00010).

The definitions of the 43-bit payload data for those three words are as follows:

#### Event data format/timer word (format flag:00000 )

The “event data format/timer word” is inserted to the event data stream every given time interval (1ms, 5ms, 10ms, 20ms, 50ms, 100ms, 200ms, 500ms, 1s, 2s, 5s, 10s, 20s, 60s, or a user-defined time interval, default: 1ms). **The “event data format/timer word” is a mandatory word that has the highest priority. OpenPET still allows the users to disable the “event data format/timer word”. However, it is NOT recommended for data integrity reasons described in next section.**



**Figure 27** The format of event data format/timer word.

The definitions of the 64 bits are:

* **Bit 63 to 62**: Status word flag. Those two bits have to be set to “00”.
* **Bit 61 to 57**: Status word format flag bits (5 bits). – “00000”
* **Bit 56, 53 to 48:** Event data format bits (7 bits). The definitions of many event data formats are described in previous sections. The definitions of the 7 event data format bits are described in the next sections.
* **Bit 55 to 54, 47 to 46**: Status word flag. Those two bits have to be set to “00”.
* **Bit 45 to 44**: System configuration flag bits (2 bits).

– “00” Standard system configuration

– “01” Simplified standard system configuration

– “10” Compact system configuration

– “11” Not defined

* **Bit 43 to 40**: Timer interval setting (4 bits).

– “0000” Generate an event data format/timer word every 1ms (default)

– “0001” Generate an event data format/timer word every 5ms

– “0010” Generate an event data format/timer word every 10ms

– “0011” Generate an event data format/timer word every 20ms

– “0100” Generate an event data format/timer word every 50ms

– “0101” Generate an event data format/timer word every 100ms

– “0110” Generate an event data format/timer word every 200ms

– “0111” Generate an event data format/timer word every 500ms

– “1000” Generate an event data format/timer word every 1s

– “1001” Generate an event data format/timer word every 2s

– “1010” Generate an event data format/timer word every 5s

– “1011” Generate an event data format/timer word every 10s

– “1100” Generate an event data format/timer word every 20s

– “1101” Generate an event data format/timer word every 60s

– “1110” Generate an event data format/timer word every user-defined interval

– “1111” Disable event data format/timer word

* **Bit 39 to 38**: Status word flag. Those two bits have to be set to “00”.
* **Bit 37 to 32**: Timer bit 29~24 (6 bits). Note that:

– This is a 30 bits timer

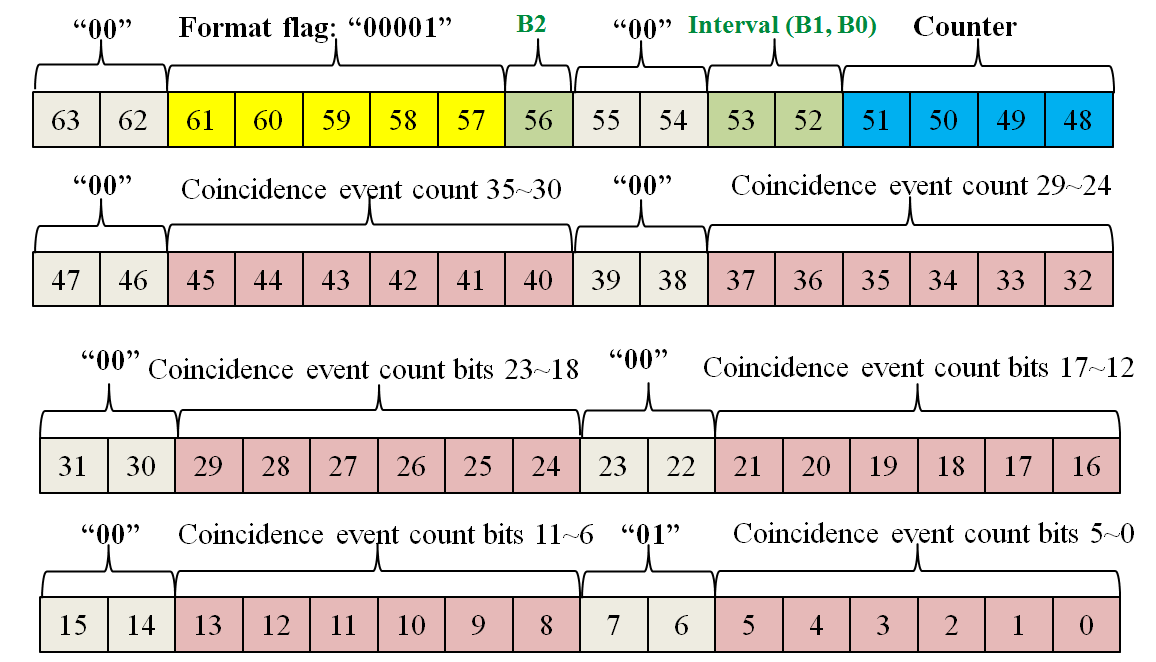
– The LSB is 1ms.

* **Bit 31 to 30**: Status word flag. Those two bits have to be set to “00”.
* **Bit 29 to 24**: Timer bit 23~18 (6 bits).
* **Bit 23 to 22**: Status word flag. Those two bits have to be set to “00”.
* **Bit 21 to 16**: Timer bit 17~12 (6 bits).
* **Bit 15 to 14**: Status word flag. Those two bits have to be set to “00”.
* **Bit 13 to 8**: Timer bit 11~6 (6 bits).
* **Bit 7 to 6**: Status word flag. Those two bits have to be set to “01”.
* **Bit 5 to 0**: Timer bit 5~0 (6 bits).

In the “Event data format/timer word”, bits 56, 53 to 48 are used to (7 bits) Event data format bits. The definitions of many event data format are described in previous sections. The definitions 7 event data format bits are described in next sections.

#### Coincidence Event rate word (format flag: 00001)

The “coincidence event rate word” is inserted to the event data stream every given time interval (1s, 1ms, 10ms, 100ms, 10s, 60s, or a user-defined time interval, default: 1s).



**Figure 28** The format of Coincidence Event rate word.

The definitions of the 64 bits are:

* **Bit 63 to 62**: Status word flag. Those two bits have to be set to “00”.
* **Bit 61 to 57**: Status word format flag bits (5 bits). – “00001”
* **Bit 56, 53 to 52:** interval setting (3 bits).

– “000” Generate a coincidence event rate word every 1s (default)

– “001” Generate a coincidence event rate word every 1ms

– “010” Generate a coincidence event rate word every 10ms

– “011” Generate a coincidence event rate word every 100ms

– “100” Generate a coincidence event rate word every 10s

– “101” Generate a coincidence event rate word every 60s

– “110” Generate a coincidence event rate word every user-defined interval

– “111” Disable coincidence event rate word

* **Bit 55 to 54, 47 to 46**: Status word flag. Those two bits have to be set to “00”.
* **Bit 51 to 48**: 4-bit counter. The value of the counter increases 1 every time a coincidence event rate word is inserted to the data stream. The users can track this counter to check if there are missed coincidence event rate words.
* **Bit 45 to 40**: Coincidence event counter bit 35~30 (6 bits). Note that:

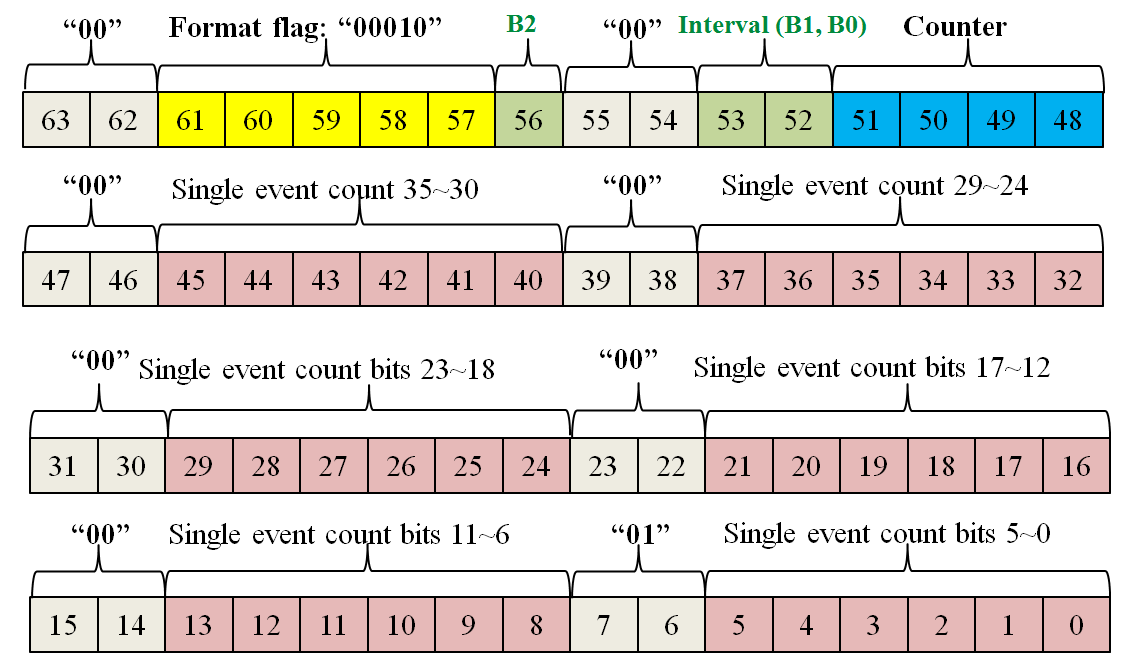
– This is a 36 bits timer

– It can count up to 64G events.

* **Bit 39 to 38**: Status word flag. Those two bits have to be set to “00”.
* **Bit 37 to 32**: Coincidence event counter bit 29~24 (6 bits).
* **Bit 31 to 30**: Status word flag. Those two bits have to be set to “00”.
* **Bit 29 to 24**: Coincidence event counter bit 23~18 (6 bits).
* **Bit 23 to 22**: Status word flag. Those two bits have to be set to “00”.
* **Bit 21 to 16**: Coincidence event counter bit 17~12 (6 bits).
* **Bit 15 to 14**: Status word flag. Those two bits have to be set to “00”.
* **Bit 13 to 8**: Coincidence event counter bit 11~6 (6 bits).
* **Bit 7 to 6**: Status word flag. Those two bits have to be set to “01”.
* **Bit 5 to 0**: Coincidence event counter bit 5~0 (6 bits).

#### Single Event rate word (format flag: 00010)

Similarly, the “single event rate word” is inserted to the event data stream every given time interval (1s, 1ms, 10ms, 100ms, 10s, 60s, or a user-defined time interval, default: 1s). Note that the single event count here is the total single event count of the whole system.



**Figure 29** The format of single event rate word.

The definitions of the 64 bits are:

* **Bit 63 to 62**: Status word flag. Those two bits have to be set to “00”.
* **Bit 61 to 57**: Status word format flag bits (5 bits). – “00010”
* **Bit 56, 53 to 52:** interval setting (3 bits).

– “000” Generate a single event rate word every 1s (default)

– “001” Generate a single event rate word every 1ms

– “010” Generate a single event rate word every 10ms

– “011” Generate a single event rate word every 100ms

– “100” Generate a single event rate word every 10s

– “101” Generate a single event rate word every 60s

– “110” Generate a single event rate word every user-defined interval

– “111” Disable single event rate word

* **Bit 55 to 54, 47 to 46**: Status word flag. Those two bits have to be set to “00”.
* **Bit 51 to 48**: 4-bit counter. The value of the counter increases 1 every time a single event rate word is inserted to the data stream. The users can track this counter to check if there are missed single event rate words.
* **Bit 45 to 40**: single event counter bit 35~30 (6 bits). Note that:

– This is a 36 bits timer

– It can count up to 64G events.

* **Bit 39 to 38**: Status word flag. Those two bits have to be set to “00”.
* **Bit 37 to 32**: Single event counter bit 29~24 (6 bits).
* **Bit 31 to 30**: Status word flag. Those two bits have to be set to “00”.
* **Bit 29 to 24**: Single event counter bit 23~18 (6 bits).
* **Bit 23 to 22**: Status word flag. Those two bits have to be set to “00”.
* **Bit 21 to 16**: Single event counter bit 17~12 (6 bits).
* **Bit 15 to 14**: Status word flag. Those two bits have to be set to “00”.
* **Bit 13 to 8**: Single event counter bit 11~6 (6 bits).
* **Bit 7 to 6**: Status word flag. Those two bits have to be set to “01”.
* **Bit 5 to 0**: Single event counter bit 5~0 (6 bits).

### *Reserved status words and user-defined status words*

OpenPET protocol reserves status word flag from 00000 to 10111 (24 words) for standard status words. So far, we only define 3 standard status words: event data format/timer word (00000), coincidence event rate word (00001) and single event rate word (00010).

The users can define their own status words (8 words at most) with status word flags that range from 11000 to 11111.

## *Coding and decoding of list mode data*

### *Redundancy and integrity of list mode data*

In the high speed list mode data transmission, there could be four types of errors:

1. Type 1: the values of one or several bits of data are changed by mistake.

* Those errors are not very likely in an OpenPET system with optimized system timing and communication protocol designs.
* The test mode event data formats are designed to detect those kinds of errors.
* Therefore, when those errors happen (not very likely), their effects are local. They won’t affect the integrity of the whole data file.

1. Type 2: the values of one or several bytes of data are changed by mistake.

Similarly to type 1, those errors are not very likely and can be detected using test mode event data.

1. Type 3: Several bits of data are dropped or added in the data stream. In the OpenPET hardware and firmware designs, the list mode data (including event words data and status words data) are organized and transmitted by byte. Therefore,

* The possibilities of missing or adding some **bits** of data in the data stream are very low. It is more likely to miss or add some **bytes** of data (discussed next).
* Even if those errors happen, they will be limited to only several bytes. They won’t misalign the whole data stream or affect the integrity of the whole data file.

1. Type 4: Several bytes of data are dropped or added in the data stream.

* This is the most possible error.
* It may misalign the whole data stream and lead to mistakes in data analysis.

Except for improving the system hardware and firmware design, there are two encoding strategies to eliminate the type 4 errors.

1. Strategy 1: Use segments of data pattern in the event and status words to match data.

* For example, one can fix a data pattern of “1010101010” in bit 7~0 of the 32-bit event data. The data analysis software can then check and match “1010101010” in the data file when loading event data.
* The main disadvantage of this method is that the data patterns occupy data payload bits. That will lead to a big decrease of payload data bits or a decrease in maximum event rates.

1. Strategy 2: Use a status word (“Event data format/timer word”) to align data.

OpenPET adapt strategy 2 to align data. The main features of this strategy are:

* Use minimum pattern bits to separate event words and status word
* The MSB of event words (32-bit, 64-bit or 128-bit) is always “1”. More details:

In 32-bit (or 4-byte) mode, bit 31=“1”;

In 64-bit (or 8-byte) mode, bit 63=“1”;

In 128-bit (or 16-byte) mode, bit 127 = “1”, bit 63 = “1”,

* The MSB of status words (64-bit, or 8 bytes) is always “0”. More details:

Bit 7 and Bit 6 of byte 7 to byte 1: “00”

Bit 7 and Bit 6 of byte 0: “01”

* Therefore, we can confidently treat 8 bytes of data as a status word when

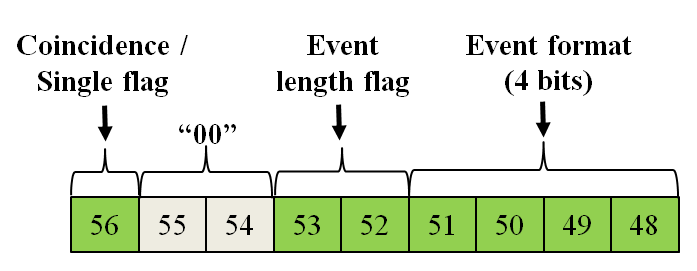
Bit 7 and Bit 6 of byte 7 to byte 1: “00”

Bit 7 and Bit 6 of byte 0: “01”

* Use a status word (“Event data format/timer word”) to determine the event data format and align event data.
* The main advantage is that it achieves a high payload rate in the event data word. More information can be loaded in the event data word. Alternatively, a higher event rate can be achieved.
* The main disadvantage is that the misaligned event data can only be realigned after a new “event data format/timer word” is detected. The minimum interval of “event data format/timer word” is 1ms. That means 1ms of event data might be lost in the worst situation. Fortunately, that is not an issue for most applications.

### *Summary of event data formats*

In “event data format/timer word”, bit 56 and bit 53 to 48 (7 bits) are used to define event data. The detailed definitions of those 7 bits are shown as follows:



**Figure 30** In “event data format/timer word”, bit 56 and bit 53 to 48 (7 bits) are used to define event data..

* **Bit 56**: Coincidence/single event flag.

– “0” Single event data

– “1” Coincidence event data

* **Bit 53 to 52**: Event length flag.

– The definitions of these two bits depend on the value of Bit 56.

– When Bit 56=“0” (Single event data), definitions of bit 53 and bit 52 are:

“00” 32-bit (to CUC/CDUC), 32-bit (from CUC/CDUC to host PC)

“01” 32-bit raw ADC data mode

“10” 64-bit (to CUC/CDUC), 64-bit (from CUC/CDUC to host PC)

“11” Not defined

– When Bit 56=“1” (Coincidence event data), definitions of bit 53 and bit 52 are:

“00” 32-bit (to CUC/CDUC), 32-bit (from CUC/CDUC to host PC)

“01” 32-bit (to CUC/CDUC), 64-bit (from CUC/CDUC to host PC)

“10” 64-bit (to CUC/CDUC), 64-bit (from CUC/CDUC to host PC)

“11” 64-bit (to CUC/CDUC), 128-bit (from CUC/CDUC to host PC)

* **Bit 51 to 48**: Event format flag (4 bits).

– The definitions of these 4 bits depend on the value of Bit 56 and Bit 53 to 52.

– When Bit 56=“0”, Bit 53-52=“00” (Single event, 32🡪32), definitions of bit 51 to 58:

“0000”: **Time mode** (refer to section 3.2.3.1.1 for details of event data format)

“0001”: **Energy mode** (refer to section 3.2.3.1.1 for details of event data format)

“0010”: **Test mode1** (refer to section 3.2.3.1.1 for details of event data format)

“0011”: **Test mode2** (refer to section 3.2.3.1.1 for details of event data format)

“0100”: **Anger-logic flood map plot mode** (refer to section 3.2.3.1.1 for details)

“0101”: **Reserved single mode**

“0110”: **Reserved single mode**

“0111”: **Reserved single mode**

“1000”: **Reserved single mode**

“1001”: **User-defined single mode 1** (refer to section 3.2.3.1.1 for details)

“1010”: **User-defined single mode 2** (refer to section 3.2.3.1.1 for details)

“1011”: **User-defined single mode 3** (refer to section 3.2.3.1.1 for details)

“1100”: **User-defined single mode 4** (refer to section 3.2.3.1.1 for details)

“1101”: **User-defined single mode 5** (refer to section 3.2.3.1.1 for details)

“1110”: **User-defined single mode 6** (refer to section 3.2.3.1.1 for details)

“1111”: **Reserved single mode**

– When Bit 56=“0”, Bit 53-52=“01” (32-bit raw ADC data mode), definitions of bit 51 to 58:

“0000”: **Single channel mode** (refer to section 3.2.3.1.2 for details)

“0001”: **Four-channel mode** (refer to section 3.2.3.1.2 for details)

“0010”: **Sixteen-channel mode** (refer to section 3.2.3.1.2 for details)

“0011”: **Thirty-two-channel mode** (refer to section 3.2.3.1.2 for details)

“0100” ~“1000”: **Reserved raw ADC modes** (refer to section 3.2.3.1.2 for details)

“1000” ~“1111”: **User-defined raw ADC modes** (refer to section 3.2.3.1.2 for details)

– When Bit 56=“0”, Bit 53-52=“10” (Single event, 64🡪64), definitions of bit 51 to 58:

“0000”: **Standard 40-bit payload mode** (refer to section 3.2.3.1.3 for details)

“0001” ~“1000”: **Reserved single modes**

“1001”: **User-defined single mode 1** (refer to section 3.2.3.1.3 for details)

“1010”: **User-defined single mode 2** (refer to section 3.2.3.1.3 for details)

“1011”: **User-defined single mode 3** (refer to section 3.2.3.1.3 for details)

“1100”: **User-defined single mode 4** (refer to section 3.2.3.1.3 for details)

“1101”: **User-defined single mode 5** (refer to section 3.2.3.1.3 for details)

“1110”: **User-defined single mode 6** (refer to section 3.2.3.1.3 for details)

“1111”: **Reserved single mode**

– When Bit 56=“1”, Bit 53-52=“00” (Coincidence, 32🡪32), definitions of bit 51 to 58:

“0000” ~“1111”: **To be defined (**refer to section 3.2.3.2.2 (3) for details**)**

– When Bit 56=“1”, Bit 53-52=“01” (Coincidence, 32🡪64), definitions of bit 51 to 58:

“0000”: **Default Coincidence** **mode** (refer to section 3.2.3.1.1 and 3.2.3.2.2 (1))

“0001”: **Reserved Coincidence** **mode**

“0010”: **Test Coincidence** **mode1** (refer to section 3.2.3.1.1 and 3.2.3.2.2 (1))

“0011”: **Test Coincidence** **mode2** (refer to section 3.2.3.1.1 and 3.2.3.2.2 (1))

“0100”: **Reserved Coincidence** **mode**

“0101”: **Reserved Coincidence** **mode**

“0110”: **Reserved Coincidence** **mode**

“0111”: **Reserved Coincidence** **mode**

“1000”: **Reserved Coincidence** **mode**

“1001”: **User-defined Coincidence** **mode 1** (refer to section 3.2.3.1.1 and 3.2.3.2.2 (1))

“1010”: **User-defined Coincidence** **mode 2** (refer to section 3.2.3.1.1 and 3.2.3.2.2 (1))

“1011”: **User-defined Coincidence** **mode 3** (refer to section 3.2.3.1.1 and 3.2.3.2.2 (1))

“1100”: **User-defined Coincidence mode 4** (refer to section 3.2.3.1.1 and 3.2.3.2.2 (1))

“1101”: **User-defined Coincidence** **mode 5** (refer to section 3.2.3.1.1 and 3.2.3.2.2 (1))

“1110”: **User-defined Coincidence** **mode 6** (refer to section 3.2.3.1.1 and 3.2.3.2.2 (1))

“1111”: **Reserved Coincidence** **mode**

– When Bit 56=“1”, Bit 53-52=“10” (Coincidence, 64🡪64), definitions of bit 51 to 58:

“0000” ~“1111”: **To be defined (**refer to section 3.2.3.2.2 (4) for details**)**

– When Bit 56=“1”, Bit 53-52=“11” (Coincidence, 64🡪128), definitions of bit 51 to 58:

“0000”: **Default Coincidence mode** (refer to section 3.2.3.1.3 for details)

“0001” ~“1000”: **Reserved Coincidence modes**

“1001”: **User-defined Coincidence mode 1** (refer to section 3.2.3.1.3 and 3.2.3.2.2 (2))

“1010”: **User-defined Coincidence mode 2** (refer to section 3.2.3.1.3 and 3.2.3.2.2 (2))

“1011”: **User-defined Coincidence mode 3** (refer to section 3.2.3.1.3 and 3.2.3.2.2 (2))

“1100”: **User-defined Coincidence mode 4** (refer to section 3.2.3.1.3 and 3.2.3.2.2 (2))

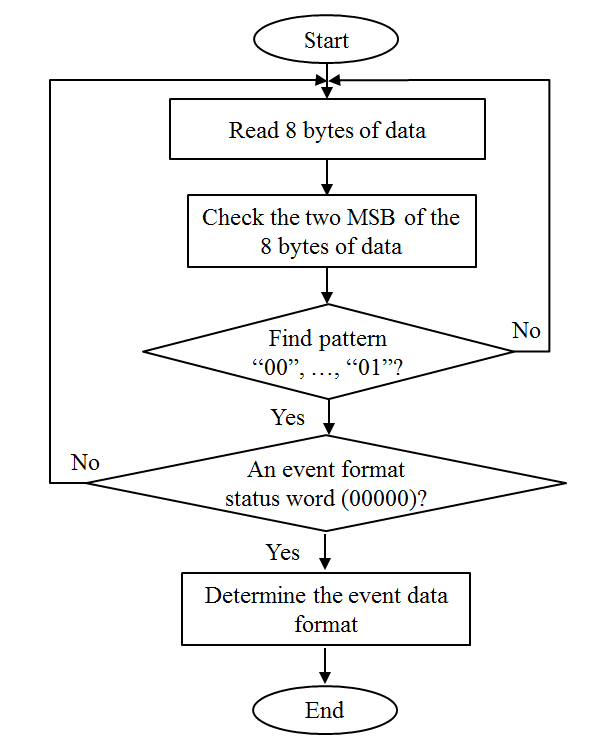
“1101”: **User-defined Coincidence mode 5** (refer to section 3.2.3.1.3 and 3.2.3.2.2 (2))

“1110”: **User-defined Coincidence mode 6** (refer to section 3.2.3.1.3 and 3.2.3.2.2 (2))

“1111”: **Reserved Coincidence mode**

### *Algorithm to decode list mode data file*

The flowchart to decode a list mode data file is shown as follows:



**Figure 31** flowchart to decode a list mode data file.

# System software framework models

As described in previous section, there are four types of software codes in the OpenPET system. One is the LabWindow/CVI C code run in the host computer. Three are C codes run in the NIOS II μ-processor of the CUC, MB and DUC.

## *Host computer software*

Operation system of the host computer is non-real-time OS Window 7.

### *Host computer software functions*

Generally speaking, the functions of the host computer are: system configuration, calibration and monitoring, data acquisition and data analysis. We have described system configuration, calibration and monitoring, in previous sections (command/response section). Here we list the data analysis needed to be done by the host computer software.

(1) Single events data analysis

a. Addressing analysis

-DUC/MB/DB/CH address mapping

-Individual crystal ID address mapping (Flood map and Crystal ID lookup table)

b. Energy data analysis

- ADC channel data analysis

- Energy histogram analysis (energy resolution, energy window and etc.)

c. Time data analysis

- TDC channel data analysis

- Time histogram analysis (time resolution, time delay correction and etc.)

d. Test mode data analysis (

- data transmission integrity analysis

(2) Coincidence events data analysis

a. Coincidence pair addressing analysis

b. Coincidence event analysis

- Sinogram

- Random correction and etc.

(3) Status words data analysis

a. Time word analysis

b. Event rate analysis

c. Temperature and voltage monitoring

d. User-defined status processing

<more data analysis functions will be added to this list>

### *Host computer software model*

Figure 12 shows that framework model for the host computer software. Note that the data acquisition and data analysis can be implemented as independent modules. And they are already discussed in previous sections. So they are not included in this software model.

The model has four layers:

1. OpenPET Application Layer (OAL)

Top level System configuration, monitor and management (functional configuration):

* System level addressing,
* Flood map and crystal decoding,
* Energy calibration (including analog gain setting and energy window configuration)
* Time calibration (including TDC calibration and individual channel time delay calibration)
* Coincidence pair settings
* Other top level functions for user defined applications

1. OpenPET System Configuration Layer (OSCL)

Main functions are: (hardware board oriented configuration)

* Configuration command addressing and process (for CUC, MB, DUC and DB )
* Event data analysis and process (for single event data, coincidence data and status data)
* CUC, MB, DUC and DB addressing and configuration control;
* CUC, MB, DUC and DB hardware test, monitor, and management;
* CUC, MB, DUC and DB firmware and software configuration and management;
* CUC, MB, DUC and DB data file (configuration database for lookup table and etc.) configuration and management.

1. Communication Support Layer (CSL)

Main functions are:

* Event data communication protocol
* Status/Command communication protocol

1. Hardware Abstract Layer (HAL)

Low level hardware drivers for:

* High speed disk access
* USB
* Gb Ethernet (optional)
* Optic fiber cable (optional)

\* Note: The differences between OAL configuration and OSCL configuration.

* OAL configuration: functional configuration.
* OSCL configuration: hardware board oriented configuration



**Figure 32** The framework model for the host computer software.

## *CUC/MB/DUC/CDUC software*

The framework model for the CUC/MB/DUC/CDUC software is shown in Figure 13. The model has three layers:

1. OpenPET Configuration Layer (OCL)

* Configuration command addressing and process (for CUC/MB/DUC/DB )
* Generate status event data (CUC only)
* CUC/MB/DUC/ DB addressing and configuration control;
* CUC/MB/DUC/ DB hardware test, monitor, and management;
* CUC/MB/DUC/ DB firmware and software configuration and management;
* CUC/MB/DUC/ DB data file (configuration database for lookup table and etc.) configuration and management.

1. Application Support Layer (ASL)

Main functions are:

* Status/Command communication protocol
* Devices (DIO, Memory) Access Control
* FAT32 file system

1. Hardware Abstract Layer (HAL)

C standard library and low level hardware drivers for:

* Digital IO
* Memory
* SD card
* Miscellaneous devices



**Figure 33** The framework model for the CUC/MB/DUC/CDUC software.

\* Note:

In the host computer software, the OSCL configures and manages all CUC, MB, DUC and DB hardware boards.

In the CUC software, OCL configures and manages CUC only.

In the MB software, OCL configures and manages MB only.

In the DUC software, OCL configures and manages DUC and all DBs plugged into that DUC.

Operation system of the CUC/MB/DUC/CDUC is real-time NIOS II RTOS. The following diagram demonstrates how the CUC/MB/DUC/CDUC software works.

## *An example (how host computer and CUC/MB/DUC/CDUC NIOS work together)*

Here is an example show how the host computer and CUC/MB/DUC/CDUC NIOS work together to setup the energy window for an analog channel:

1. In host computer sends the command to the CUC

* We call an OAL function to configure the energy window (e.g. low energy cut: 150, high energy cut: 200) for the analog data channel connected to MB2/DUC3/DB4/CH5.
* The OAL function will call a lower level OCL function to set the SRAM in MB2/DUC3/DB4/SRAM address 5, which are defined for the energy window thresholds setting.
* The OCL function will then call HAL function to load the SRAM address (MB2/DUC3/DB4/SRAM address 5) and data (150, 200) into a command data package as follows:



**Figure 34** The command message package, an example.

* The CSL function then calls an ASL function to load the data package on the USB bus, sets up a timer interrupt and waits for responses;
* If the host computer doesn’t receive a response before the timer is time out, the command fails. Otherwise, goes to step (9) (Host computer receives and processes the response from CUC).

1. CUC receives and processes the command from host computer

* The command from the USB bus causes an interruptive event in CUC.
* If there is no other event processing going on, or the new command event has the highest priority than all the other command events, an OCL function will be called to handle the new command event.
* The OCL function calls an ASL function to analyze the command event.
* The ASL function calls a HAL function to fetch the command package from the command buffer or registers, decodes the CMD ID and target address and return them to the OCL function.
* The OCL function analyzes the CMD ID and target address, and finds that the destination of the command is the DB board.
* Therefore, another OCL function is called to forward the command to the right MB.
* The OCL function call an ASL function, and the ASL function call an HAL function to send the command to MB2 through the status/command communication interface between CUC and MB2.

1. MB2 receives and processes the command from CUC

* The command from status/command communication interface causes an interruptive event in MB2.
* If there is no other event processing going on, or the new command event has the highest priority than all the other command events, an OCL function will be called to handle the new command event.
* The OCL function calls an ASL function to analyze the command event.
* The ASL function calls a HAL function to fetch the command package from the command buffer or registers, decodes the CMD ID and target address and return them to the OCL function.
* The OCL function analyzes the CMD ID and target address, and finds that the destination of the command is the DB board.
* Therefore, another OCL function is called to forward the command to the right DUC.
* The OCL function call an ASL function, and the ASL function call an HAL function to send the command to DUC3 through the status/command communication interface between MB2 and DUC3.

1. DUC3 receives and processes the command from MB2

* The command from status/command communication interface causes an interruptive event in DUC3.
* If there is no other event processing going on, or the new command event has the highest priority than all the other command events, an OCL function will be called to handle the new command event.
* The OCL function calls an ASL function to analyze the command event.
* The ASL function calls a HAL function to fetch the command package from the command buffer or registers, decodes the CMD ID and target address and return them to the OCL function.
* The OCL function analyzes the CMD ID and target address, and finds that the destination of the command is the DB board.
* Therefore, another OCL function is called to forward the command to a right DB.
* The OCL function call an ASL function, and the ASL function call an HAL function to send the command to DB4 through the status/command communication interface between DUC3 and DB4.

1. DB4 receives and processes the command from DUC3

There is no NIOS II CPU in the DB. The FPGA program accepts the command, decodes it, performs it (write 150 and 200 to the SDRAM address 5) and sends a response back to DUC3 through the status/command communication interface between DUC3 and DB4.



**Figure 35** The response message package, an example.

1. DUC3 receives and processes the response from DB4

* The response from status/command communication interface causes an interruptive event in DUC3.
* If there is no other event processing going on, or the new command/response event has the highest priority than all the other command/response events, an OCL function will be called to handle the new response event.
* The OCL function calls an ASL function to analyze the response event.
* The ASL function calls a HAL function to fetch the response package from the command/ response buffer or registers, decodes the Response ID and target address and return them to the OCL function.
* The OCL function analyzes the Response ID and target address, and finds that the destination of the Response is not DUC3.
* Therefore, another OCL function is called to forward the response to its parent node MB2.
* The OCL function calls an ASL function, and the ASL function call an HAL function to send the response to MB2 through the status/command communication interface between DUC3 and MB2.

1. MB2 receives and processes the response from DUC3

* The response from status/command communication interface causes an interruptive event in MB2.
* If there is no other event processing going on, or the new command/response event has the highest priority than all the other command/response events, an OCL function will be called to handle the new response event.
* The OCL function calls an ASL function to analyze the response event.
* The ASL function calls a HAL function to fetch the response package from the command/ response buffer or registers, decodes the Response ID and target address and return them to the OSC function.
* The OCL function analyzes the Response ID and target address, and finds that the destination of the Response is not MB2.
* Therefore, another OCL function is called to forward the response to its parent node CUC.
* The OCL function calls an ASL function, and the ASL function call an HAL function to send the response to CUC through the status/command communication interface between CUC and MB2.

1. CUC receives and processes the response from MB2

* The response from status/command communication interface causes an interruptive event in CUC.
* If there is no other event processing going on, or the new command/response event has the highest priority than all the other command/response events, an OCL function will be called to handle the new response event.
* The OSC function calls an ASL function to analyze the response event.
* The ASL function calls a HAL function to fetch the response package from the command/ response buffer or registers, decodes the Response ID and target address and return them to the OCL function.
* The OCL function analyzes the Response ID and target address, and finds that the destination of the Response is not CUC.
* Therefore, another OCL function is called to forward the response to host PC.
* The OCL function calls an ASL function, and the ASL function call an HAL function to send the response to host computer through USB bus.

1. Host computer receives and processes the response from CUC

* The response from USB cable causes an interruptive event in host PC.
* If there is no other event processing going on, or the new command/response event has the highest priority than all the other command/response events, an OAL function will be called to handle the new response event.
* The OAL function calls an OCL function to analyze the response event.
* The OCL function calls an CSL function to analyze the response event.
* The CSL function calls a HAL function to fetch the response package from the command/ response buffer or registers, decodes the Response ID, source address and target address and return them to the OCL function.
* The OCL function analyzes the Response ID, source address and target address, and finds that the destination of the Response is host computer.
* Therefore, another OCL function returns the results of energy window setting (success or failed) to the OAL function.

# System firmware (FPGA) framework models

## *CUC/MB/DUC/CDUC firmware (FPGA) framework model*

The CUC/MB/DUC/CDUC firmware (FPGA) framework model is shown as Figure 14.



**Figure 36** CUC/MB/DUC/CDUC firmware (FPGA) framework model.

The CUC/MB/DUC/CDUC firmware framework model has 5 modules:

1. NIOS CPU core.

Its main functions are board configuration and board status monitoring.

1. Interface register array

Its main functions are:

1. Interface to PET function logic, hardware interface logic and DB command coprocessor.
2. Board firmware configuration and version control.

* NIOS CPU is able to determine the board firmware configuration and version by reading the configuration and version control registers;
* We can also implement or remove modules in the FPGA firmware by setting/resetting board firmware configuration and version control registers. In another word, similar to the compiler switches in c code, the Board firmware configuration and version control registers can determine which firmware module will be implemented.

1. PET function logic circuits

* PET function logic circuits are the logic circuits specifically designed for PET application. For example, in the CUC, the PET function logic circuits need to process coincidence events.
* NIOS CPU will be able to configure and monitor the PET function logic circuits through the Interface register array module.

1. Hardware interface logic circuits

* NIOS CPU and PET function logic circuits need to share the low level hardware resources.
* PET function logic circuits have higher priority to access to the low level hardware resources.

For example, when the NIOS CPU needs to read the DDRII RAM, it will firstly send a command to the interface register array. The interface register array will then initialize a read operation in the DDRII RAM module of the interface logic circuits. If the DDRII RAM module is idle, it will start the read operation immediately, and send the data back to the interface register array. So the CPU can read the RAM value from the registers. If the DDRII RAM module is busy at handling R/W request from the PET function logic circuits, it will return a busy response to the interface logic circuits. So the CPU knows the read operation is failed.

Note: in our current FPGA codes for board test, the NIOS CPU is interfaced to the low level hardware circuits directly. It works well for board diagnosis. However, when a PET system is in data acquisition mode, the high speed data stream is processed in timely pipeline mode. The PET function logic circuits need to have full control of the hardware resources (e.g. RAM). Therefore, we will not let the CPU control the hardware resources directly in our framework design.

1. DB command coprocessor (DUC only)

NIOS CPU takes a big chunk of the resource in the FPGA. That is not a big issue for CUC/MB/DUC/CDUC for two reasons: (a) Those boards have three FPGAs on board and (b) The tasks need to be performed in CUC/MB/DUC/CDUC are relatively light.

For DB boards, the situation is a little bit different. Firstly, the DB board has only one FPGA on board. Secondly, most of the heavy duty tasks need to be performed in the DB. For example, the high-precision FPGA-based 32 channel TDC may need a big chunk of the FPGA resource.

Therefore, we will not implement a NIOS CPU in the DB board. Instead, we will use the NIOS CPU on the DUC board to configure and monitor the DB boards. That will be implemented through two DB command coprocessors. One is in the DUC board. It will receive the commands from the CPU through the register interface array module, transmit it to its partner on the DB, receives the responses from its partner on the DB, and send the responses back to the CPU through the register interface array module.

For the DUC CPU, the DB command coprocessors are completely transparent. DUC CPU will treat DB (and hardware resources in DB) as regular hardware resources.

## *DB (FPGA) framework model*

The DB firmware framework model has 3 modules (Figure 15):

1. DB command coprocessor
2. PET function logic circuits
3. Hardware interface logic circuits



**Figure 37** The DB firmware framework model.

# OpenPET Programming Tools / Environment

The OpenPET firmware and software include 4 main functional modules (Hardware Integrity Test, system configuration, data acquisition and data analysis), which are implemented in 5 main hardware modules (CUC, MB, DUC, DB and Host PC).

The programming languages used in OpenPET are VHDL (and schematic design) and standard C.

The programming tools required for OpenPET are Altera free design tools (**X**) and NI LabWindows/CVI ($2,599) (**X**)

Table 1 OpenPET Programming Tools / Environment

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **DB** | **DUC** | **MB** | **CUC** | **Host PC** |
| **FPGA Firmware** | **X** | **X** | **X** | **X** |  |
| **Embedded Microprocessor Software** |  | **X** | **X** | **X** |  |
| **PC Software** |  |  |  |  | **X** |

Host computer: Window 7

Nios II microprocessor: Nios II real time OS.

## *Collaboration through Github*

All source code and documentation will be made publically available on github.com and can be found by searching “OpenPET”. The OpenPET project has been divided into several Github repositories to store the hardware and software files independently. All users have read-only access to these files by default. If you are new to Github, a Github in windows tutorial is provided in “HostPC/Getting Started”[[1](#_ENREF_1)]. If you would like to contribute source code from a local directory, you can make a pull request on the Github website. Your code will be checked by an administrator and included as long as it is bug-free and adheres to OpenPET coding standards (uses Doxygen style comments, etc.). If the administrators trust your ability to contribute to the project and adhere to OpenPET coding standards, you can be given independent read/write permission by being listed as a collaborator on Github.

## *OpenPET Programming Code Styles*

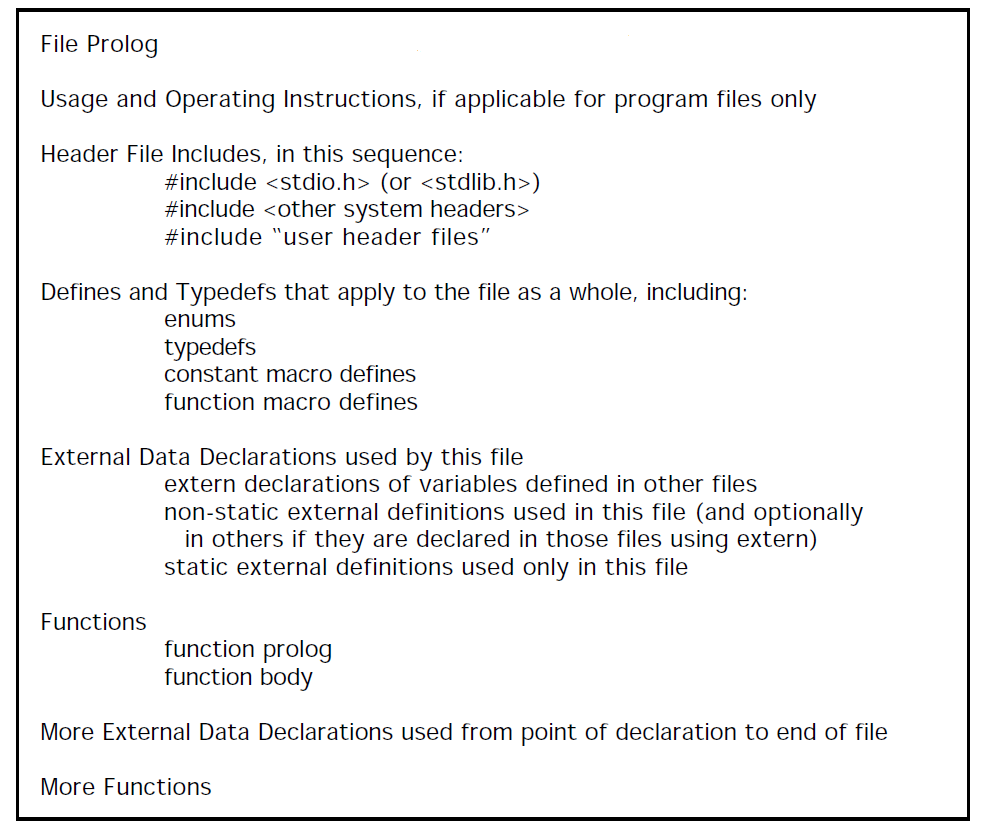
The source codes developed for OpenPET will be in a style which is well organized, easy to read and understand, maintainable and efficient.

### *C Programming Code Styles*

We suggest that the C codes running in the NIOS CPU and host PC are be developed in a style following the “C Style Guide” suggested by Software Engineering Laboratory (SEL), the National Aeronautics and Space Administration (NASA) in 1994. For detailed information, please download and read “nasa-c-style.pdf” from “homepages.inf.ed.ac.uk/dts/pm/Papers/nasa-c-style.pdf‎”[[2](#_ENREF_2)].

#### Program file style

To guarantee the readability and maintainability of the source codes, we request the programmers to write the C code program files in a style shown in Figure below.



**Figure 38** C code program files style.

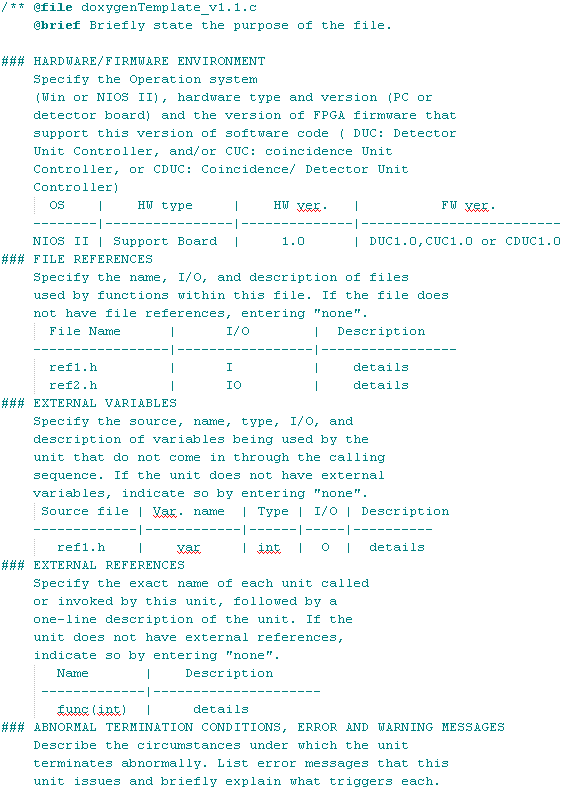
#### Comments style

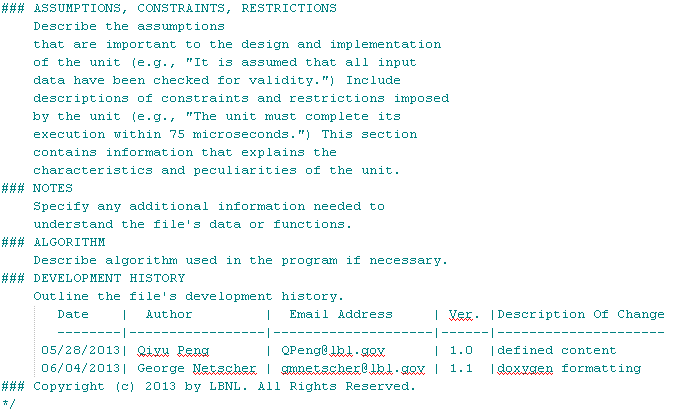
We use Doxygen (<http://www.stack.nl/~dimitri/doxygen/>) in order to automatically generate html documentation from comments placed within C and VHDL code. Special care must be taken to make comments in the correct format so that Doxygen can properly parse the comments and create error-free documentation.

To guarantee the readability and maintainability of the source codes, we request the programmer to placed different levels of comments in the codes:

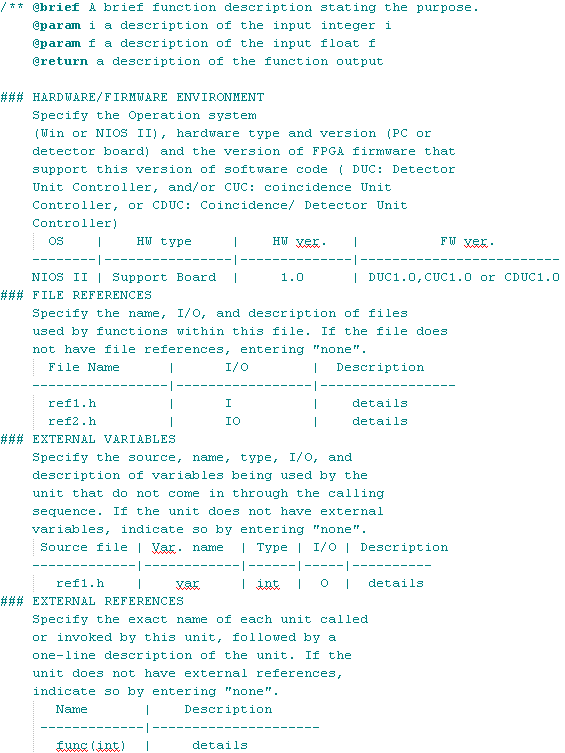
1. Optional: at the program level, include a README file that provides a general description of the program and explains its organization. The README file might include (a) all conditional compilation flags and their meanings; (b) files that are machine dependent; and (c) paths to reused components.
2. At the file level, include a file prolog that explains the purpose of the file and provides other information (figure shown below, example shown in doxygenTemplate\_vl.2.c[[3](#_ENREF_3)]). Note that the comment block should start at the beginning of the file, open with /\*\*, contain @file and @brief commands to tell Doxygen the file name and brief description, and contain each of the specified subheadings marked by ###. Please, include each subheading even if the value specified is simply “none”. Plain text tables should be created exactly in the style shown with | to separate each column and --- to separate only the first heading row from the following rows. This specific format will be recognized by Doxygen to create an elegant table in the documentation.
3. At the function level, add a comment which serves as a function prolog. (figure shown below, example shown indoxygenTemplate\_v1.2.c). Note that the comment block must be placed directly before the function prototype and open with /\*\*. It should contain @brief, @param, and @return to provide a brief description of the function, of each parameter, and of the return variable, respectively. Other commands may be used, such as @sa “see also” to specify related functions (<http://www.stack.nl/~dimitri/doxygen/manual/commands.html>). Again, subheadings should be denoted by ### and tables created with | and ---. To avoid redundancy, we do not ask you to include all function subheadings. Only use those that provide relevant information about the function.
4. Throughout the file, where data are being declared or defined, add comments to explain the purpose of the variables. (please refer to “nasa-c-style.pdf”[[2](#_ENREF_2)] page 7 and 8 for examples of recommended boxed comments, Block comments, Short comments and Inline comments).

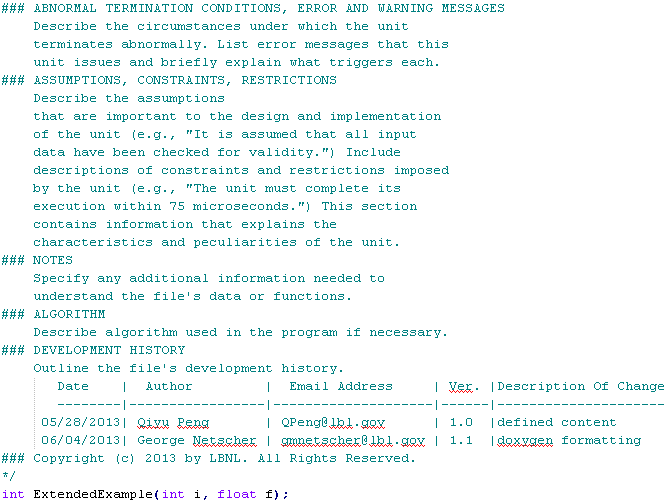
In order to maintain organization with the OpenPET project, we request the developers to use the template shown below. For more information about the template, please refer to “doxygenGuide\_For\_OpenPET\_Developer\_v1.2.docx”.[[4](#_ENREF_4)]





**Figure 39** Program file prolog format.





**Figure 40** function prolog format.

#### Meaningful Names

To guarantee the readability and maintainability of the source codes, we request the programmers to obey the following rules to create element names:

* Variables (and Type names): Use lower-case words separated by underscores.

For example: open\_database

* Function names: Capitalize the first letter of each word; do not use underscores.

For example: ProcessError

* Constants (and Enumeration types) : Use upper-case words separated by underscores.

For example: MAX\_COUNT

We suggest the programmers to follow the following guidelines to create element names:

* Choose names with meanings that are precise and use them consistently throughout the program.
* Follow a uniform scheme when abbreviating names. For example, if you have a number of functions associated with the “data refresher,” you may want to prefix the functions with “dr\_”.
* Avoid abbreviations that form letter combinations that may suggest unintended meanings. For example, the name “inch” is a misleading abbreviation for “input character.” The name “in\_char” would be better.
* Use underscores within names to improve readability and clarity:

get\_best\_fit\_model

load\_best\_estimate\_model

* Assign names that are unique (with respect to the number of unique characters).
* Use longer names to improve readability and clarity. However, if names are too long, the program may be more difficult to understand and it may be difficult to express the structure of the program using proper indentation.
* Names more than four characters in length should differ by at least two characters. For example, “systst” and “sysstst” are easily confused. Add underscores to distinguish between similar names:

systst sys\_tst

sysstst sys\_s\_tst

* Do not rely on letter case to make a name unique. Although C is casesensitive (i.e., “LineLength” is different from “linelength” in C), all names should be unique irrespective of letter case. Do not define two variables with the same spelling, but different case.
* Do not assign a variable and a typedef (or struct) with the same name, even though C allows this. This type of redundancy can make the program difficult to follow.
* Some standard short names for code elements are listed in the example below. While use of these names is acceptable if their meaning is clear, we recommend using longer, more explicit names, such as “buffer\_index.”

-- Example: standard short names

c characters

i, j, k indices

n counters

p, q pointers

s strings

-- Example: standard suffixes for variables

\_ptr pointer

\_file variable of type file\*

\_fd file descriptor

* When naming internal variables used by a function, do not duplicate global variable names. Duplicate names can create hidden variables, which can cause your program not to function as you intended.
* In separate functions, variables that share the same name can be declared. However, the identical name should be used only when the variables also have the identical meaning. When the meanings of two variables are only similar or coincidental, use unique names to avoid confusion.

### *VHDL Programming Code Styles*

We request that OpenPET FPGA programmers using VHDL in styles suggested by Xilinx (Coding Style Guidelines, <http://users.eecs.northwestern.edu/~seda/coding_guidelines_013003.pdf>), Altera (Recommended HDL coding styles, <http://www.altera.com/literature/hb/qts/qts_qii51007.pdf>), and Dr. Erno Salminen (VHDL Coding Rules, Tampere University of Technology, <http://www.cs.tut.fi/~ege/Misc/dcs_vhdl_coding_rules_es_v4_4.pdf>).

#### Rules and guidelines

To guarantee the readability and maintainability of the source codes, we suggest the programmers to write the codes following rules and suggestions below.

**Basic rules:**

* One entity and one architecture per VHDL file. The entity and the file must have the same name.
* Every VHDL file starts with a standard header (details will be described in the next section).
* Use naming conventions (details will be described in the next section).
* Provide enough comments (details will be described in the next section).
* Use a VHDL-aware text editor that provides a consistent indentation style. (refer to Page 13-9, Figure 13-7 in Xilinx Coding Style Guidelines).
* For declarations, instantiations, and mappings use one line for each signal.
* Avoid of magic numbers, use constants or generics instead.
* In a clocked process
* ONLY an asynchronous set or reset and clock should be in the sensitivity list.
* All control registers must be initialized in set or reset
* In a combinatorial process, all signals that are read (which can change) must be in the sensitivity list.

**Suggestions:**

* Avoid mixing of different coding styles (register transfer level, structural, gate level)
* Design a test bench for each entity.
* Register the output from an entity
* With names have suffixes \_in, \_out or \_inout.
* Only types STD\_LOGIC and STD\_LOGIC\_VECTOR
* Use registered outputs
* Indexes of STD\_LOGIC\_VECTOR are defined as DOWNTO
* Use named signal mapping in component instantiation, never ordered mapping
* Use assertions

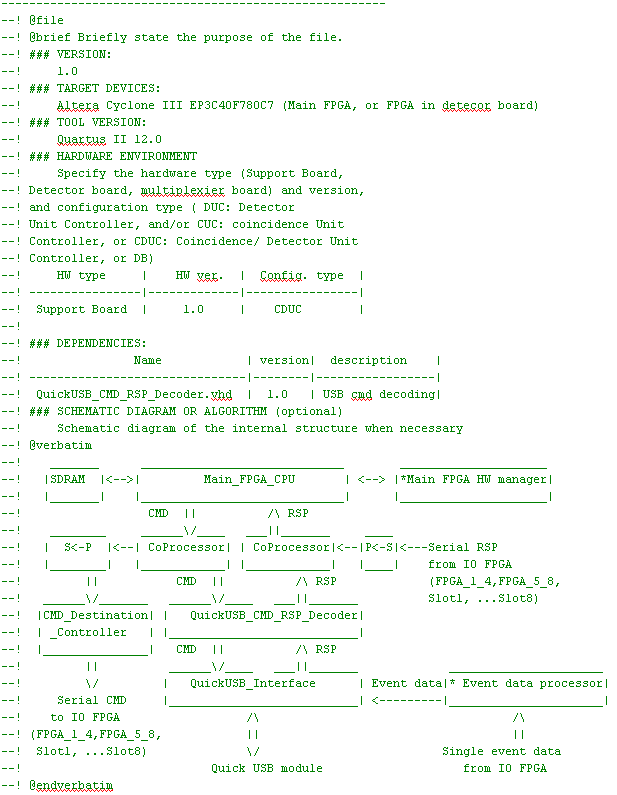
#### Comments style

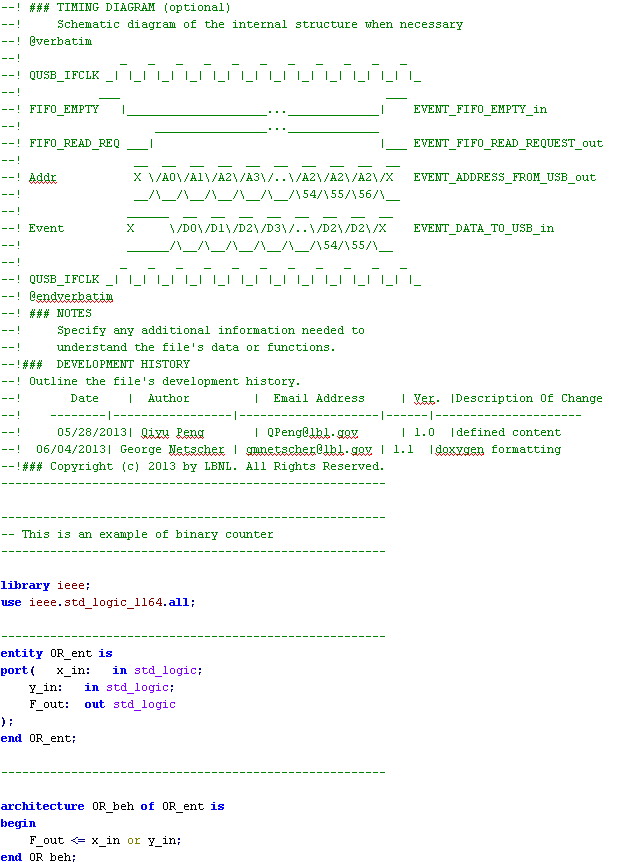
We use Doxygen (<http://www.stack.nl/~dimitri/doxygen/>) in order to automatically generate html documentation from comments placed within C and VHDL code. Special care must be taken to make comments in the correct format so that Doxygen can properly parse the comments and create error-free documentation.

To guarantee the readability and maintainability of the source codes, we request the programmer to place different levels of comments in the codes:

1. At the file level, include a file prolog that explains the purpose of the file and provides other information (figure shown below, example shown in doxygenTemplate\_v1.2.vhd[[3](#_ENREF_3)]). Note that the comment block should start at the beginning of the file, open and close with a line of -----, begin each line with --!, contain @file and @brief commands to tell Doxygen the file name and brief description, and contain each of the specified subheadings marked by ###. Please, include each subheading even if the value specified is simply “none”. Plain text tables should be created exactly in the style shown with | to separate each column and --- to separate only the first heading row from the following rows. This specific format will be recognized by Doxygen to create an elegant table in the documentation. As shown below, schematic diagrams should be created when possible. To preserve the formatting of these diagrams, they must begin with the command @verbatim and end with the command @endverbatim.
2. Throughout the file, where data are being declared or defined, add comments to explain the purpose of the signals and variables.

In order to maintain organization with the OpenPET project, we request the developers to use the template shown below. For more information about the template, please refer to “doxygenGuide\_For\_OpenPET\_Developer\_v1.2.docx”.[[4](#_ENREF_4)]



 **Figure 41** VHDL file prolog format.

#### Meaningful Names

To guarantee the readability and maintainability of the source codes, we suggest the programmers to obey the following rules to create element names:

* Architecture name is one of following:
* Behavioral -- Implies physical logic, cannot be compiled with RTL tools
* Rtl -- Implies physical logic, compiled with RTL tools
* Structural -- Implies physical connections, but not any logic
* Clk and reset signals/inputs
* Synchronous reset: rst
* Asynchronous reset: arst
* Active-low synchronous reset: rst\_n
* Active-low asynchronous reset: arst\_n
* Clock signal: clk, or clk\_#MHz
* Name conventions for IO, signal and etc. :
* Input port: portname\_in
* Output port: portname\_out
* Combinatorial signal: signalname
* General register output: signalname\_r
* Constant: constantname\_c
* Generic: genericname\_g
* Variable: variablename\_v
* Order of entity’s ports. For example, group ports as:
* rst -- Resets
* clk -- Clocks (preferably just one)
* A\_in --Signals of group A
* B\_out --Signals of group B
* C\_in --Signals of group C
* Name intermediate signals
* Signals from instance a to b are named: signalname\_a\_b
* With multiple targets use: signalname\_from\_a
* Use enumeration for coding states in FSM
* Do not use: s0, s1, a, b, state0 , ...
* Use: idle, wait\_for\_x, start\_tx, read\_mem, ...
* Name the instantances: i\_componentname\_id
* Component instance name is formed from the component name
* Attach prefix “i\_” and identifier as a postfix
* Name generate statements: g\_componentname
* Label every process: label is written two times (before and after the process)

# Appendix 1 The OpenPET System Boot-up Sequences



**Figure 41** The OpenPET System Boot-up Sequence.

# References

[1] G. Netscher. *https://github.com/openpet-developer/HostPC/tree/master/Getting%20Started/Getting Started with Github.pptx*.

[2] *https://github.com/openpet-developer/HostPC/tree/master/Getting%20Started/nasa-c-style.pdf*.

[3] *https://github.com/openpet-developer/HostPC/tree/master/Getting%20Started/doxygenTemplate\_v1.2.h*.

[4] *https://github.com/openpet-developer/HostPC/tree/master/Getting%20Started/doxygenGuide\_For\_OpenPET\_Developer.docx*.