OpenPET Early Adopters Meeting

Dates: May 10-11, 2012

Location: Lawrence Berkeley National Laboratory

The presentations included here are from a small, informal workshop with the LBNL OpenPET developers and a few Early Adopters/Users. The content is preliminary in nature and may conflict with documentation and presentations posted on this website in the future.

openPET Early Adopters Meeting Agenda
Dates: May 10-11, 2012
Location: Lawrence Berkeley Laboratory, Berkeley, California
Room number: Building 55-121, Room 117 (Conference Room)

Thursday, May 10, 2012		
Time	Item	Presenter
9:30-10:00	Setup	
10:00-10:50	Introduction and Overview of Groups	
	Goal: Introduce groups to each other and understand how each group plans to make use of the OpenPET system	
10:00-10:20	Introduction of LBL Group and development history of OpenPET	Bill Moses
10:20-10:35	Overview of Manitoba Group and Plans	Andrew Goertzen
10:35-10:50	Overview of Davis Group and Plans	Simon Cherry
10:50-11:00	Break	
11:00-12:30	Overview of System Hardware	
	Goal: Provide an understanding of the technical specifications and capabilities of the system and how the pieces communicate with each other.	
11:00-11:30	Detector board design	Seng Choong
11:30-12:00	Support crate	Bill Moses
12:00-12:30	Board and system level communication overview	Qiyu Peng
12:30-13:30	Lunch	
13:30-15:30	Software/Firmware Framework and Demonstration	
	Goal: Provide an understanding of how to interface with the OpenPET, including writing of firmware, uploading of firmware, and acquisition of data from system.	
13:30-14:30	Overview and discussion of Software/Firmware Framework	Qiyu Peng
14:30-15:30	Demonstration of Hardware and Existing Programming Tools	Qiyu Peng
15:30-15:45	Break	
15:45-17:15	Functionality Requirements	
	Goal: Define what functions are required to enable the acquisition of data using an OpenPET system.	
15:45-16:15	Overview of desired functionality requirements from an end-user's perspective	Andrew Goertzen

Thursday, May 10, 2012		
16:15-17:15	Discussion to define and prioritize requirements	
17:15-17:30	Break	
17:30-18:30	Software/Firmware Management	
	Goal: Provide overview of web-based tools for communication between users.	
17:30-18:00	Website Description	Jenny Huber
18:00-18:30	Overview of wiki and software/firmware management	Martin Judenhofer
19:00-21:00	Group Dinner	
	Location and timing TBD.	

Friday, May 11, 2012		
Time	Item	Presenter
9:00-10:15	Collaboration Structure and Communication	
	Goal: Define how user groups will interact with each other and LBL.	
	Discussion topics: Frequency and method of communication. Progress monitoring and documenting. Authorship on publications.	
10:15-10:30	Break	
10:30-12:30	Data, Communications and Command Specifications	
	Goal: Discuss and define some of the more in-depth technical aspects of the operation of the system.	
10:30-11:30	Data format standards	Qiyu Peng
11:30-12:30	Communications and Command Specifications	Qiyu Peng
12:30-13:30	Lunch	
13:30-17:00	Discussions and Miscellaneous	
	Goal: Address all items left over.	
13:30-15:00	Division of tasks discussion	
15:00-15:15	Break	
15:15-17:00	TBD	

openPET

A Flexible Electronics System for Radiotracer Imaging

Introduction & Overview

May 10, 2012

William W. Moses

BERKELEY LAB

openPET Vision

General-Purpose Electronics & Software for Nuclear Medical Imaging Cameras

Open Source

- Hardware, Firmware, and Software
- Schematics, Gerbers, BOM,…

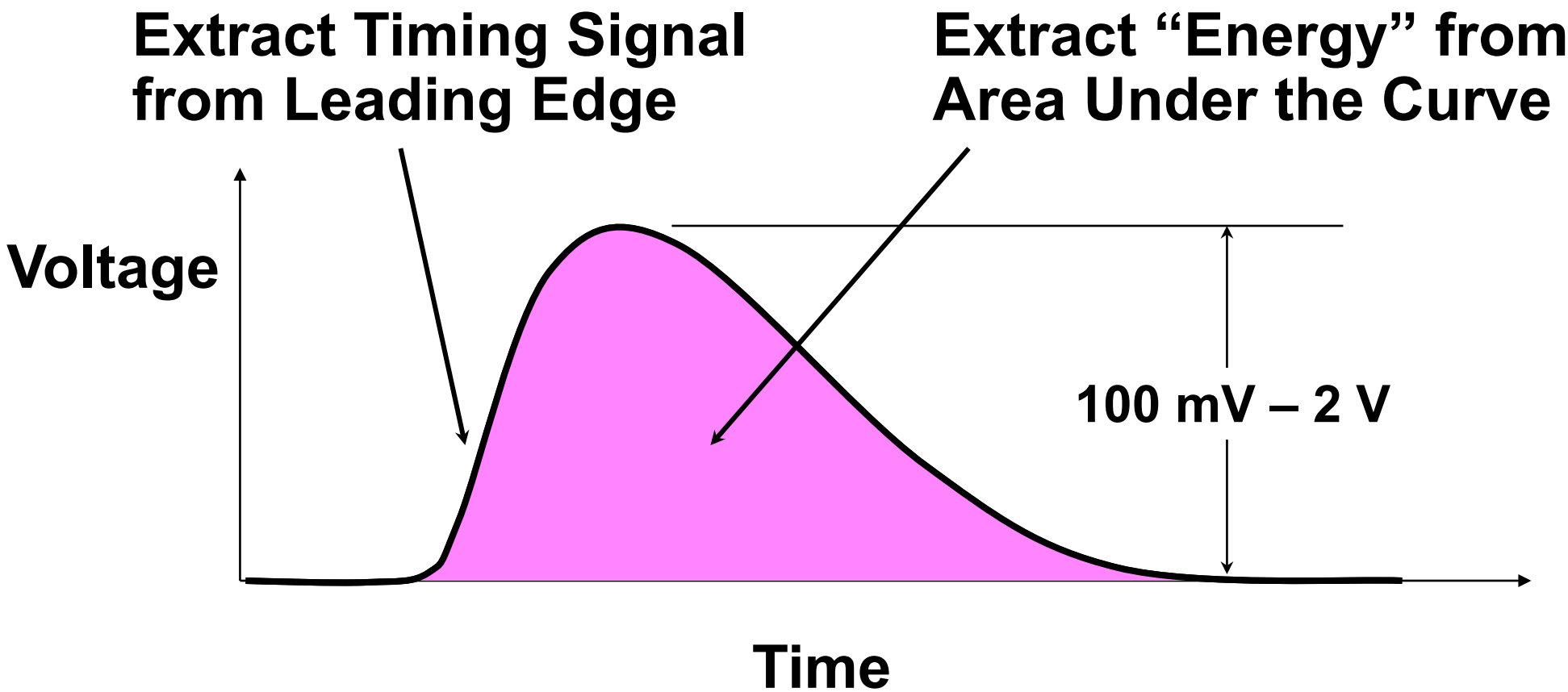
Active User Community

- Share Software and Expertise
- Module, Calibration, DAQ, Display,…

http://OpenPET.LBL.gov

BERKELEY LAB

All Detector Outputs Look the Same



- *Tremendous* Variation in How Outputs Are Combined
  ⇒Combine Outputs in Firmware

Electronics System Requirements

High-Performance

- # of Channels, Rate, Energy, Timing, …

Very Flexible

- Type of Detector, Camera Configuration, Event Word Definition

User-Modifiable

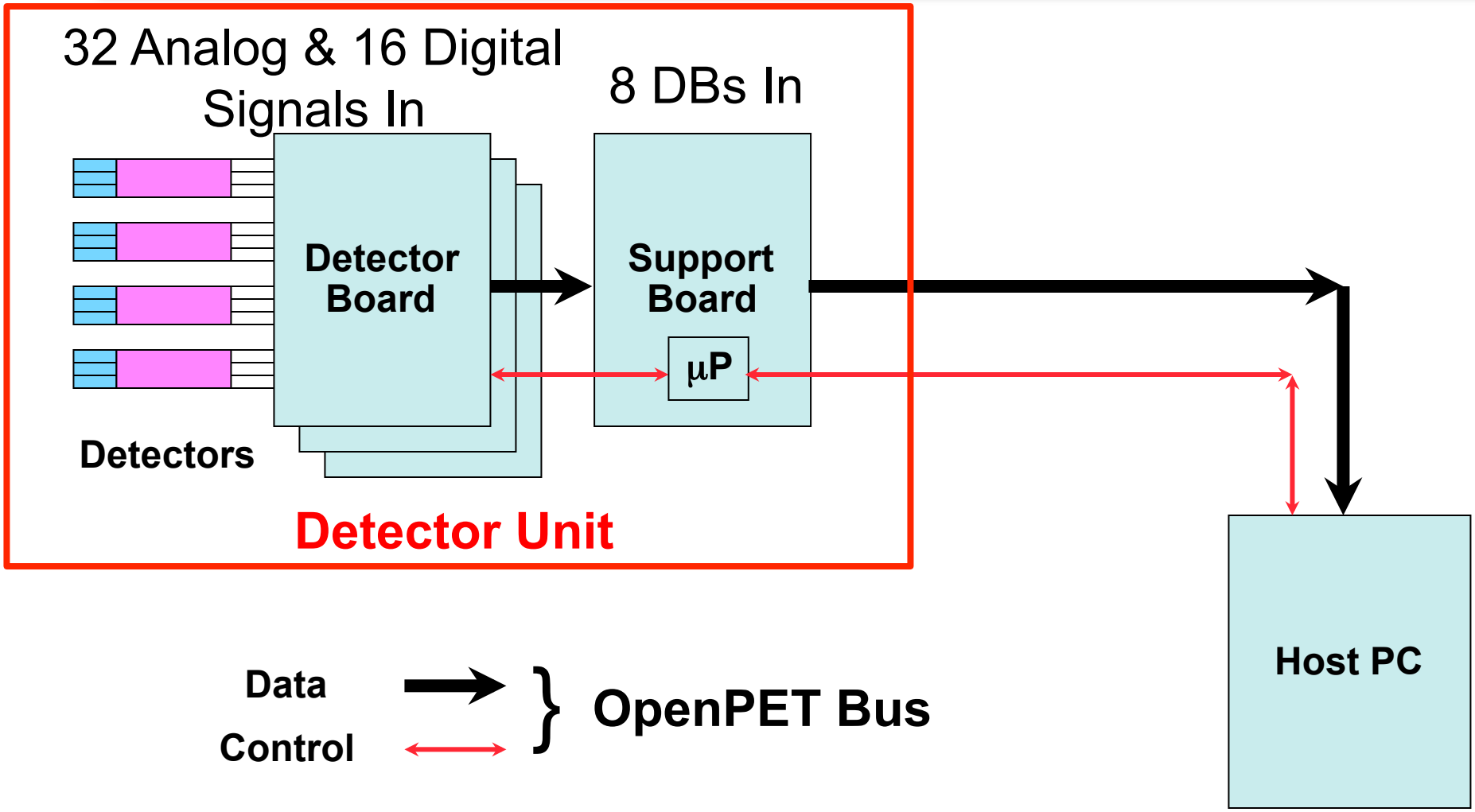
- Schematics, Source Code, Knowledge Base

User-Friendly

- Instructions, Documentation, Can Buy Boards

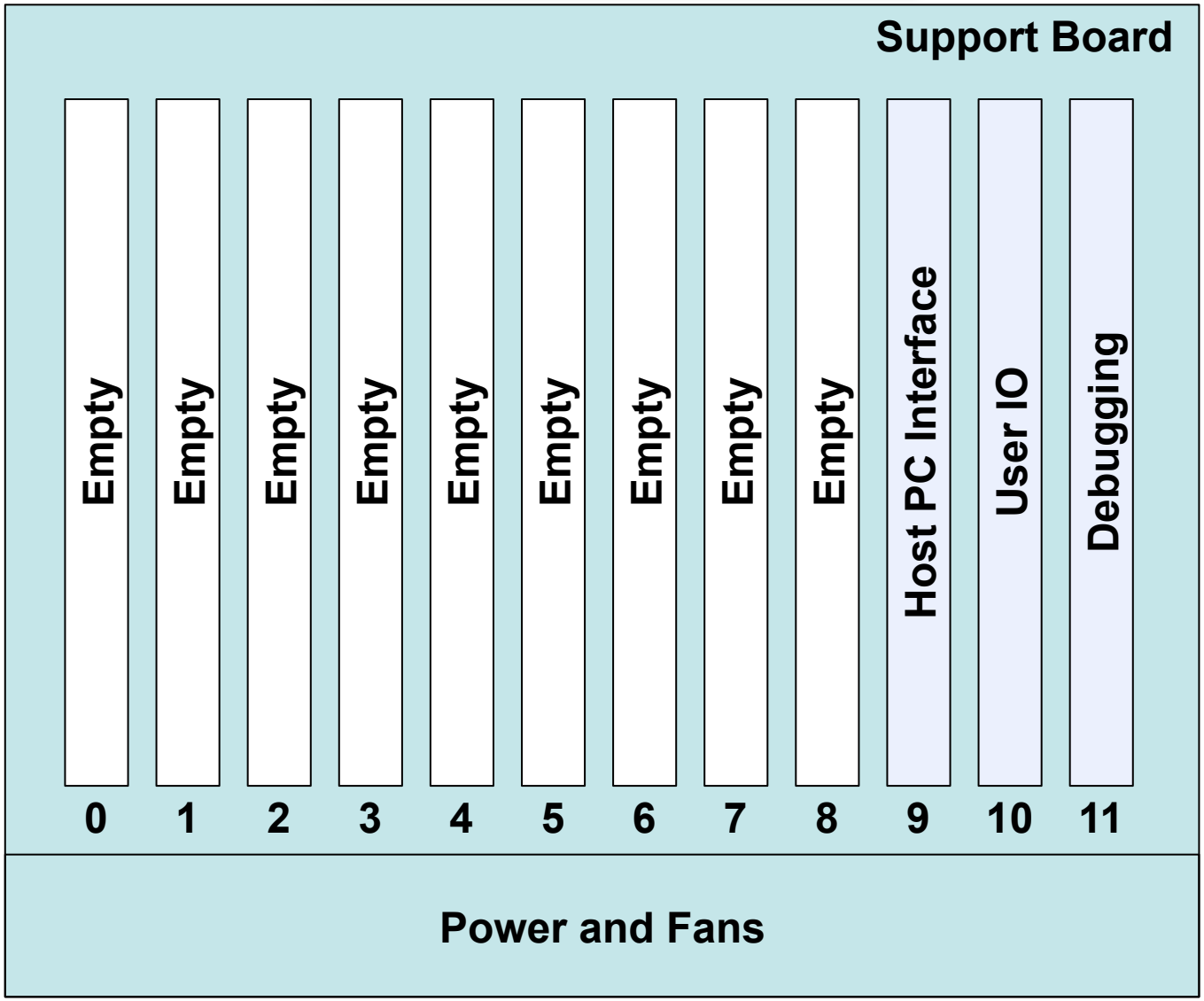
Like Open Source Software

OpenPET Architecture (Small System)



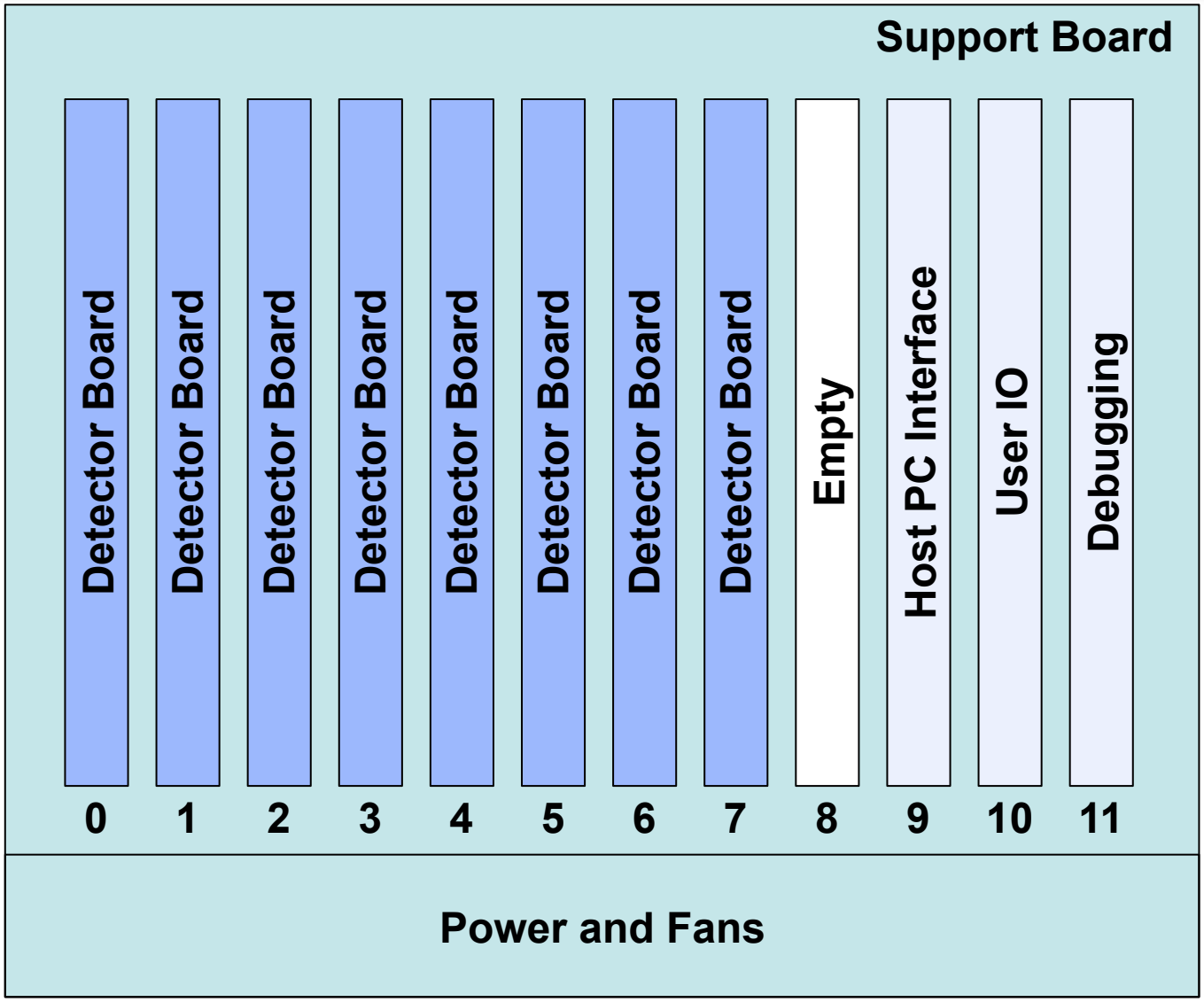
Supports 64 Block Detectors

Support Crate



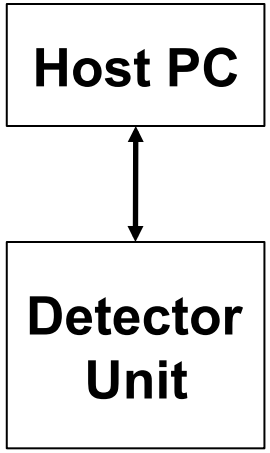
"VME Crate," Support Board, and 3 Interface Boards

Detector Unit (in Small System)

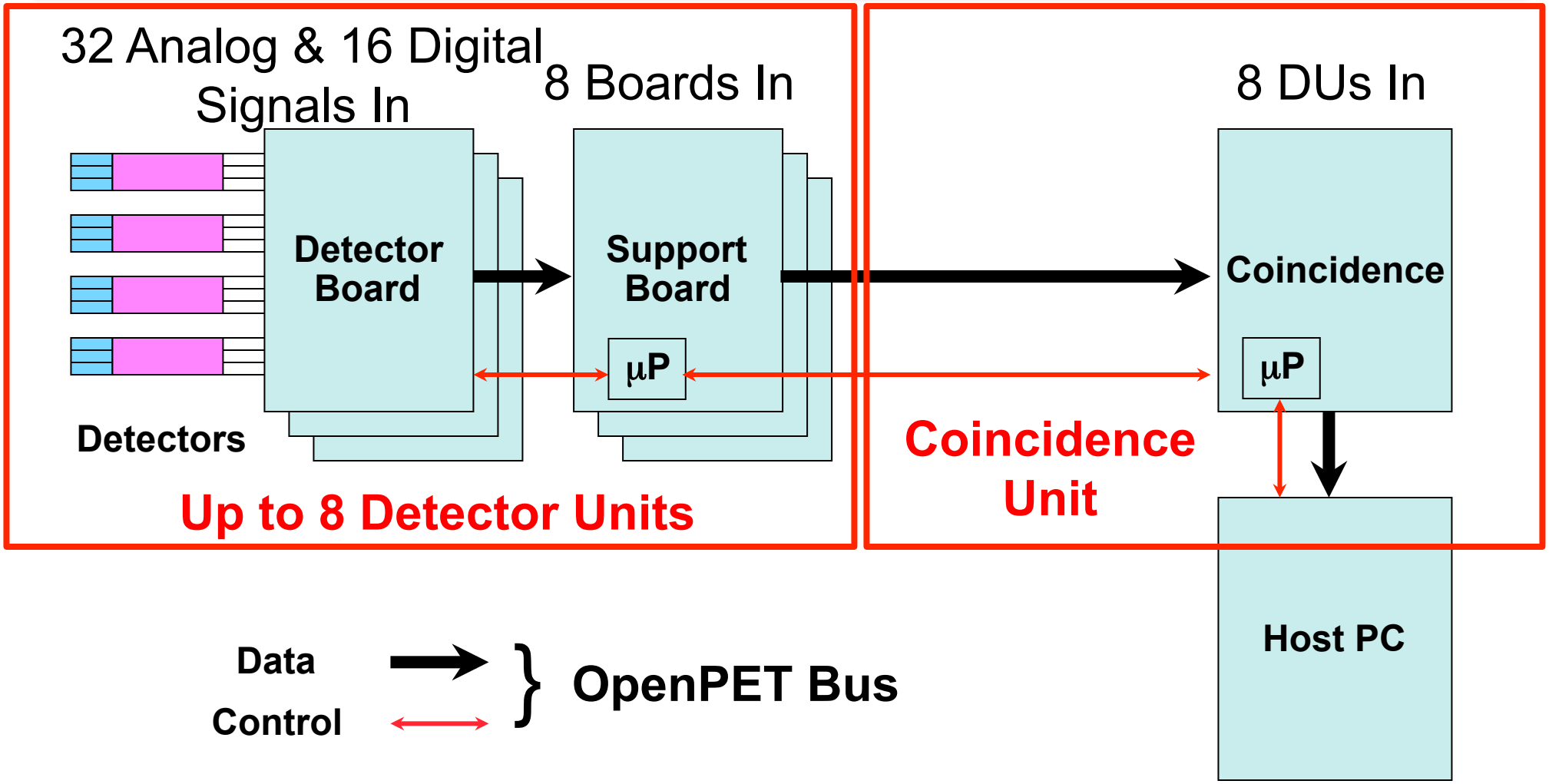


Support Crate and Up To 8 Detector Boards

Small System

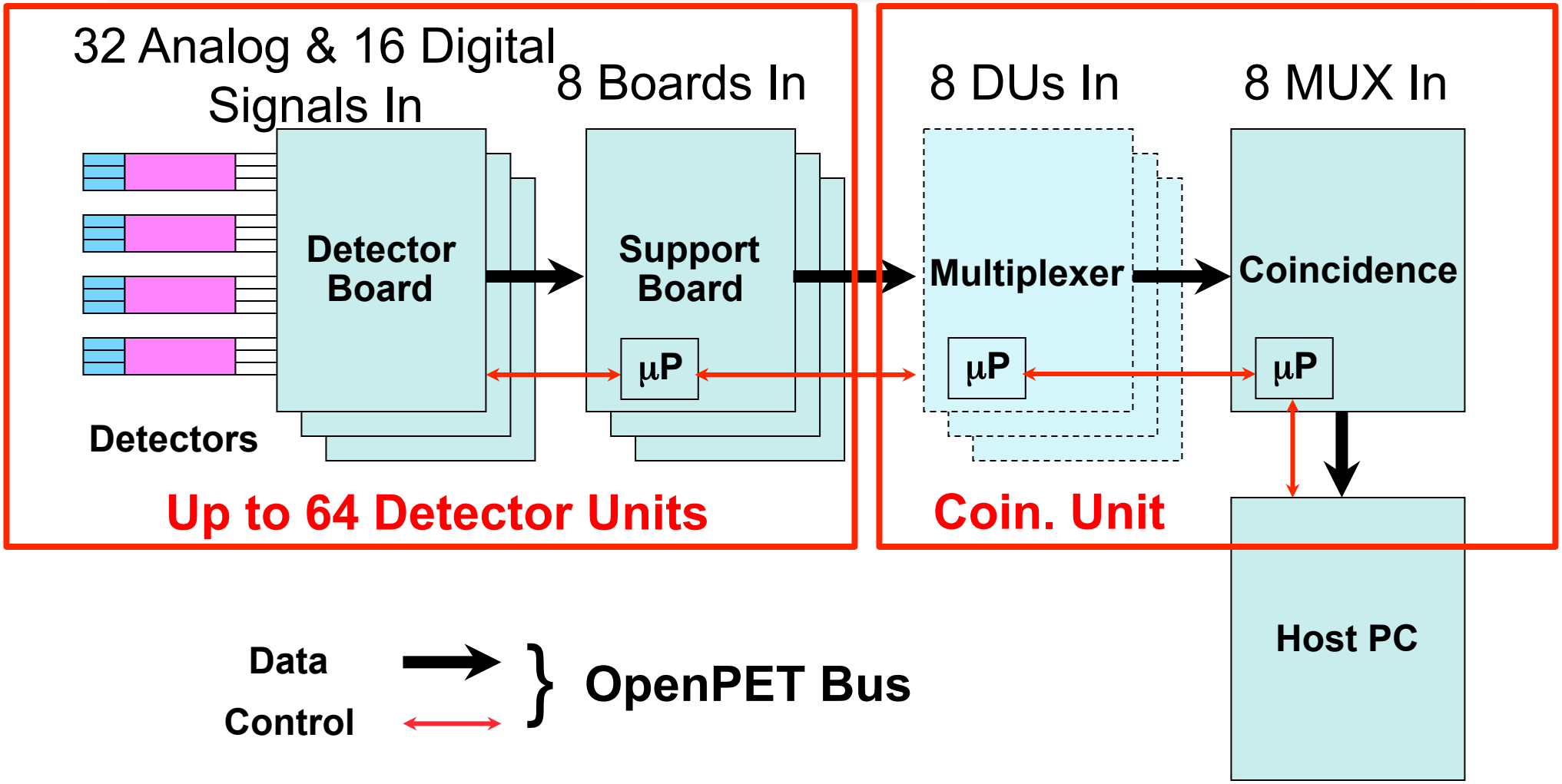


OpenPET Architecture (Standard System)



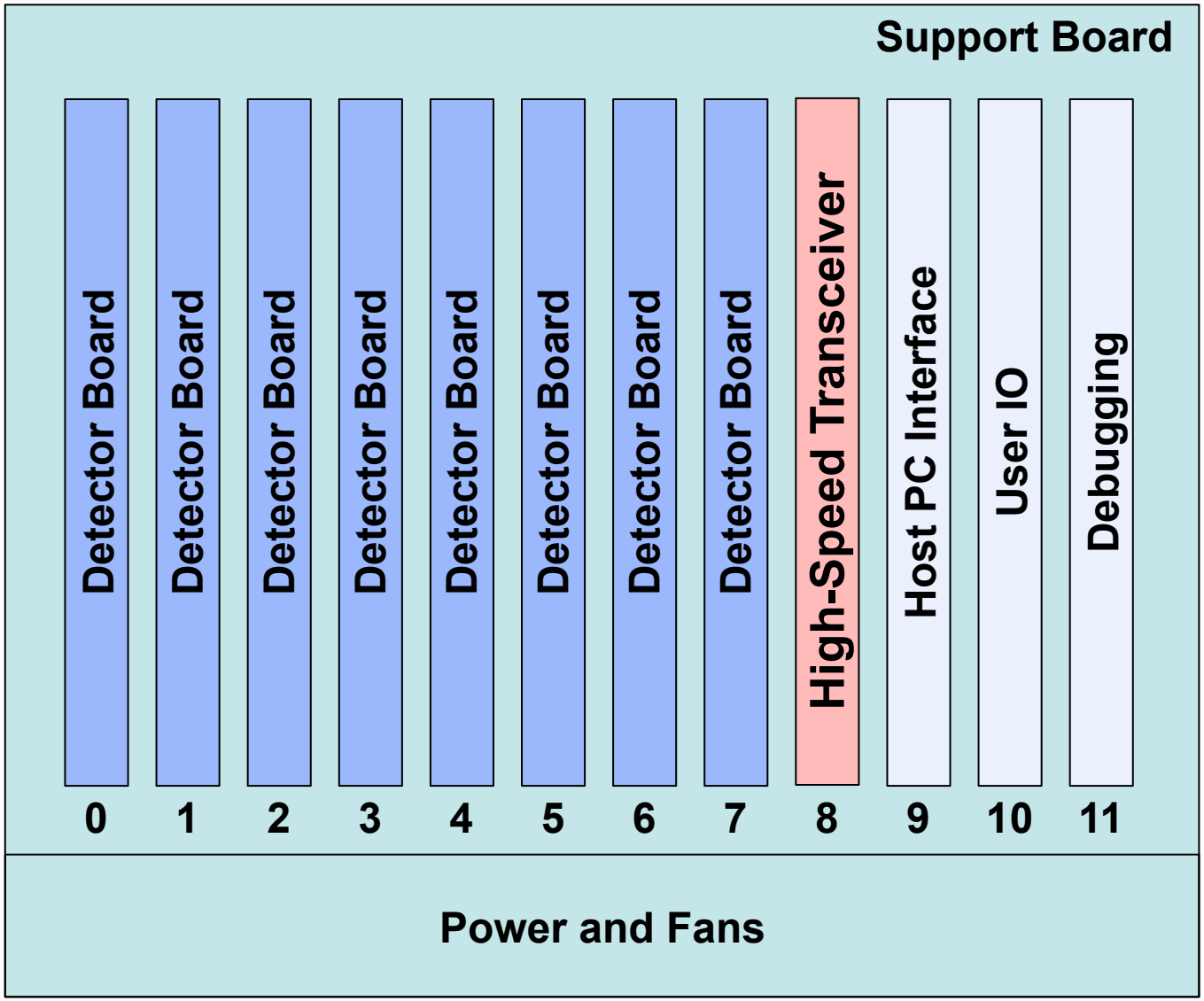
- Supports 512 Block Detectors
- Adds Coincidence Unit

OpenPET Architecture (Large System)



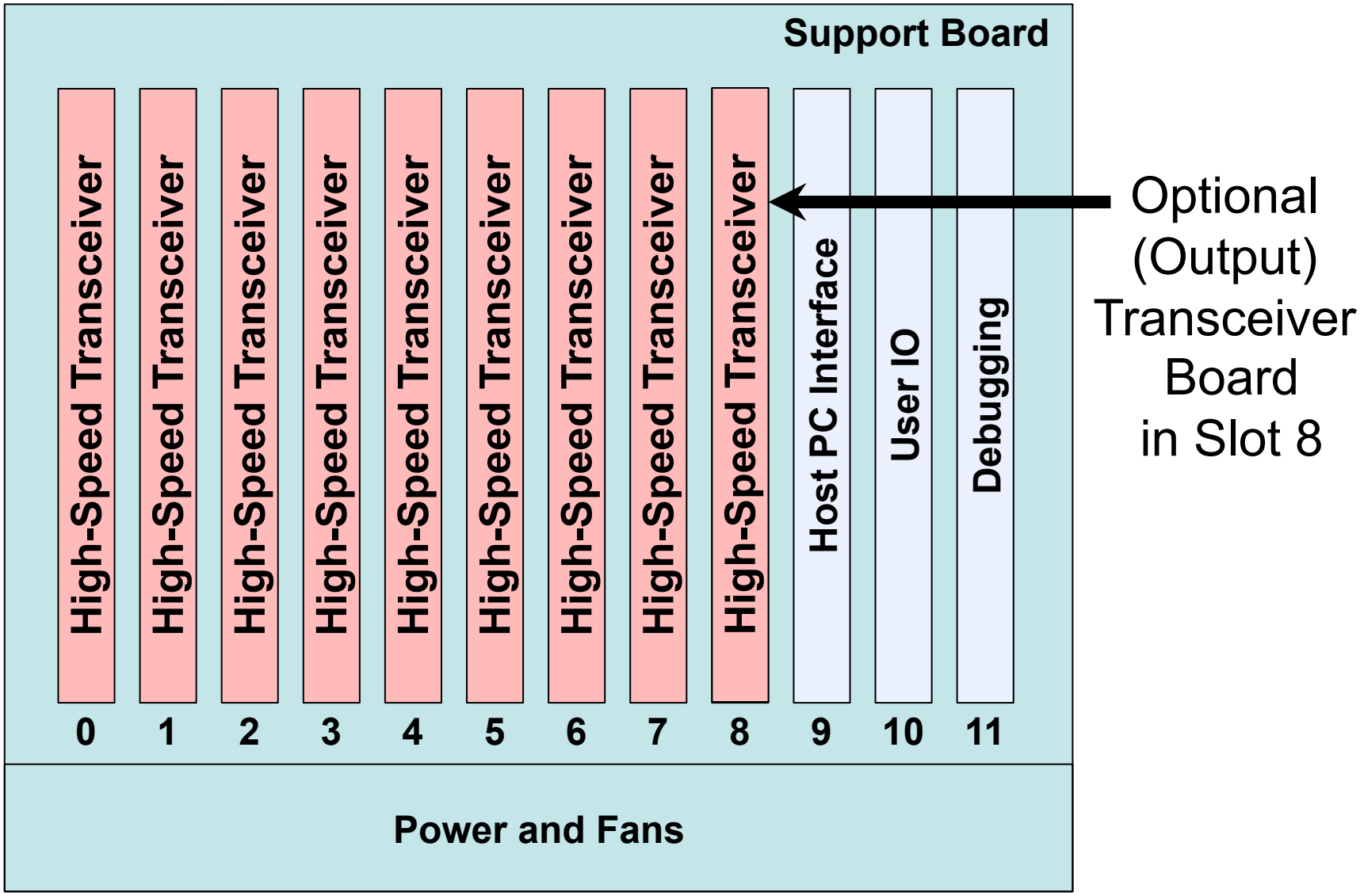
- Supports 4096 Block Detectors
- Adds Multiplexers

Detector Unit (in Standard & Large Systems)



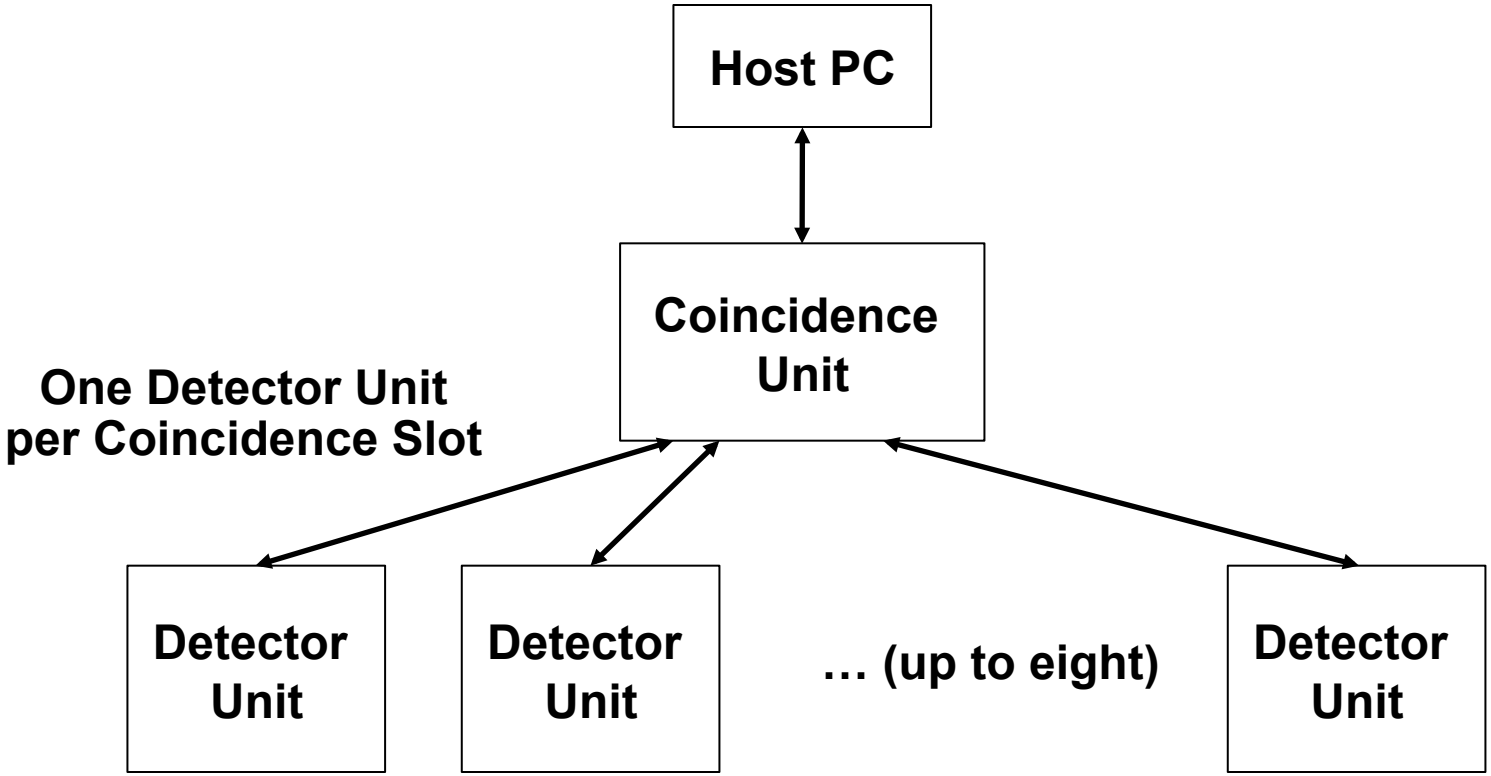
Small System Detector Unit and HS Transceiver Board

Coincidence Unit

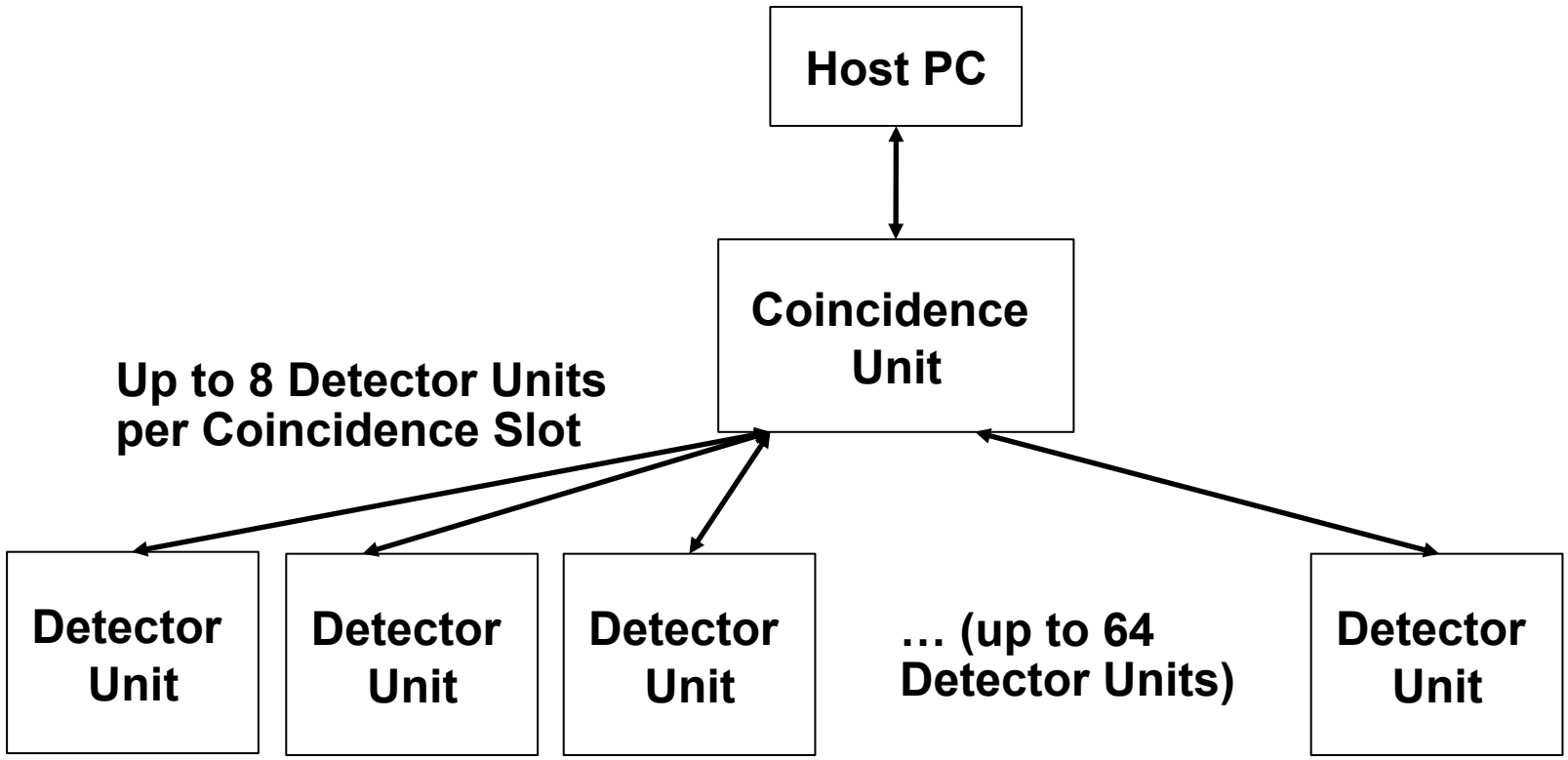


Support Crate and Up To 8 HS Transceiver Boards

Standard System



Large System



Proposed System to use openPET

Martin S. Judenhofer, PhD

Department of Biomedical Engineering, University of California-Davis, Davis, CA

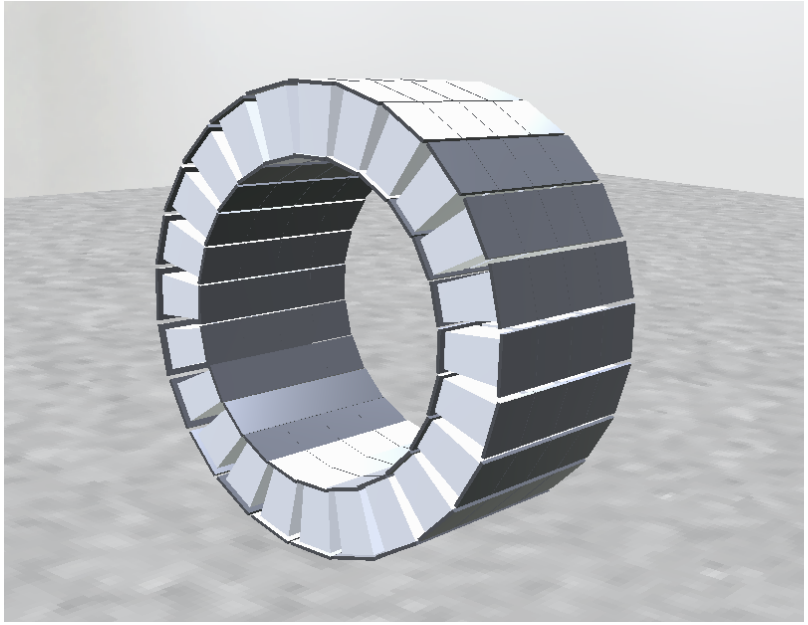
OpenPET Meeting, Berkeley, May 11-12, 2012

UCDAVIS
UNIVERSITY OF CALIFORNIA

cmgi

Proposed System to use openPET: DOI based PET scanner

- Use small crystals blocks
- Use dual ended readout to obtain depth of interaction (DOI) using PSAPDS
- 4-ring system, each ring 24 blocks (96 blocks total)



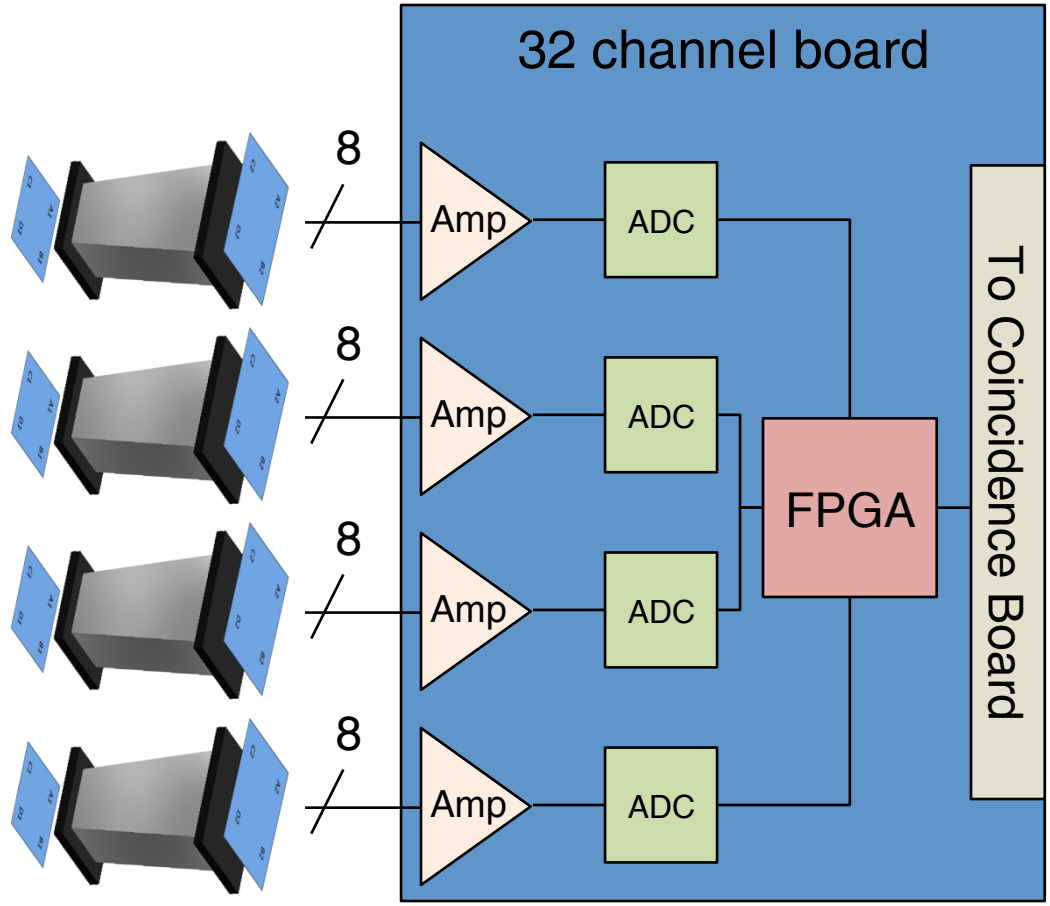
Block Detector Principle

- Each PSAPD has 4 corner signals
- Sum of all 8 signal is energy
- Ratio of Sum1 to Sum2 is DOI
- X/Y is calculated from ABCD and sum

→ 8 channels need to be sampled

OpenPET Detector-Board Usage

- Per Block 8 digitizer channels are required
- Sum of all 8 channels can be used as trigger
- All 8 signals are used to generate X/Y/E/DOI
- Signal duration ~ 200-300 ns
- Rise time 40-60 ns



Requirements

Data Processing Requirements

- Need to sample about 10-15 consecutive samples per event
- No CFD → Triggering on leading edge → large time walk → additional processing required to improve timing
- May require multiple X/Y look up tables for crystal positions depending on DOI depth (3-5)

Hardware

- 4 Blocks per detector board
- 24 Detector boards
- 3 Support boards to read all 24 detector boards (8 per board)
- 1 coincidence board

openPET

A Flexible Electronics System for Radiotracer Imaging

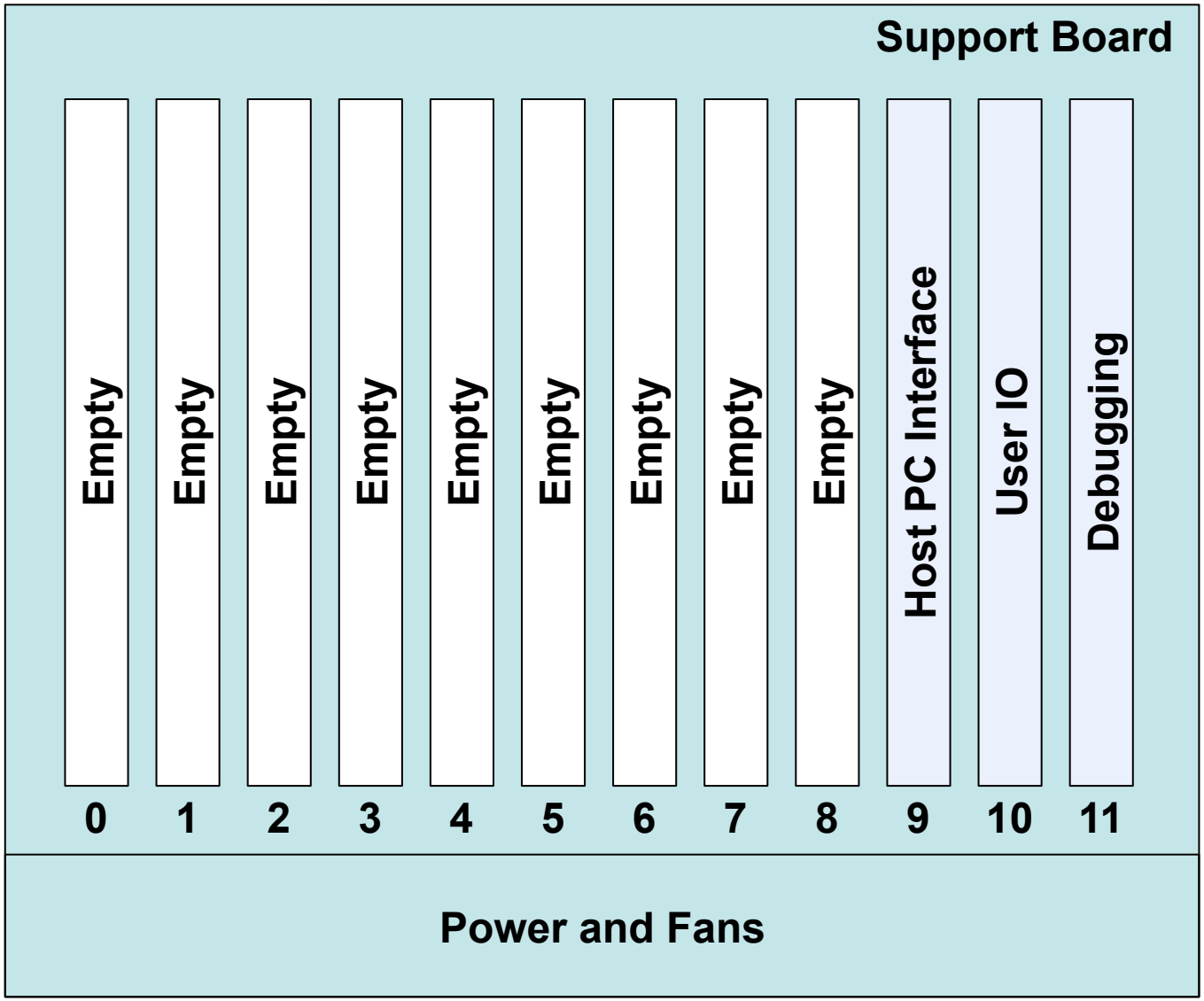
Support Crate

May 10, 2012

William W. Moses

BERKELEY LAB

Support Crate



"VME Crate," Support Board, and 3 Interface Boards

Support Crate Form Factor

Same as 12-Slot VME 6-U Crate

- Connectors in Different Locations
  - Avoid accidently plugging in VME boards!
- 8 Input Slots (Slots 0–7)
  - Detector Boards if Detector Unit
  - High-Speed Transceivers if Coincidence Unit
- 4 Slots for “Plug-In” Boards
  - High-Speed Transceiver (Slot 8)
  - Interface to Host PC (Slot 9)
  - User IO (Slot 10)
  - Debugging (Slot 11)

BERKELEY LAB

Plug-In Boards

Host PC Interface (Slot 9)

Front panel Ethernet connector, USB2 connector, SD card connector, 3 reset switches, 20 LEDs, detector bias voltage input (BNC connector, –100 V to +100 V).

User I/O (Slot 10)

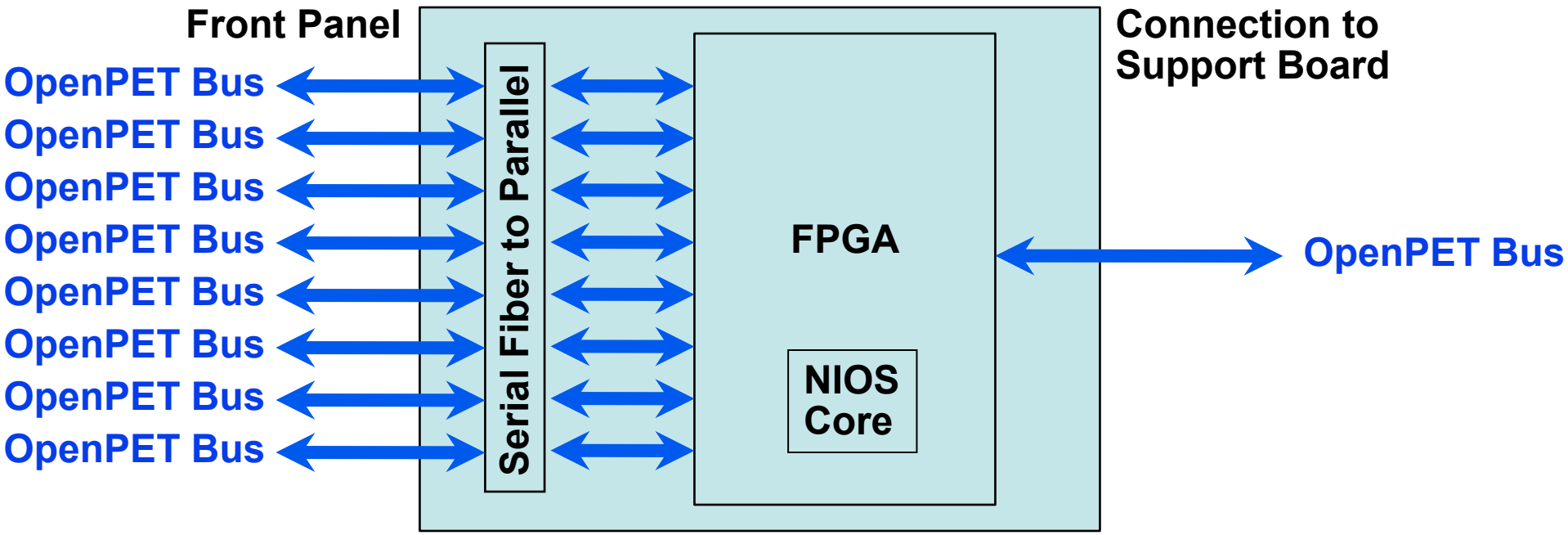
Front panel connectors for two RS-232 cables, an External Clock input (SMA), and 48 external digital IO lines. A jumper sets the logic levels for all 48 lines to 3.3 V or 5.0 V. Each Support Board FPGA (the Master and both Slaves) is connected to 16 lines. Groups of 4 adjacent lines are set via a DIP switch to be inputs or outputs.

Debugging (Slot 11)

Front panel JTAG connector (that can be used to program the FPGAs directly), four Aligent 16902B connectors for logic analyzers (two connect to the Main FPGA, and one to each of the two Slave FPGAs), and 30 user-defined LEDs (10 connected to each of the 3 FPGAs).

Brings Support Board Connectors to Front Panel

High-Speed Transceiver Board

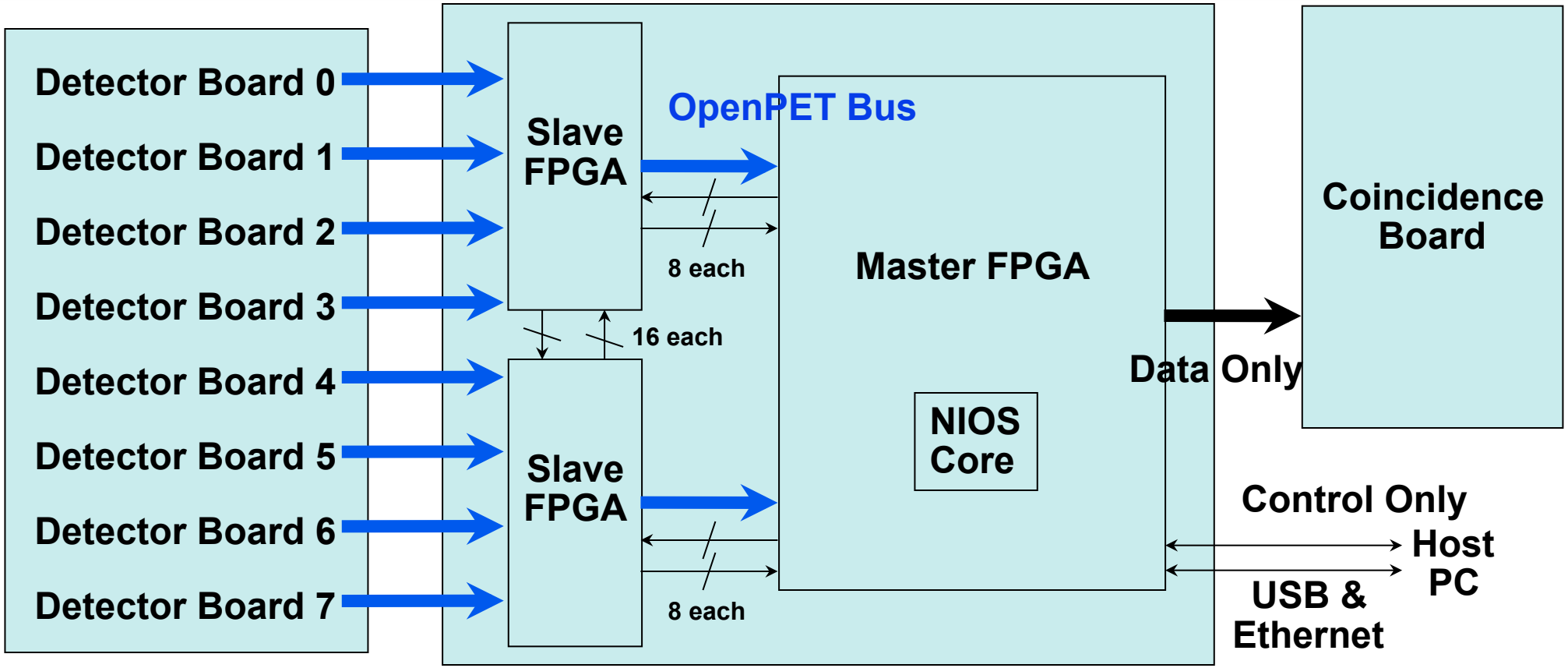


- Each OpenPET Bus can be programmed to be either an input or an output
- The FPGA can act as a multiplexer or fan-out
- This board can plug into Slot 0–7 (as an input to the SB) or into Slot 8 (as an output from the Support Board)
- NIOS Core supervises communication and control

Conceptual Design

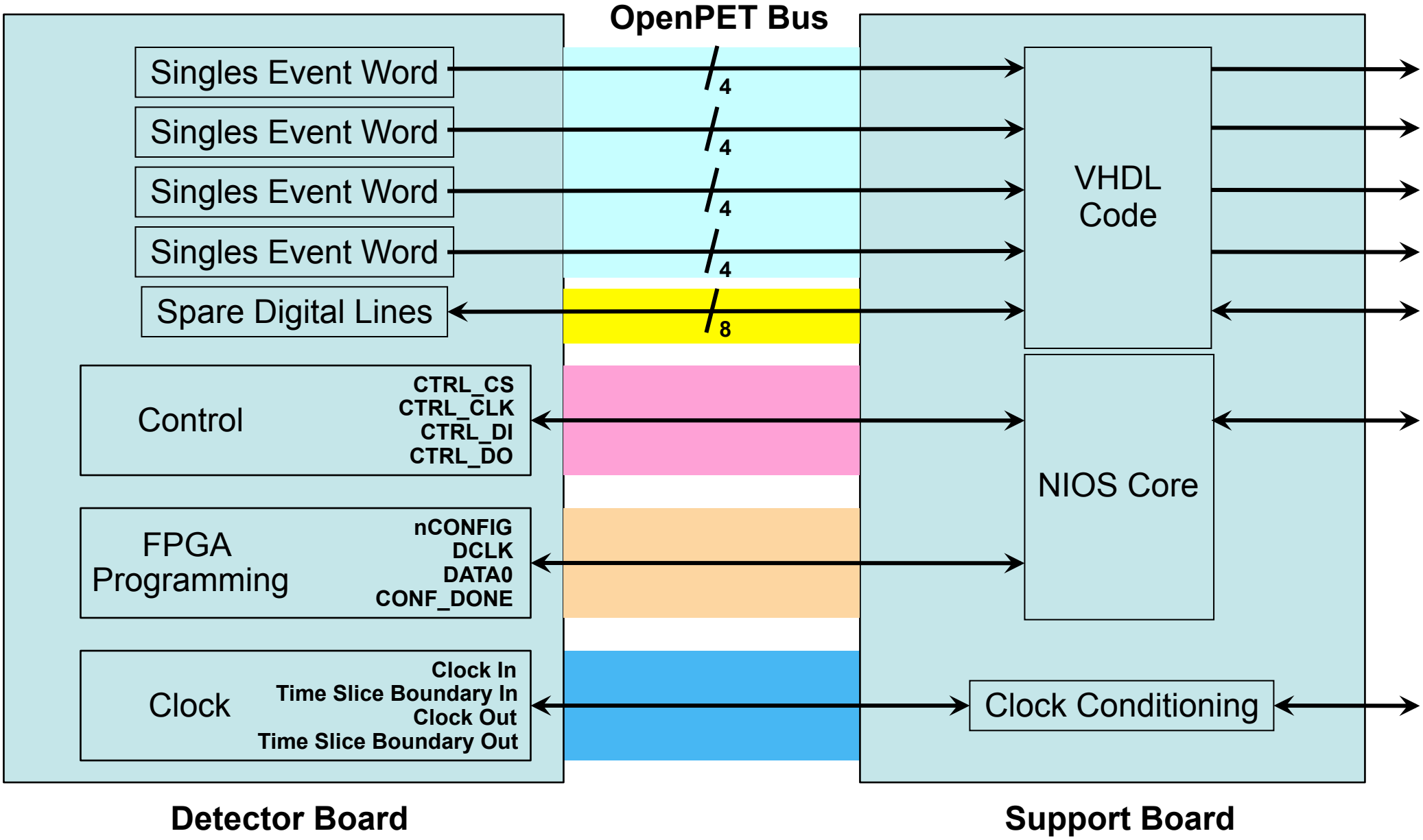
BERKELEY LAB

Support Board Event & Communication Flow



- Input and output of each functional unit is OpenPET Bus (sometimes with additional IO lines)
- VHDL handles all real-time multiplexing & data transfer
- "NIOS Core" is a CPU inside the FPGA that runs C, handles communication and control

Detector Board to Support Board Bus IO



Connector Between Detector Board and Support Board (OpenPET Bus)

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

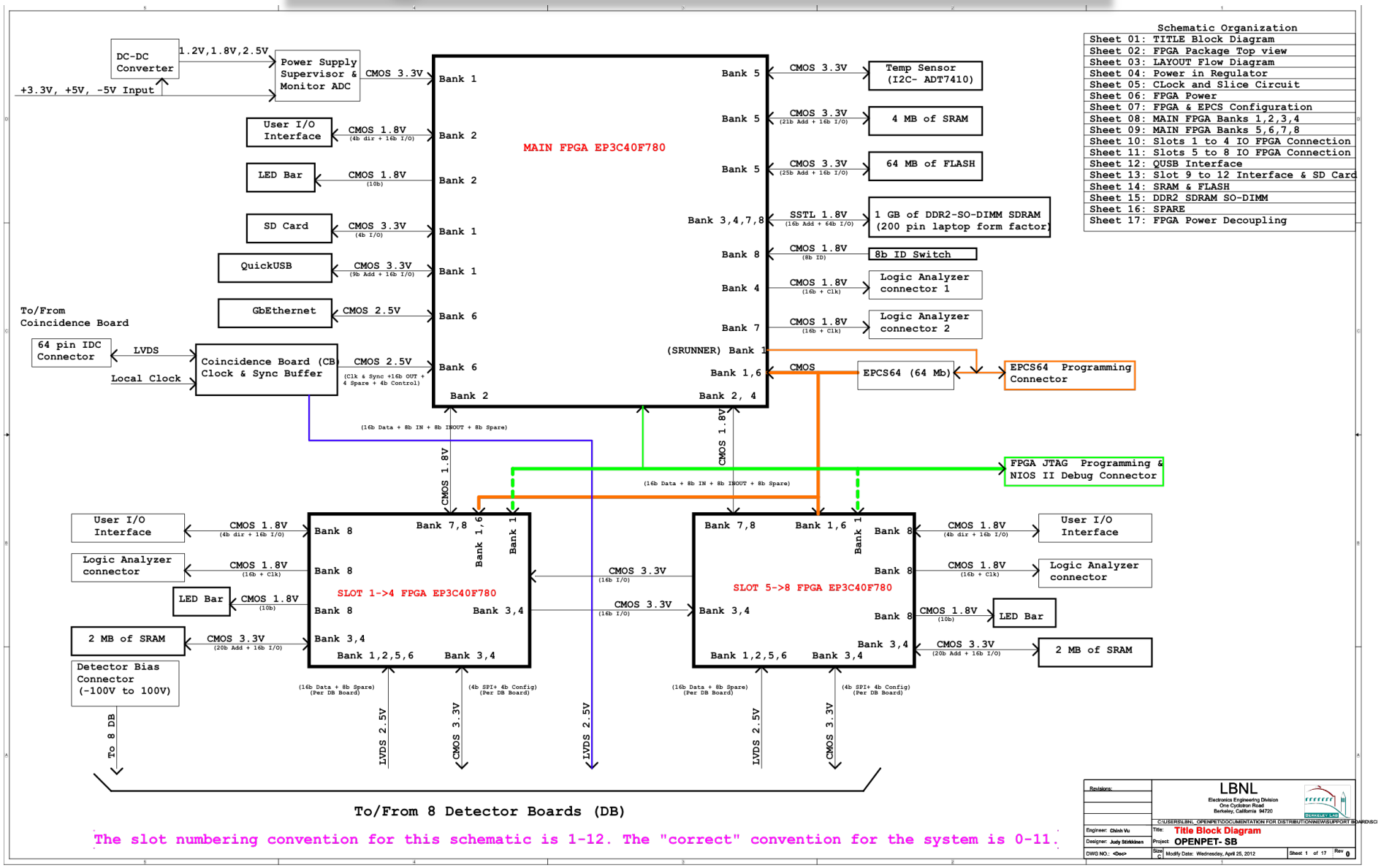
Color	Group Description
	LVDS Data to DB FPGA & Coincidence Board
	Clock & Slice IN/OUT
	Undefined pins between DB FPGA & DSB FPGA
	Slow control SPI interface signals
	DB FPGA serial programming pins
	No connection
	Power & GND
	Detector Bias Voltage (100 V max.)

Power & FPGA Programming Sometimes Omitted

Data coming into a board clocked by clock_in / slice_in, Data coming out of a board clocked by clock_out / slice_out

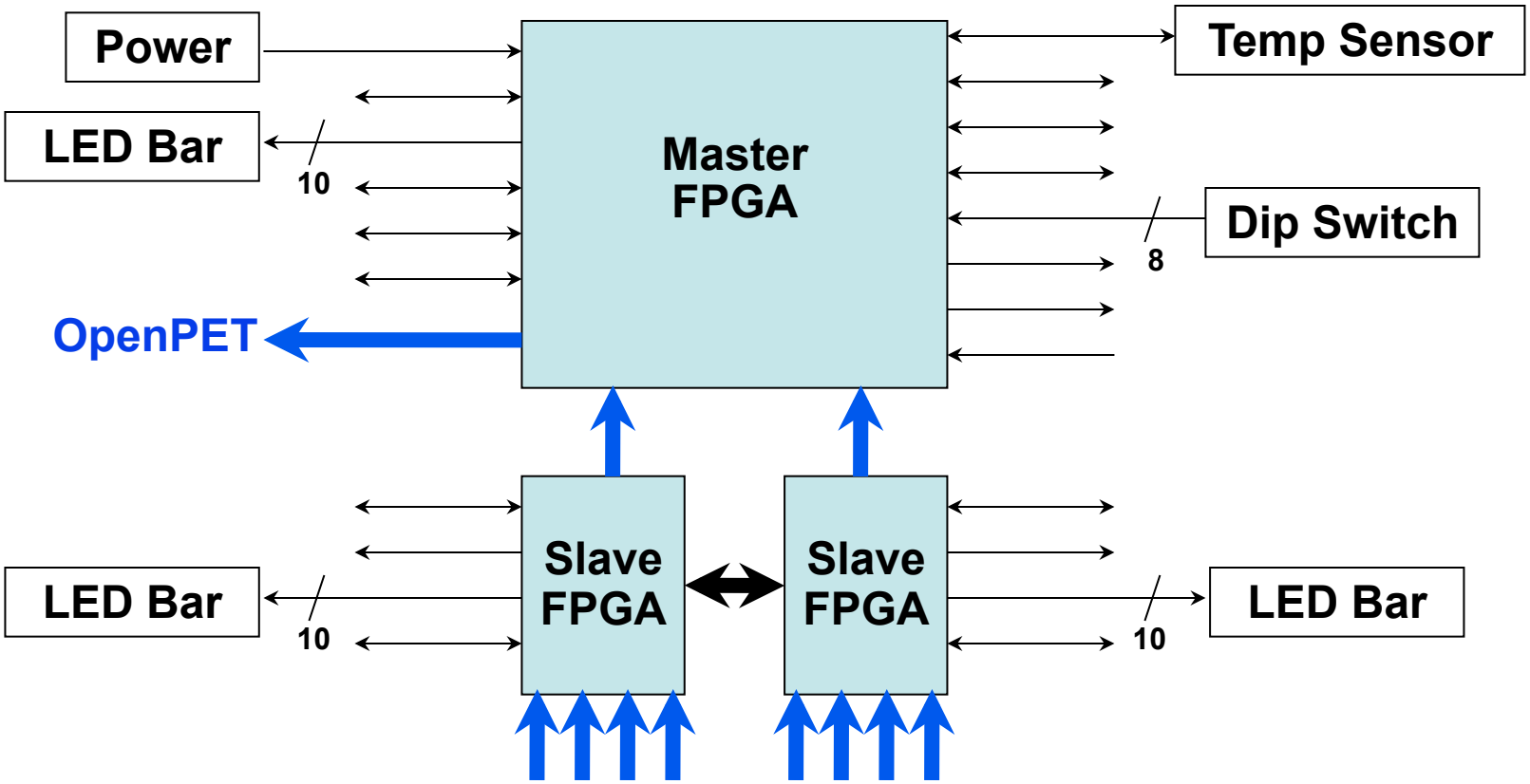
BERKELEY LAB

Page 1 of SB Schematic



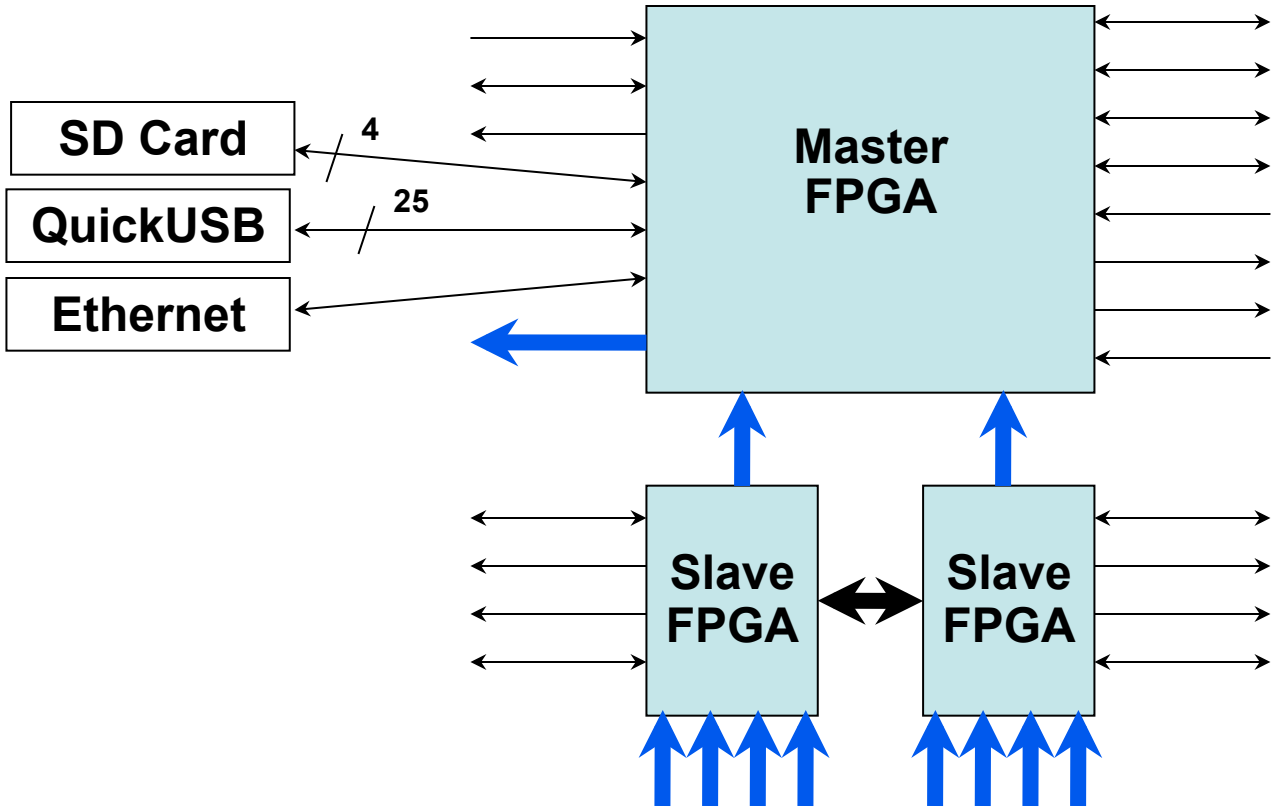
Will Use “Cartoon” of Schematic (Old Eyes…)

Self-Explanatory Connections



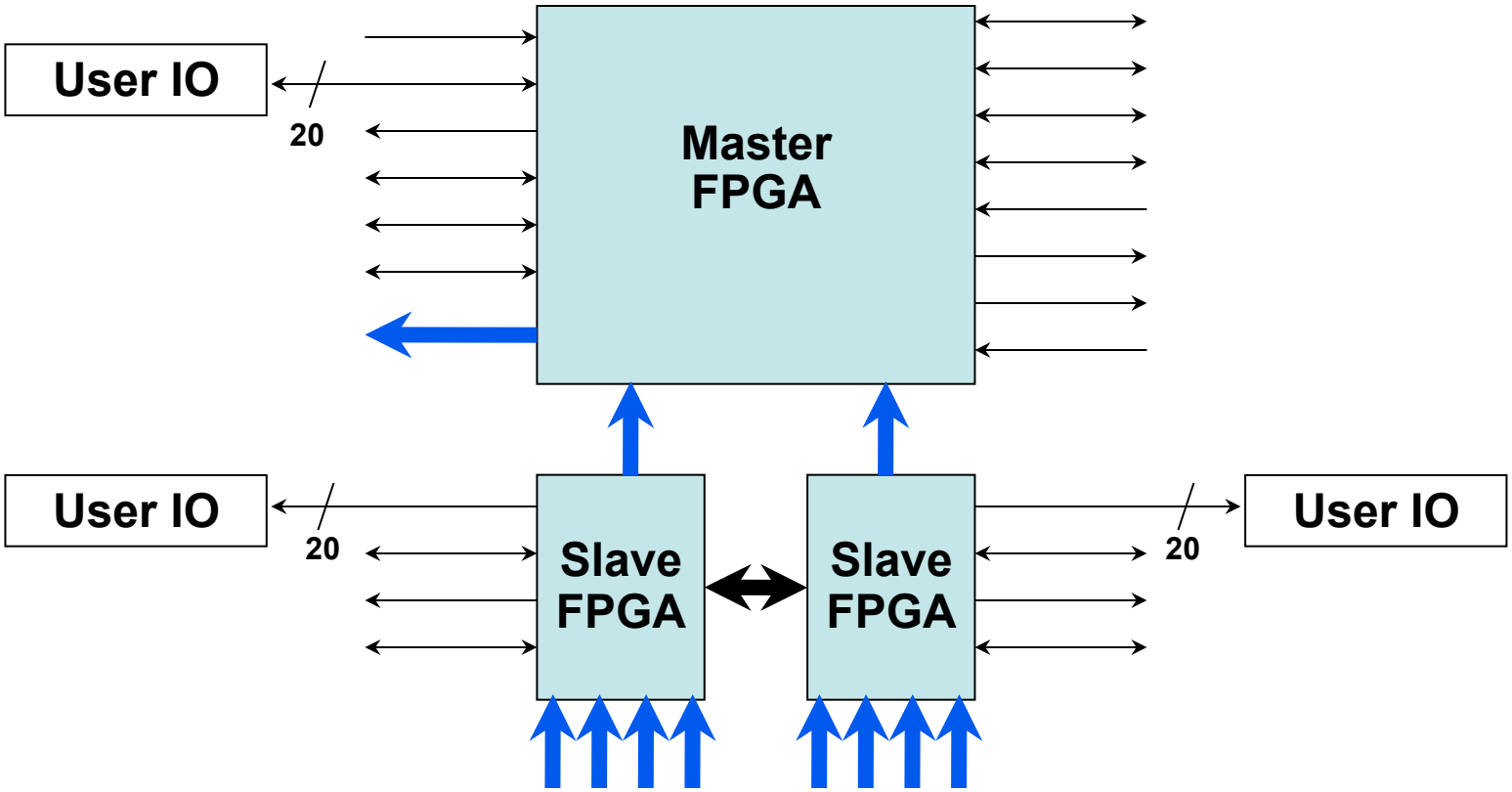
- LED Bars – User definable, no present allocation
- DIP Switch – User definable, no present function
- OpenPET Data and Control sent through FPGA, clock sent through separate conditioning / distribution circuits

Host PC Interface Board Connections



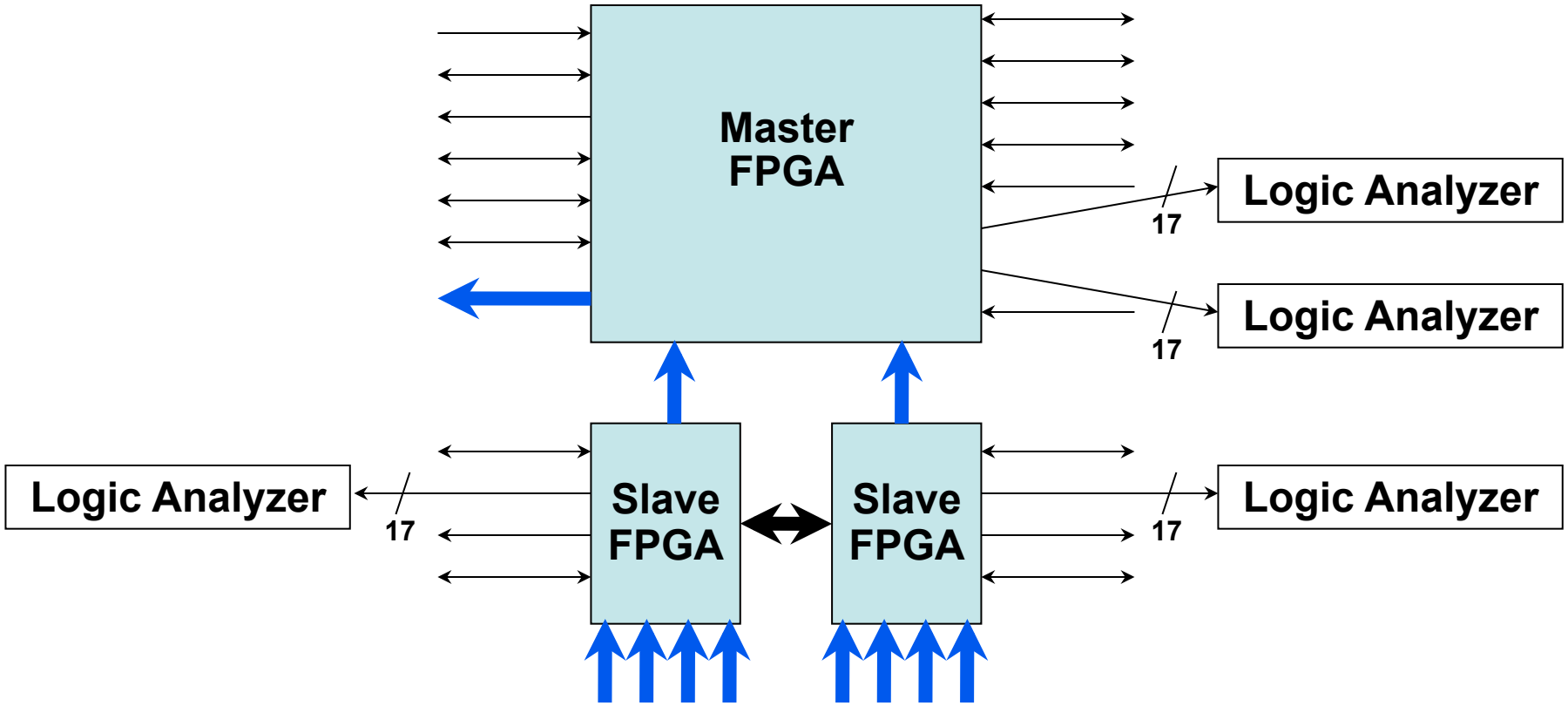
- SD Card Connector on *both* SB and Interface Board, but only one can be plugged in at any given time
- QuickUSB Connector on *both* SB and Interface Board, but only one can be plugged in at any given time
- Ethernet Connector only on Interface Board

User IO Board Connections



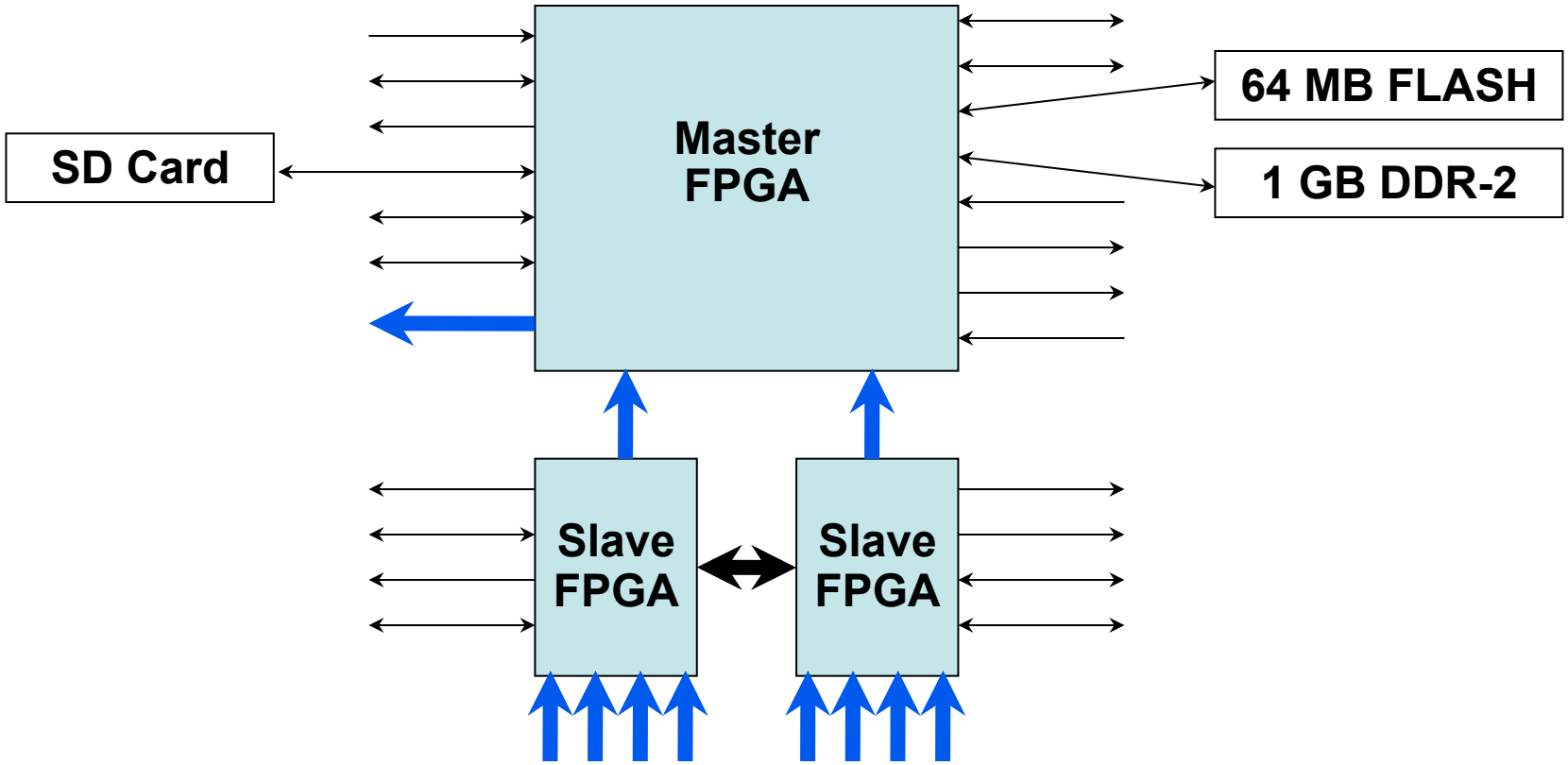
- Each FPGA has 16 data lines plus 4 direction lines
- These lines go to the User IO Board, and are buffered on that board
- The two RS-232 ports and the External Clock are not shown on the block diagram

Debugging Board Connections



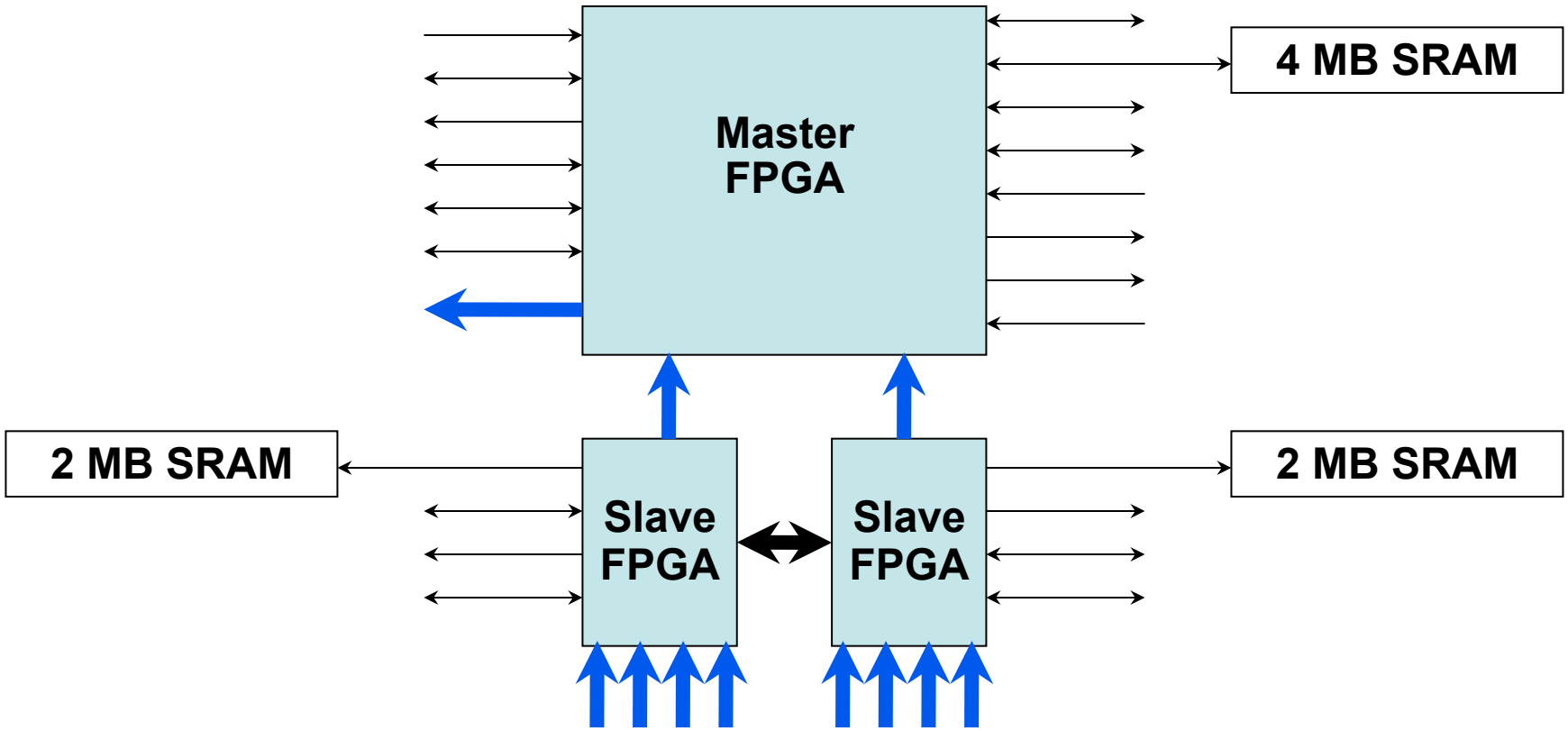
- Each Logic Analyzer connector has 16 data lines plus 1 clock line
- These lines go to the Debugging Board, and are not buffered
- JTAG connection described separately

Memory for NIOS Core (Microprocessor)



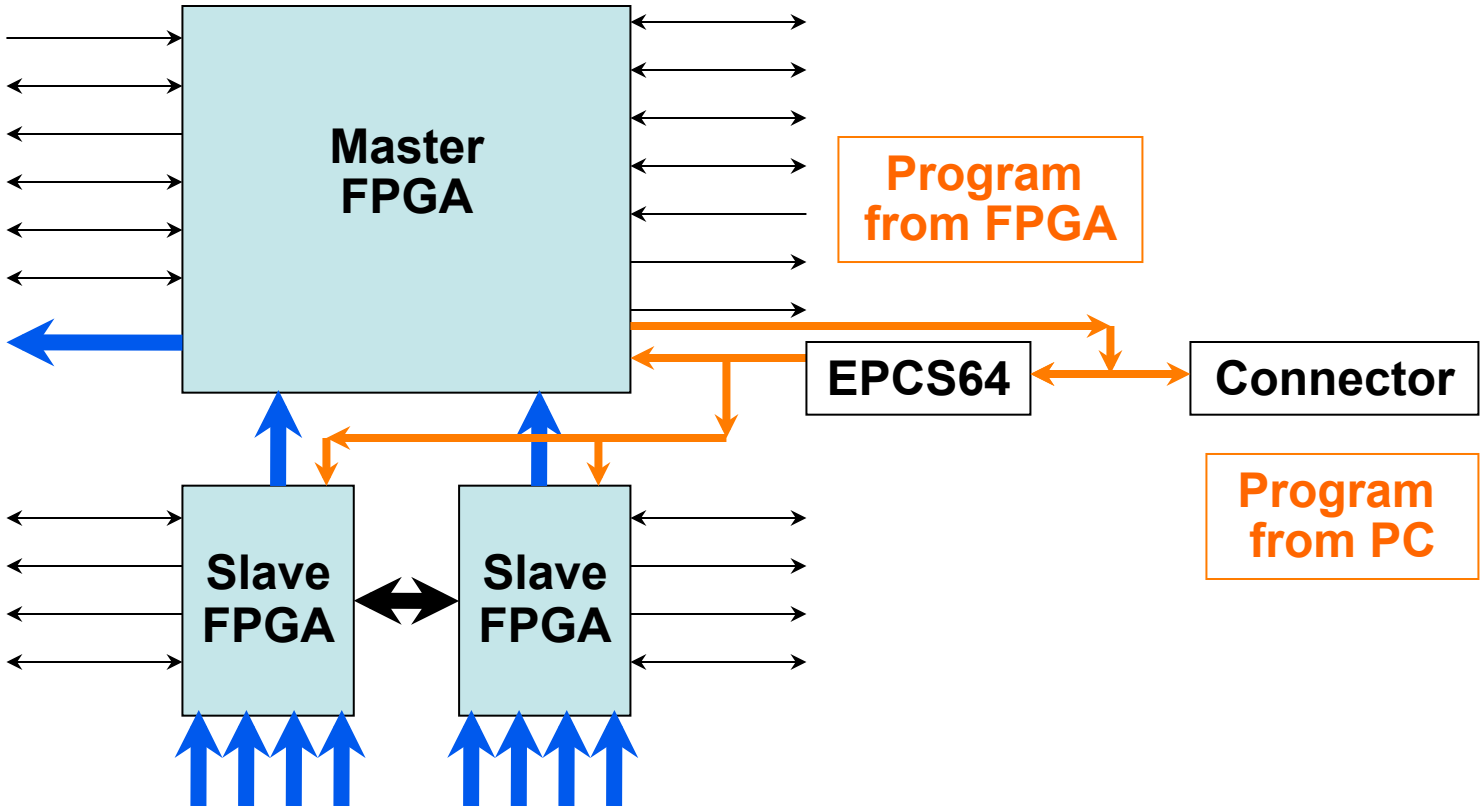
- DDR-2 used like "PC RAM memory" by NIOS Core
- SD Card is primary "PC hard disk" for NIOS Core
- FLASH is secondary "PC hard disk"
  (in case of problems with SD Card)

Memory for "VHDL" Part of FPGA



- SRAM is fast, with fixed latency (look-up time)
- Used for lookup tables, etc. for "VHDL" (real-time) FPGA code

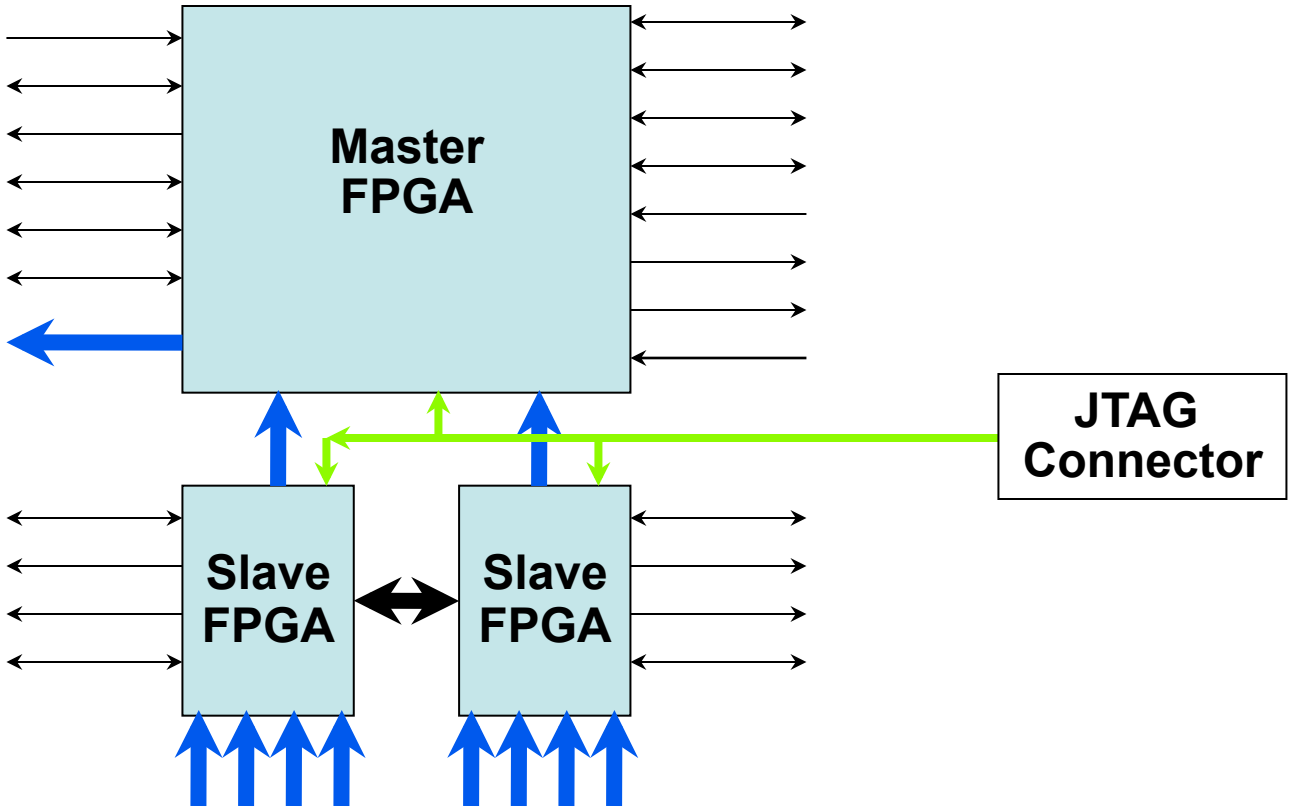
FPGA Programming Connections



- All 3 FPGAs programmed by EPCS64 Memory
- EPCS64 programmed by PC via connector on SB, or
- EPCS64 programmed directly by FPGA

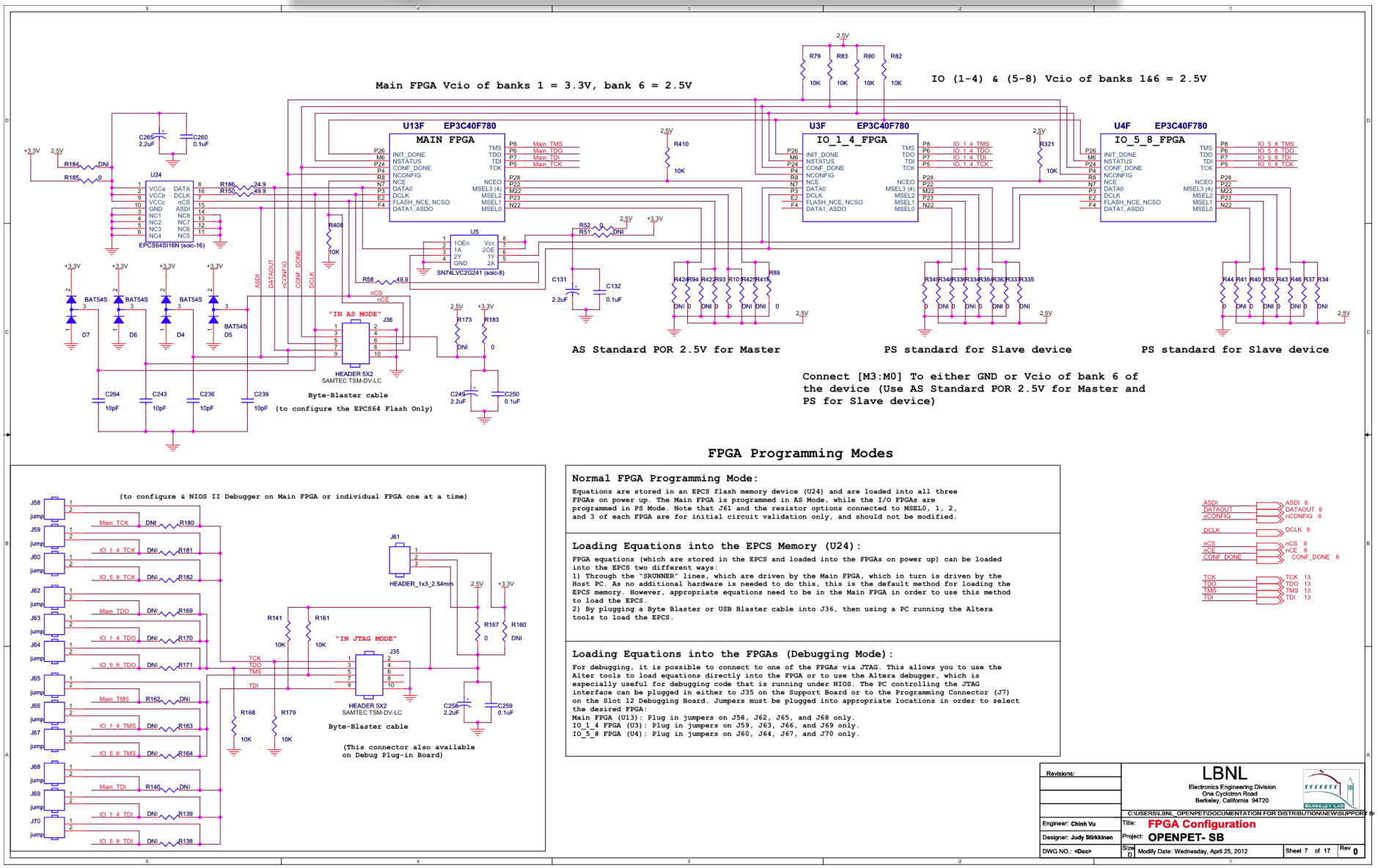
BERKELEY LAB

FPGA JTAG Connections



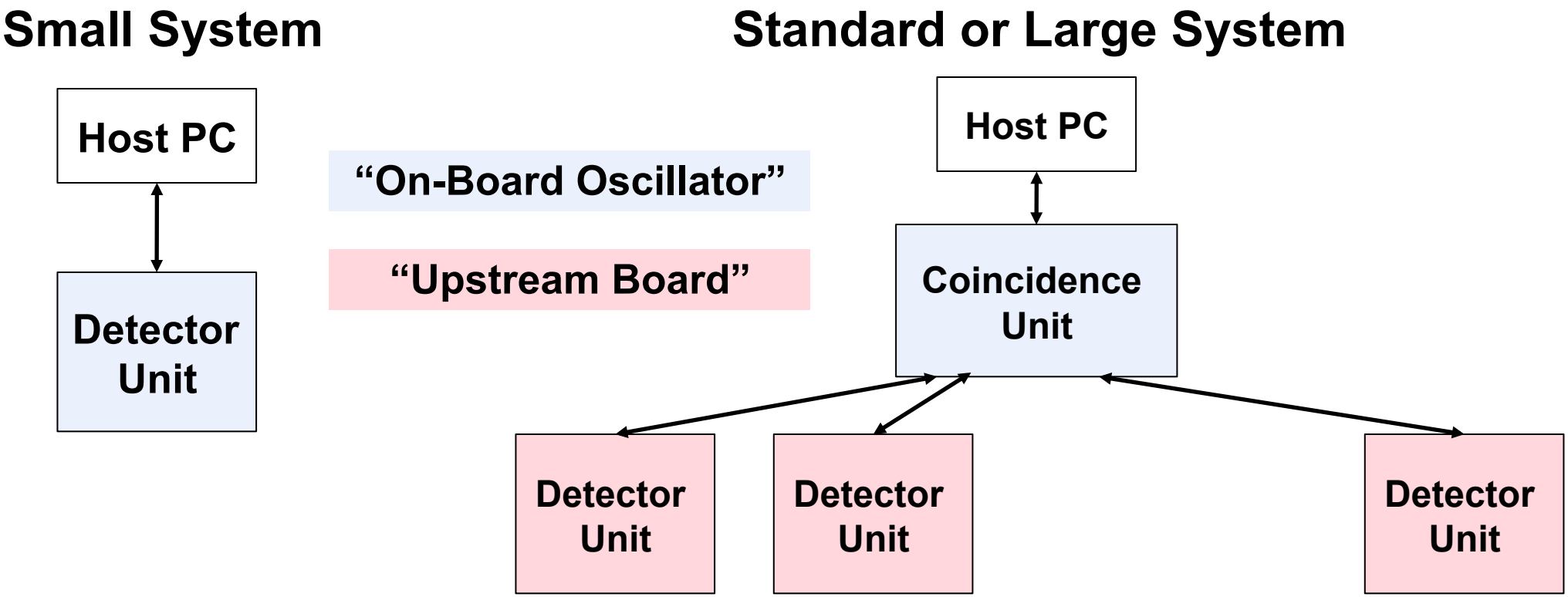
- Can load equations into *one* FPGA via JTAG connector (on Support Board or Debugging Board)
- Jumpers select *which* FPGA programmed
- Can also run NIOS debugger through JTAG

Page 7 of SB Schematic



Non-Trivial, But Detailed Instructions…

Clock Distribution



- Three Possible Sources On Each Support Board:
  1) On-Board Oscillator, 2) Upstream Board, 3) Debug Connector
- Jumpers select Source on Each SB
- "On-Board Oscillator" Should Be Used on the Top Level SB, "Upstream Board" for the Rest (Exactly 1 Oscillator per System)

The OpenPET Firmware and Software Framework

Qiyu Peng

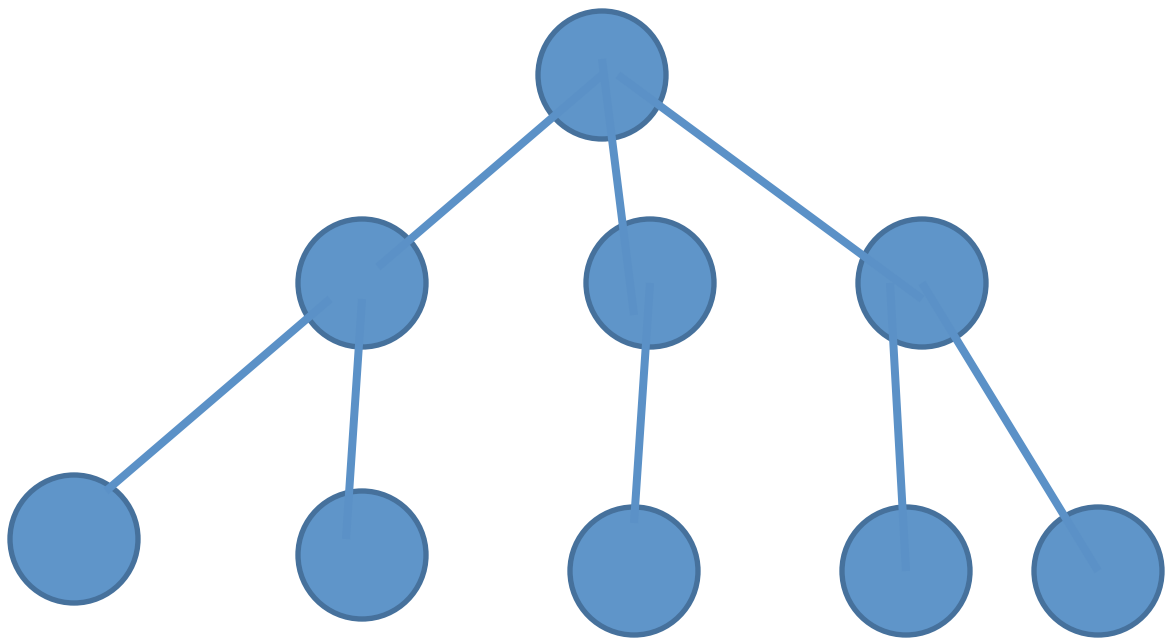
May 10-11, 2012

Requirements

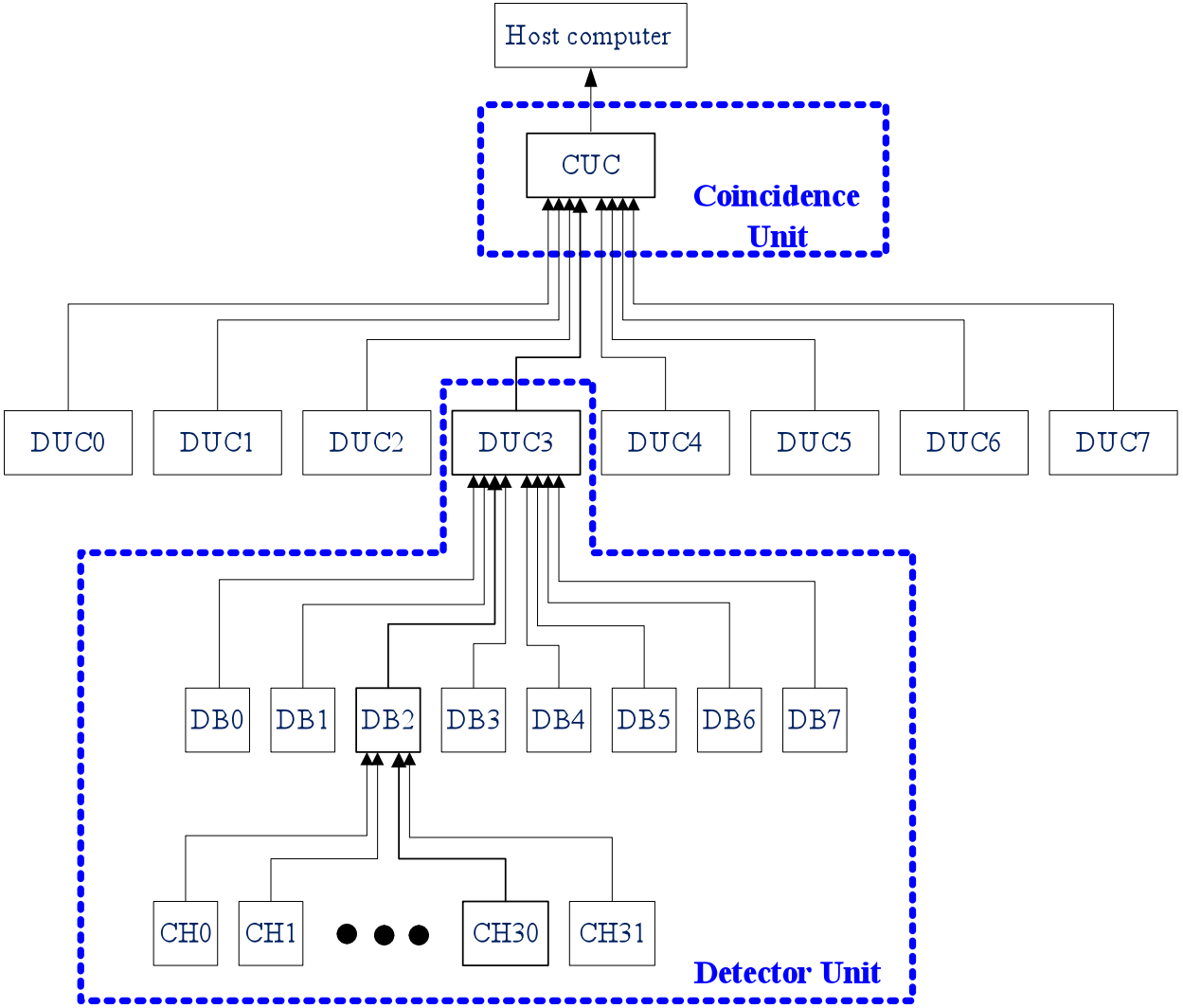
- Reliability
- Stability
- Flexibility
- Scalability
- Compatibility
- Simplicity

System Configuration

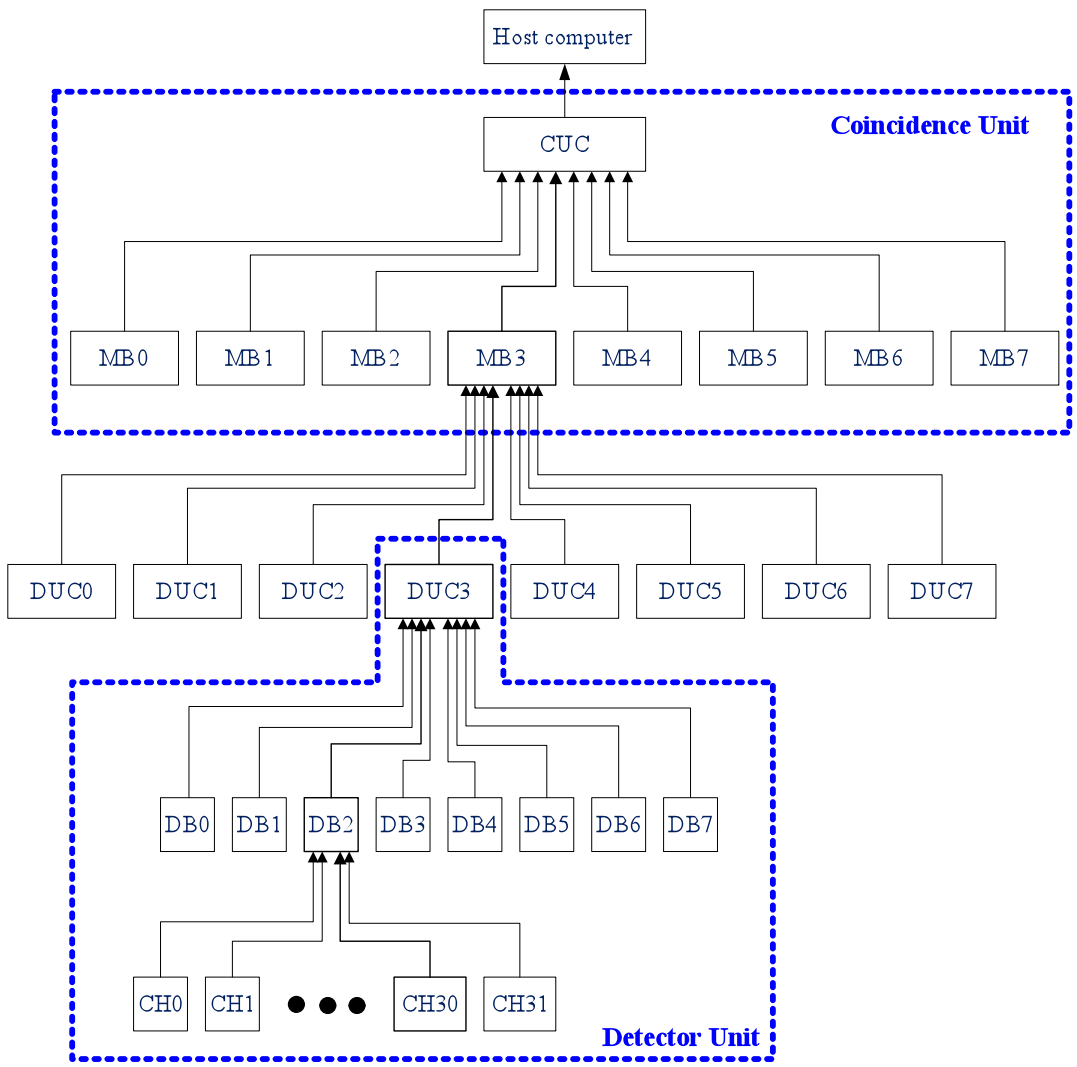
- The OpenPET system is essentially a computer network with a tree topology.



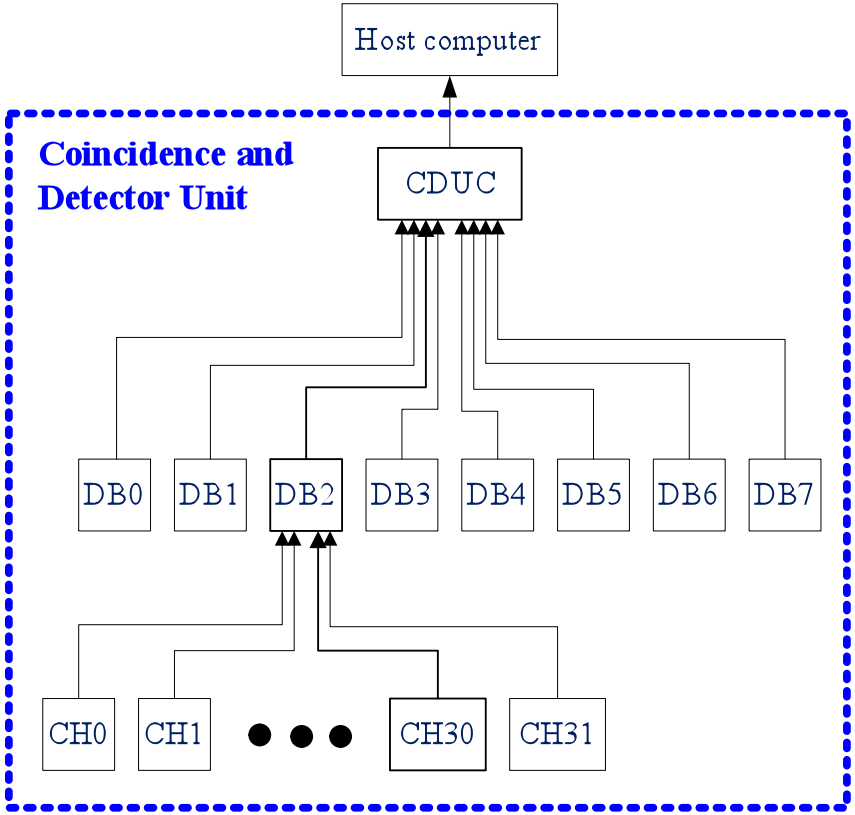
Standard system configuration



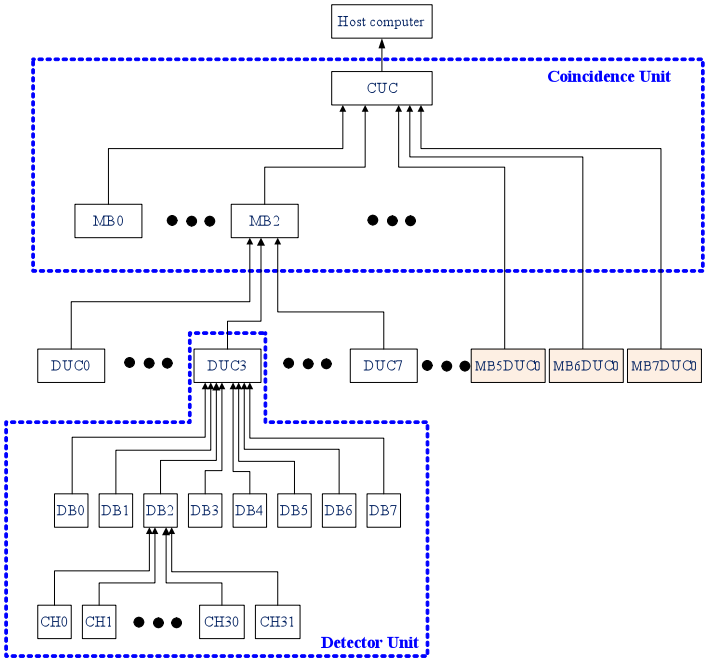
Large system configuration

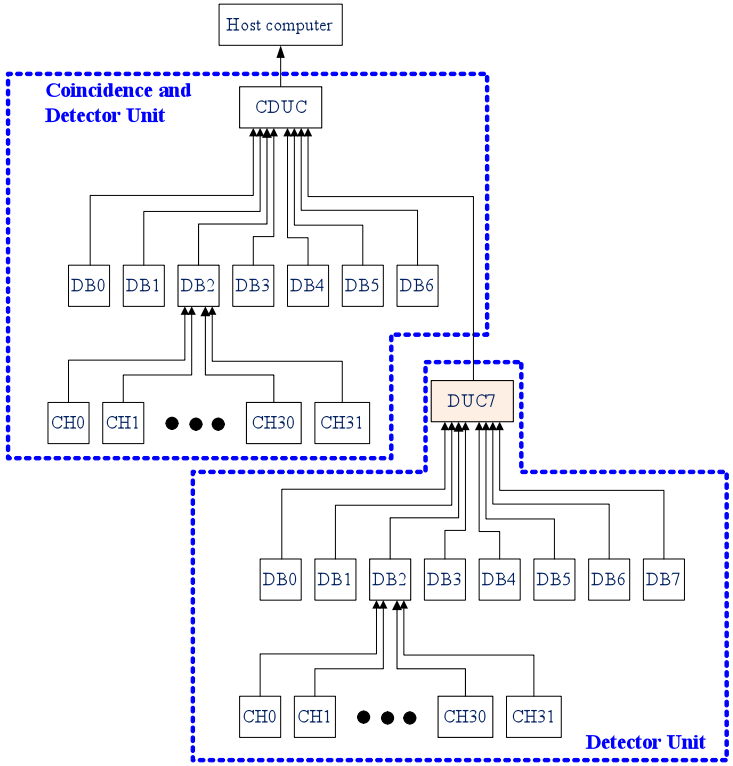


Small system configurations



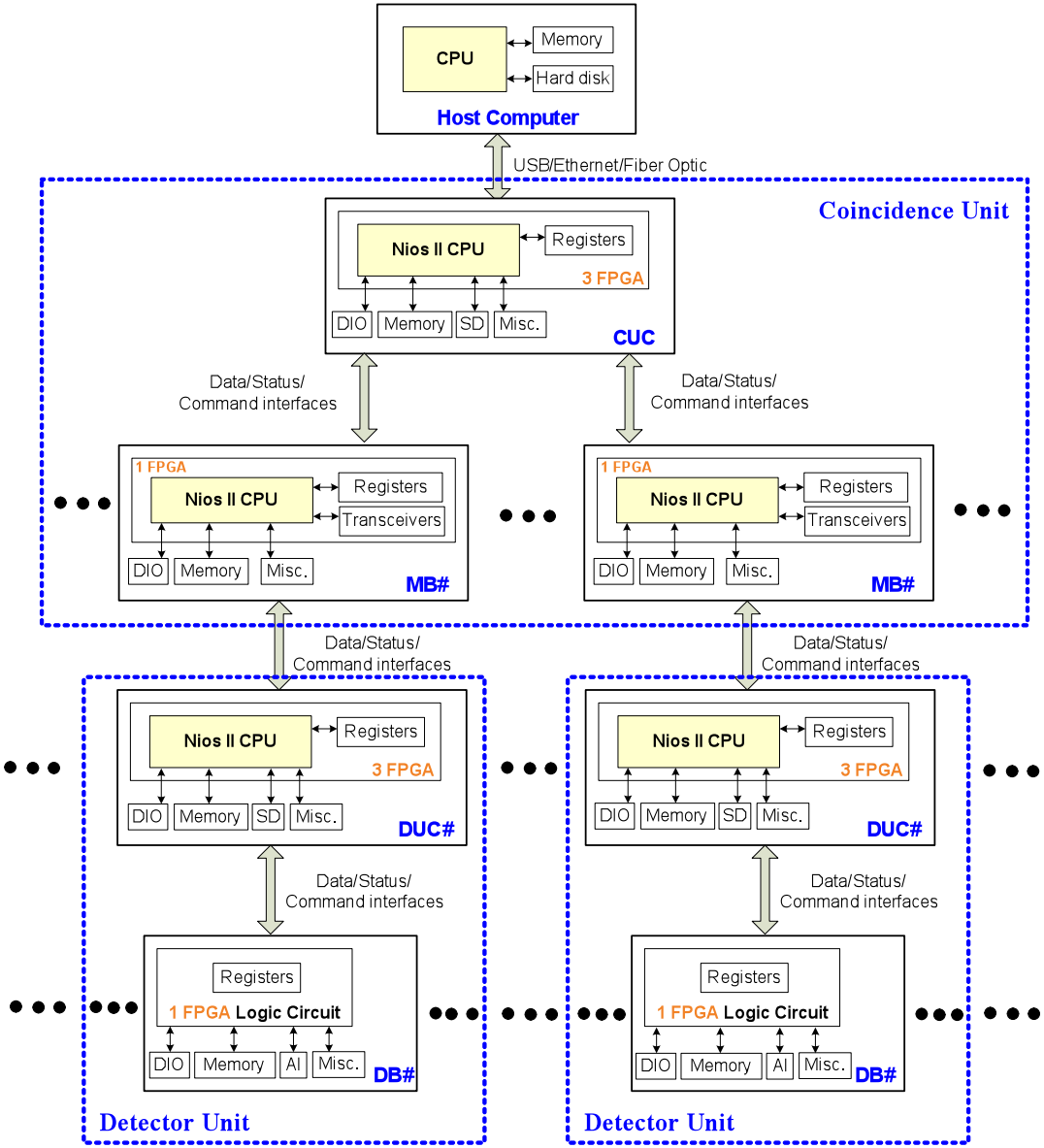
Mixed system configurations





Not recommended

System hardware structure



Digital IO

In Main FPGA:

16 bits digital IO (4 bits direction control)

10 Bits LED bar

Two logic analyzer connectors (16+1 bits each)

In IO FPGA 1:

16 bits digital IO (4 bits direction control)

10 Bits LED bar

One logic analyzer connector (16+1 bits each)

In IO FPGA 2:

16 bits digital IO (4 bits direction control)

10 Bits LED bar

One logic analyzer connector (16+1 bits each)

Memory

In Main FPGA:

Flash memory

EPCS64 flash memory

64MB flash memory

RAM

4MB SRAM

1GB DDR2-SO-DIMM SDRAM

In IO FPGA 1:

RAM

2MB SRAM

In IO FPGA 2:

RAM

2MB SRAM

SD card

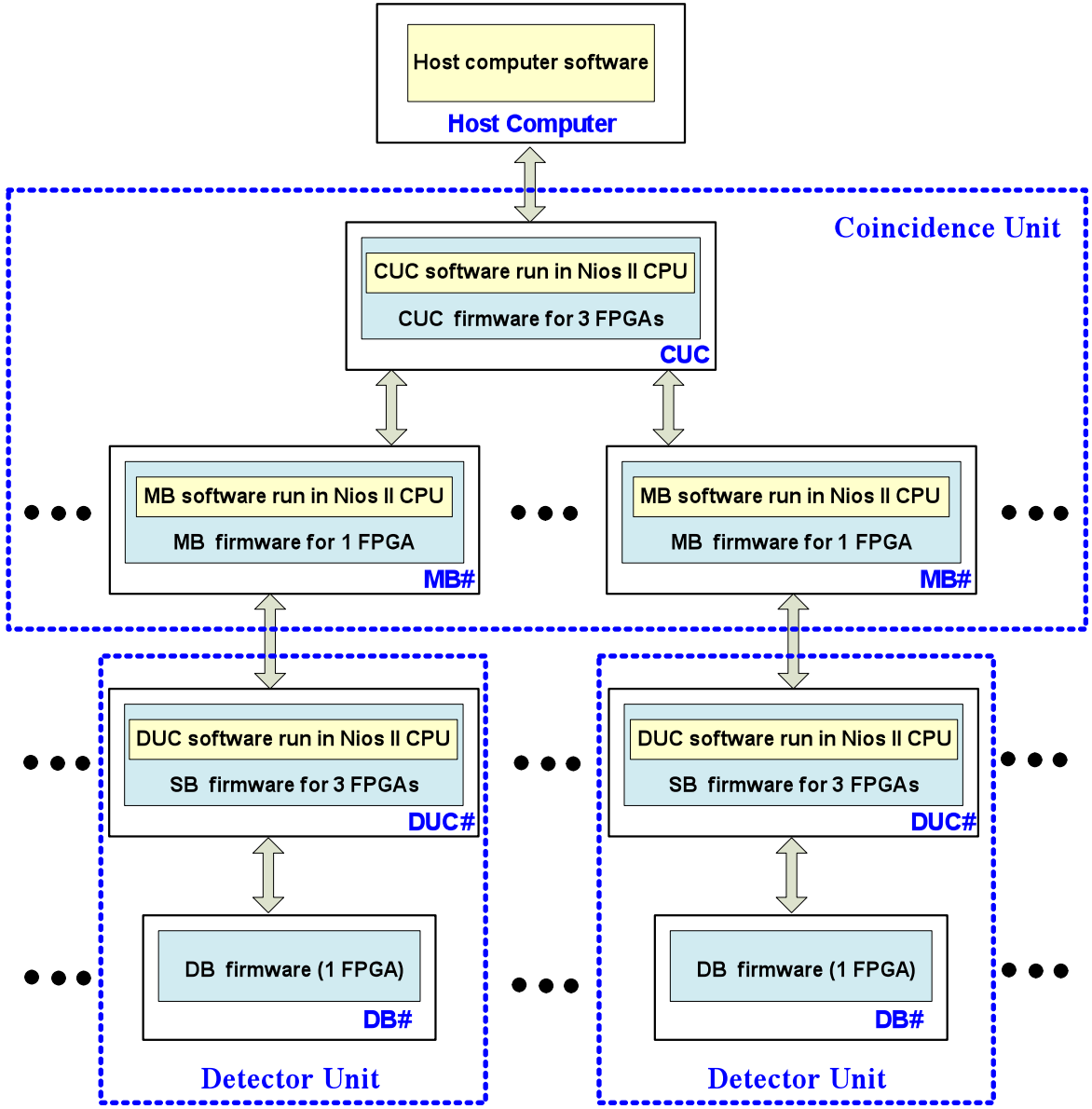
Miscellaneous devices

Power monitor

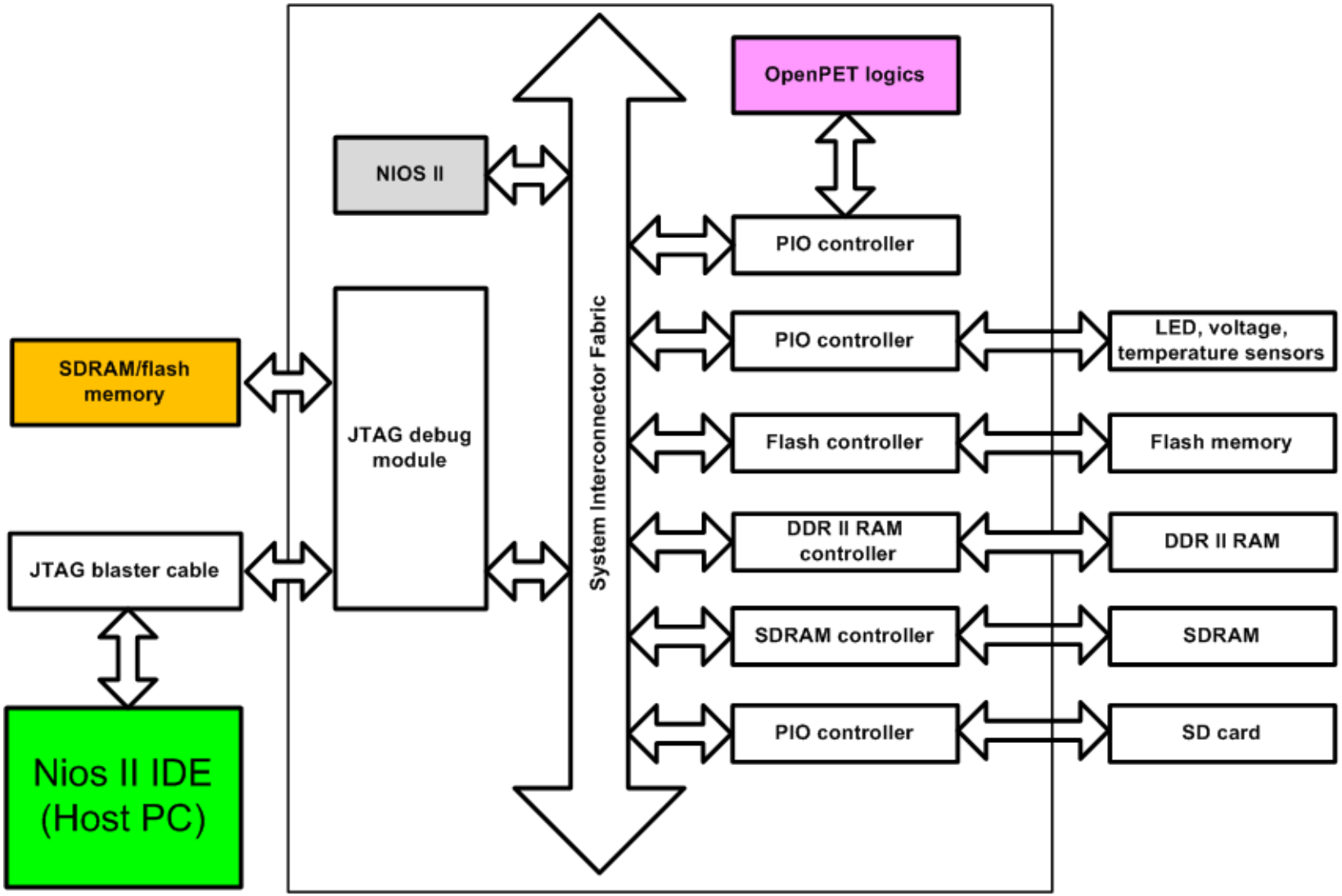
Temperature sensors

Two RS232 interfaces

Firmware and software structure



Brief Introduction to NIOS



Brief Introduction to NIOS

NIOS	PC
The real-time micro-processor with FPU	CPU
Volatile memory	RAM
Non-volatile memory	Hard disk
Peripherals (analog and digital IO)	Parallel, serial IO and etc
None or Nios II IDE running in the Host PC	Key board and monitor

Programming Tools / Environment

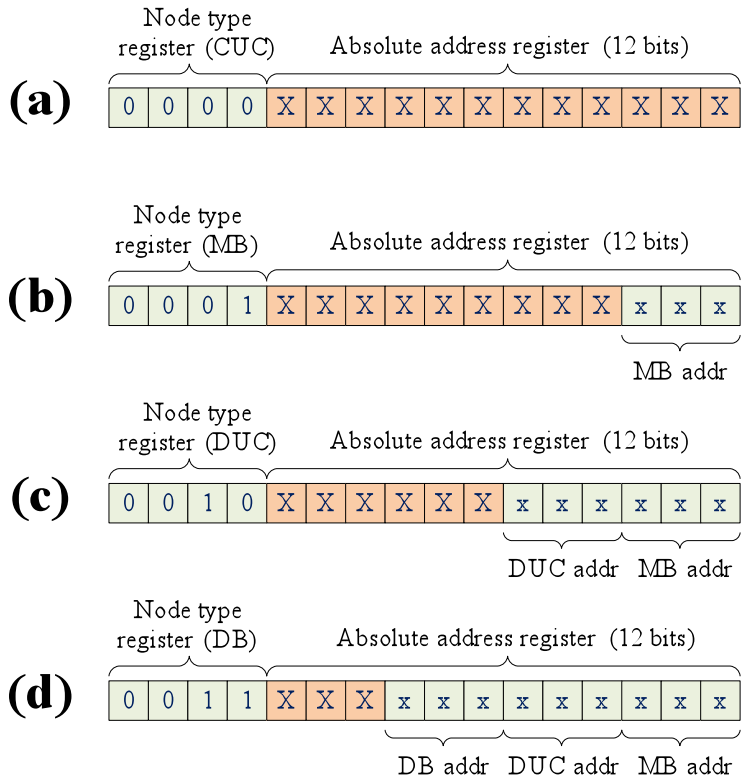
	DB	DUC	MB	CUC	Host PC
FPGA Firmware	X	X	X	X	
Embedded Microprocessor Software		X	X	X	
PC Software					X

System addressing strategies

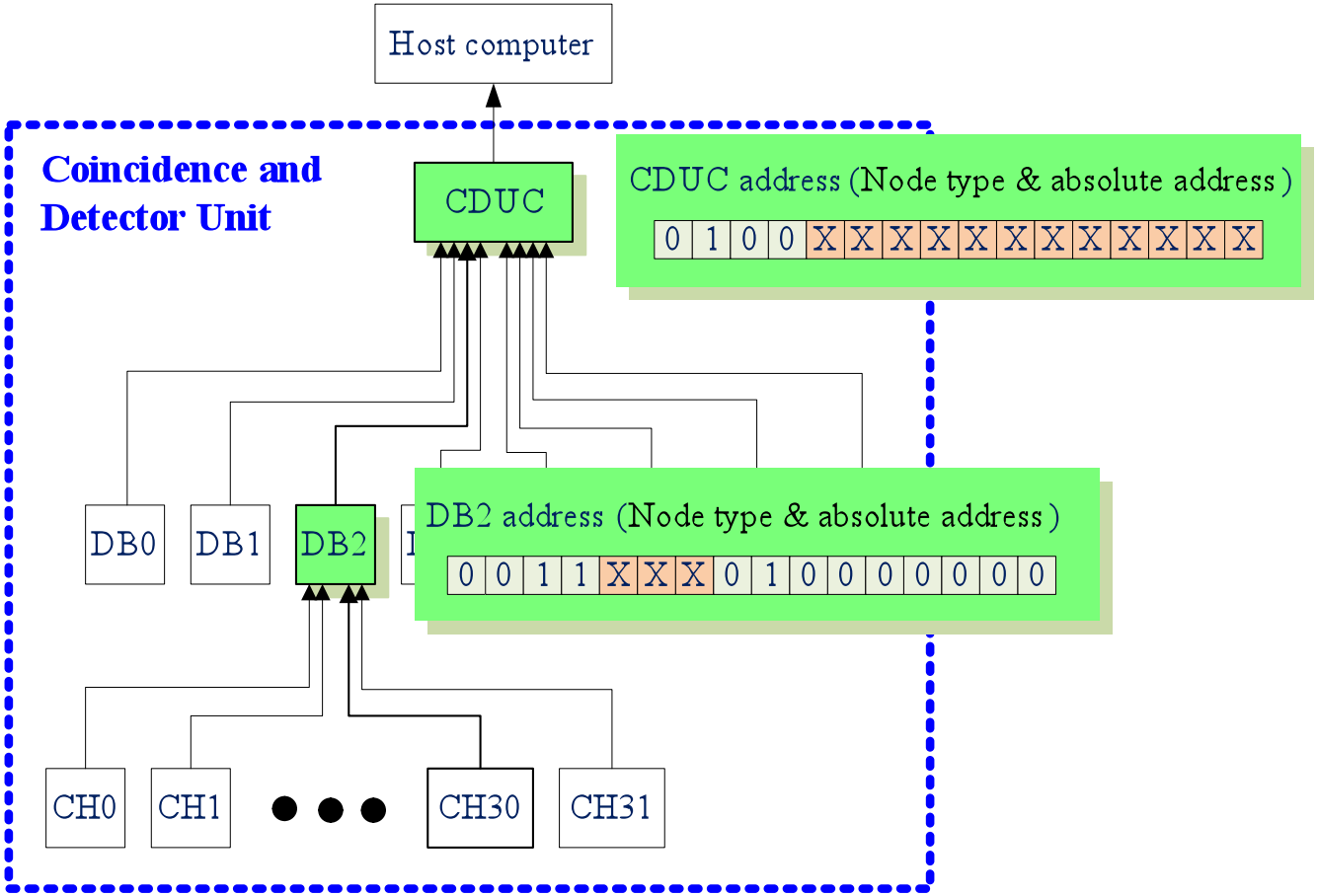
- ***System configuration profile (***host computer ***)***
- A tree data structure of the whole system
- A data structure contains details of all nodes in the tree (node types, absolute addresses, connection status and etc.)
- Integrity of all the hardware devices connected to all the nodes

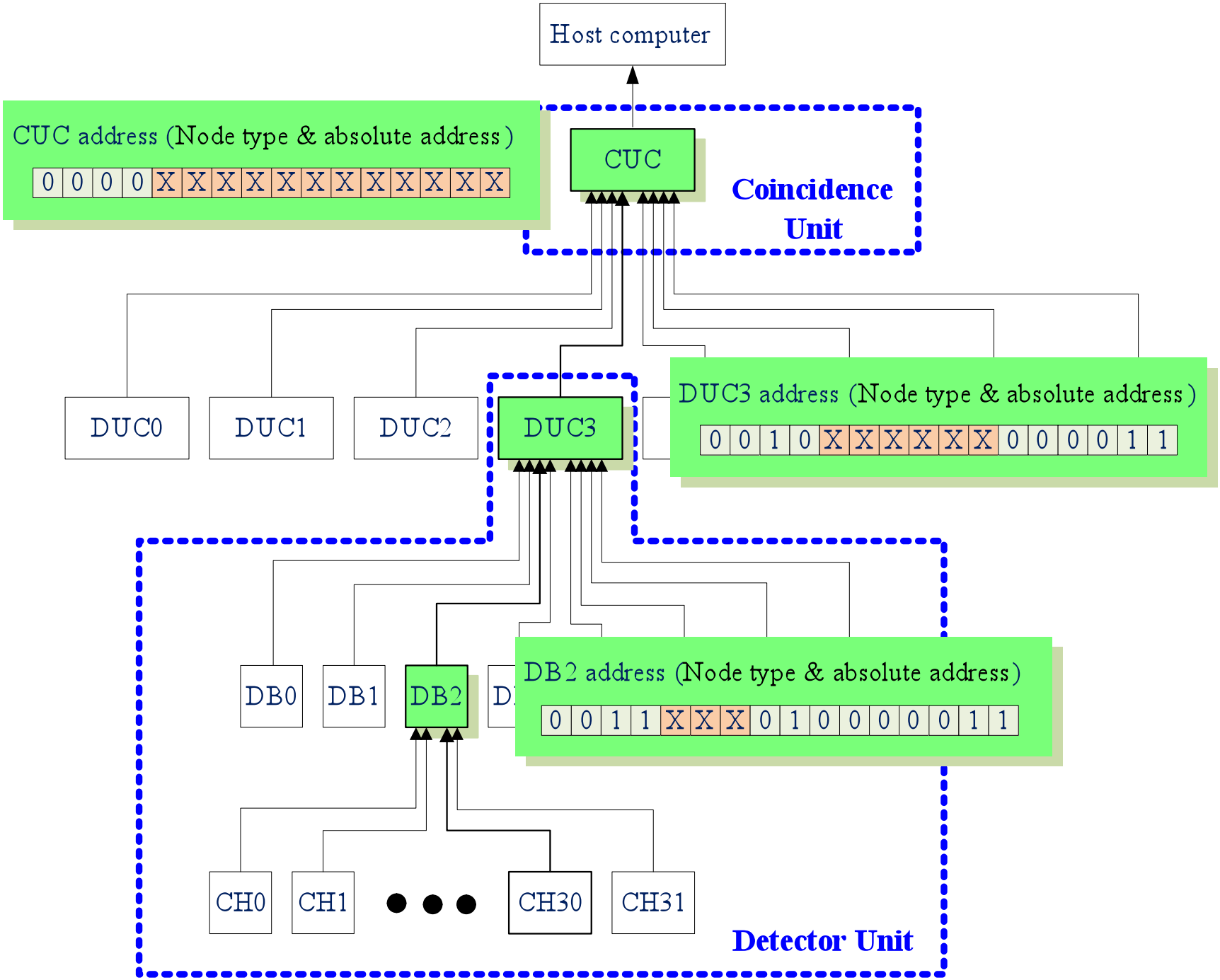
System addressing strategies

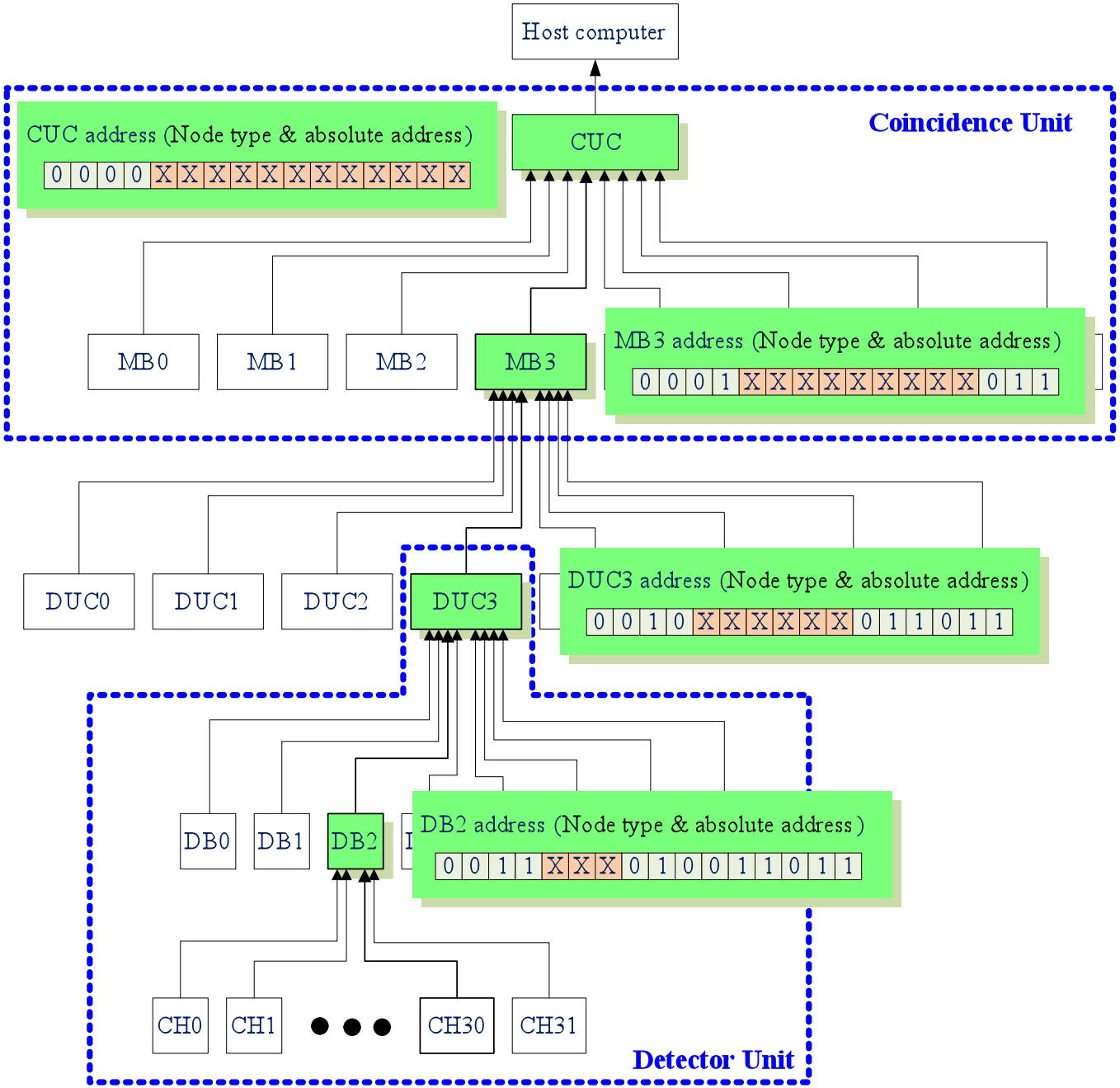
- ***Node type register and absolute address register***



Examples


Host computer
Coincidence and Detector Unit
CDUC
CDUC address (Node type & absolute address)
0 1 0 0 X X X X X X X X X X X X
DB0
DB1
DB2
DB2 address (Node type & absolute address)
0 0 1 1 X X X 0 1 0 0 0 0 0 0 0
CH0
CH1
CH30
CH31


Host computer
CUC address (Node type & absolute address)
0 0 0 0 X X X X X X X X X X X X
CUC
Coincidence Unit
DUC0
DUC1
DUC2
DUC3
DUC3 address (Node type & absolute address)
0 0 1 0 X X X X X X 0 0 0 0 1 1
DB0
DB1
DB2
DB2 address (Node type & absolute address)
0 0 1 1 X X X 0 1 0 0 0 0 0 1 1
CH0
CH1
CH30
CH31
Detector Unit


Host computer
Coincidence Unit
CUC address (Node type & absolute address)
0 0 0 0 X X X X X X X X X X X X
CUC
MB0
MB1
MB2
MB3
MB3 address (Node type & absolute address)
0 0 0 1 X X X X X X X X X 0 1 1
DUC0
DUC1
DUC2
DUC3
DUC3 address (Node type & absolute address)
0 0 1 0 X X X X X X 0 1 1 0 1 1
DB0
DB1
DB2
DB2 address (Node type & absolute address)
0 0 1 1 X X X 0 1 0 0 1 1 0 1 1
CH0
CH1
CH30
CH31
Detector Unit

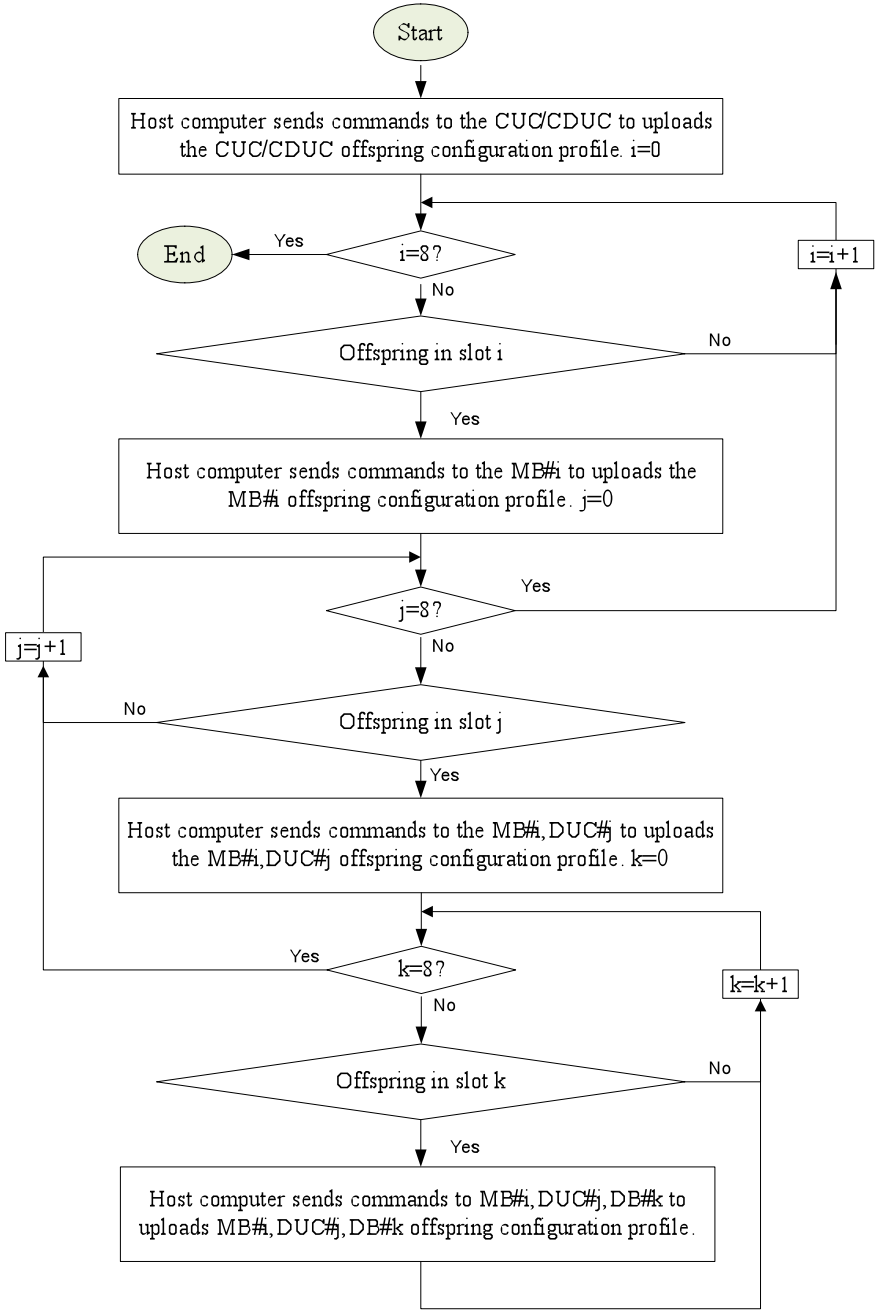
System addressing strategies

- ***Offspring configuration profile***

- Offspring Connection Status
- Offspring Enable/disable Status
- Offspring's node types and absolute addresses

Absolute address assignment strategy

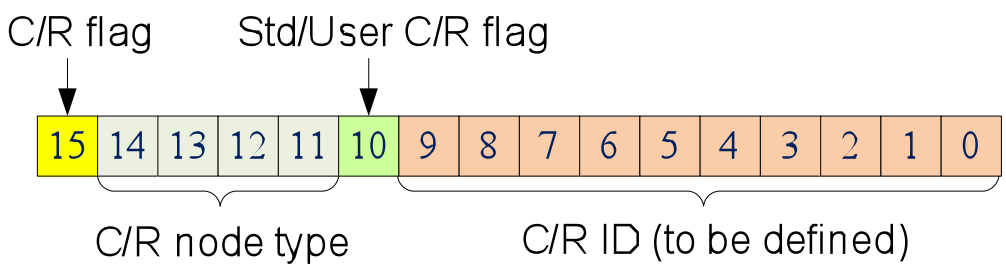
- Step 1: assign absolute addresses to all nodes connected to the system (or establish Offspring configuration profile)
- Step 2: Host PC read offspring configuration profile from all nodes and establish system configuration profile


Start
Host computer sends commands to the CUC/CDUC to uploads the CUC/CDUC offspring configuration profile. i=0
i=8?
Yes
End
No
i=i+1
Offspring in slot i
No
Yes
Host computer sends commands to the MB#i to uploads the MB#i offspring configuration profile. j=0
j=8?
Yes
No
j=j+1
Offspring in slot j
No
Yes
Host computer sends commands to the MB#i, DUC#j to uploads the MB#i, DUC#j offspring configuration profile. k=0
k=8?
Yes
No
k=k+1
Offspring in slot k
No
Yes
Host computer sends commands to MB#i, DUC#j, DB#k to uploads MB#i, DUC#j, DB#k offspring configuration profile.

System commands and responses

- ***Format of system commands and responses***

CMD/Response ID (16 bits)	Source Address (16 bits)	Target Address (16 bits)	Data Payload (1~256bytes)	CRC (8 bits)



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

List of Commands/responses for CUC, MB, DB, CDUC nodes

- **Low level hardware devices control commands**
- **System configuration commands**
- **High-level application-specified commands**

List mode data

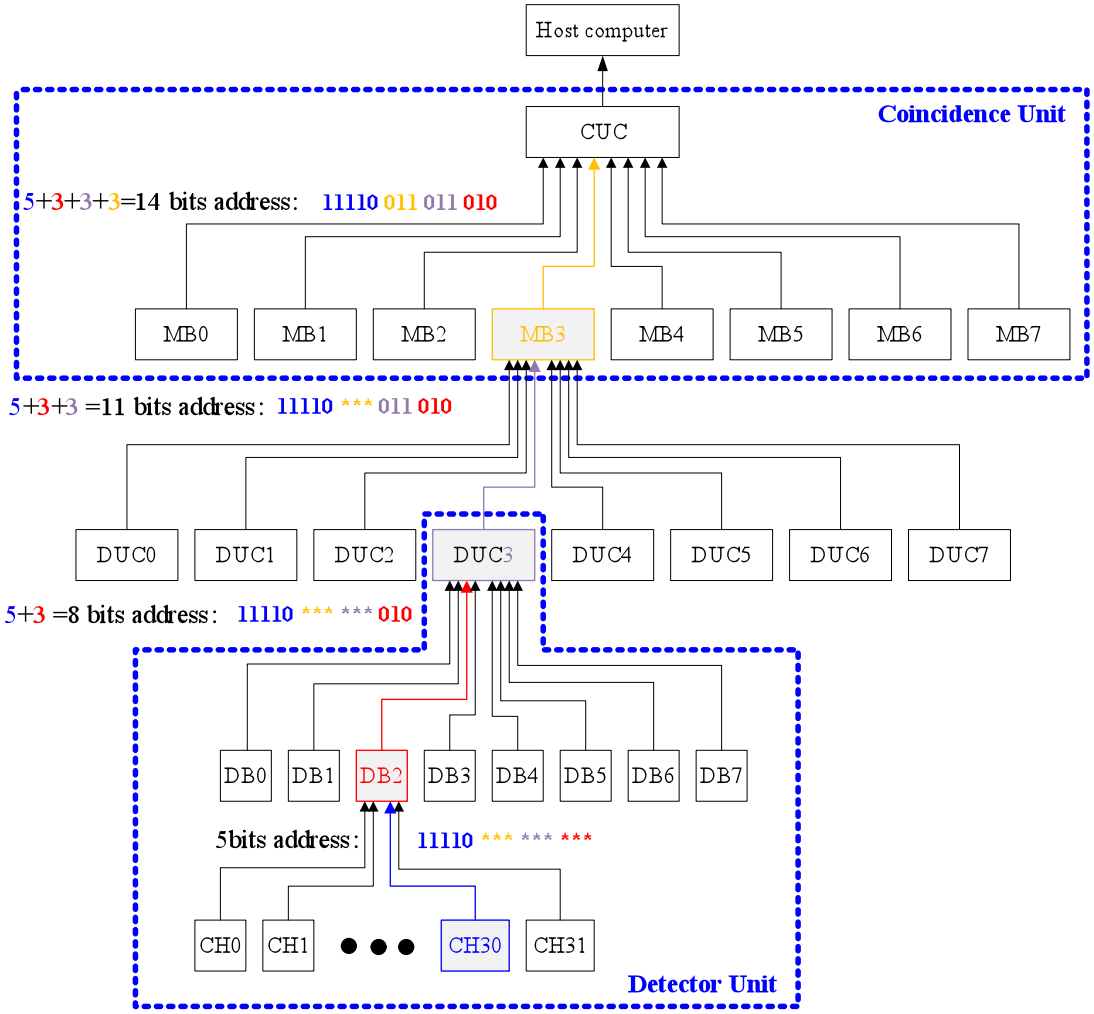
- ***Data addressing strategies***

In the 80-bit OpenPET list mode data, there are 22 bits allocated for addressing.

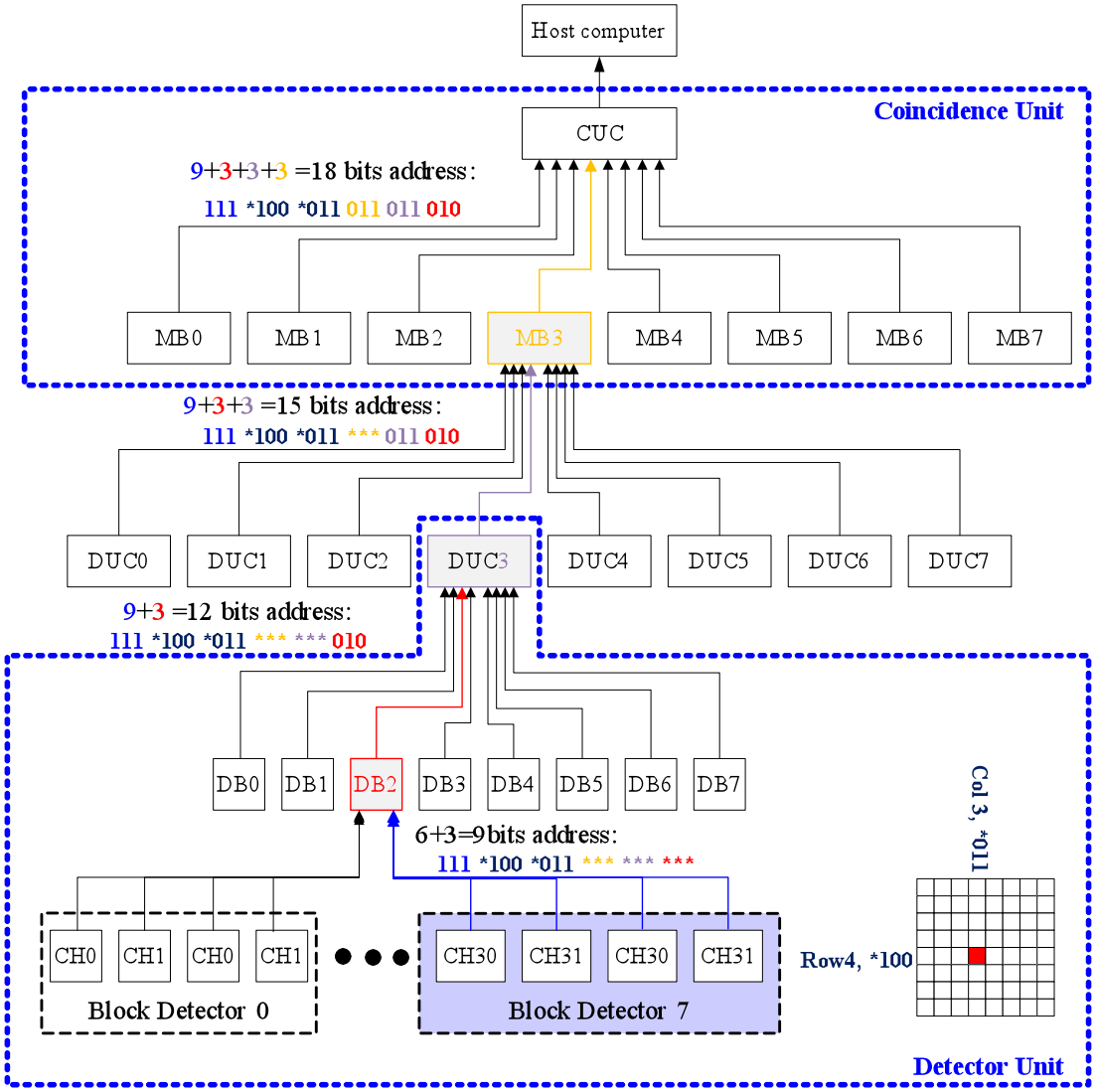
- Among them, 14 bits are standard address bits.
- an extra 8-bit address is reserved and can be redefined by the OpenPET users.

00: individual channel data addressing mode
01: crystal data addressing mode
10~11: user defined data addressing mode

Individual channel data addressing mode (00)



Crystal data addressing mode



List mode data format

- ***Coincidence events data format***

Bit 79: 1 (1: coincidence event data format flag)
Bit 78~76: MB address bits (3 bits)
Bit 75~73: DUC address bits (3 bits)
Bit 72~70: DB address bits (3 bits)
Bit 69~65: Channel address bits (5 bits)
Bit 64~57: User defined address bits (8 bits)
Bit 56~54: DOI data bits (3 bits)
Bit 53~42: TDC data bits (12 bits, LSB: 25ps)
Bit 41~40: unused bits (2 bits)
Bit 39: valid bit (1: valid coincidence data; 0: invalid coincidence data)
Bit 38~36: MB address bits (3 bits)
Bit 35~33: DUC address bits (3 bits)
Bit 32~30: DB address bits (3 bits)
Bit 29~25: Channel address bits (5 bits)
Bit 24~17: User defined address bits (8 bits)
Bit 16~14: DOI data bits (3 bits)
Bit 13~2: TDC data bits (12 bits, LSB: 25ps)
Bit 1~0: unused bits (2 bits)

Single events data format

Time Mode

Bit 79: 0 (0: not coincidence event data format)
Bit 78: 1 (1: single event data format)
Bit 77~73: 00000 (5 bits) (time mode)
Bit 72~40: unused bits (33 bits)
Bit 39: valid bit (1: valid single event data; 0: invalid single event data)
Bit 38~36: MB address bits (3 bits)
Bit 35~33: DUC address bits (3 bits)
Bit 32~30: DB address bits (3 bits)
Bit 29~25: Channel address bits (5 bits)
Bit 24~17: User defined address bits (8 bits)
Bit 16~14: DOI data bits (3 bits)
Bit 13~2: TDC data bits (12 bits, LSB: 25ps)
Bit 1~0: unused bits (2 bits)

Single events data format

Energy mode

Bit 79: 0 (0: not coincidence event data format)
Bit 78: 1 (1: single event data format flag)
Bit 77~73: 00001 (5 bits) (energy mode)
Bit 72~40: unused bits (33 bits)

Bit 39: valid bit (1: valid single event data; 0: invalid single event data)
Bit 38~36: MB address bits (3 bits)
Bit 35~33: DUC address bits (3 bits)
Bit 32~30: DB address bits (3 bits)
Bit 29~25: Channel address bits (5 bits)
Bit 24~17: User defined address bits (8 bits)
Bit 16~14: DOI data bits (3 bits)
Bit 13~2: Energy data bits (12 bits)
Bit 1~0: unused bits (2 bits)

Single events data format

Raw ADC data mode

Bit 79: 0 (0: not coincidence event data format)
Bit 78: 1 (1: single event data format flag)
Bit 77~73: 00010 (5 bits) (Raw ADC data mode)
Bit 72~70: (3 bits)
000: Raw ADC data from a single channel;
001: Raw ADC data from all 32 channels;
010~111: unused
Bit 69~59: total ADC data count (11 bits) (maximum: 32 channels * 64 samples = 2048)
Bit 58~48: current ADC data count (11 bits)
Bit 47~40: Reserved bits (8 bits)

Bit 39: valid bit (1: valid single event data; 0: invalid single event data)
Bit 38~36: MB address bits (3 bits)
Bit 35~33: DUC address bits (3 bits)
Bit 32~30: DB address bits (3 bits)
Bit 29~25: Channel address bits (5 bits)
Bit 24~17: User defined address bits (8 bits)
Bit 16~14: DOI data bits (3 bits)
Bit 13~2: Raw ADC data bits (12 bits)
Bit 1~0: Unused bits (2 bits)

Single events data format

Standard Anger-logic mode

Bit 79: 0 (0: not coincidence event data format)
Bit 78: 1 (1: single event data format flag)
Bit 77~73: 00011 (5 bits) (Standard Anger-logic mode)
Bit 72~40: unused bits (33 bits)

Bit 39: valid bit (1: valid single event data; 0: invalid single event data)
Bit 38~36: MB address bits (3 bits)
Bit 35~33: DUC address bits (3 bits)
Bit 32~30: DB address bits (3 bits)
Bit 29~25: Channel address bits (5 bits)
Bit 24~17: User defined address bits (8 bits)
Bit 16: unused bits (1 bits)
Bit 15~8: X (8 bits)
Bit 7~0: Y (8 bits)

Single events data format

Test modes

Bit 79: 0 (0: not coincidence event data format)
Bit 78: 1 (1: single event data format)
Bit 77~73: 00100 (5 bits) (Test mode1)
Bit 72~40: 0101010101010101010101010101010101010 (33 bits)
Bit 39~0: 0101010101010101010101010101010101010101010101(40 bits)

Single events data format

Reserved modes (10 modes)

Bit 79: 0 (0: not coincidence event data format)
Bit 78: 1 (1: single event data format flag)
Bit 77~73: 00101~01111 (5 bits)
Bit 72~0: to be defined

User defined modes (16 modes)

Bit 79: 0 (0: not coincidence event data format)
Bit 78: 1 (1: single event data format flag)
Bit 77~73: 10000~11111 (5 bits)
Bit 72~0: user defined

Single events data format

Status words format

Time status word
Bit 79: 0 (0: not coincidence event data format)
Bit 78: 0 (0: status words format flag)
Bit 77~73: 00000 (5 bits) (Time word format)
Bit 72~39: 1us timer (34 bits)
Bit 38~0: reserved (39 bits)

Event rate status word
Bit 79: 0 (0: not coincidence event data format)
Bit 78: 0 (0: status words format flag)
Bit 77~73: 00001 (5 bits) (event rate format)
Bit 72~40: reserved (33 bits)
Bit 39~0: total number of events (40 bits)

Single events data format

Status words format

Temperature status word (3 digits, range from 0.1°C ~99.9°C)

Bit 79: 0 (0: not coincidence event data format)
Bit 78: 0 (0: status words format flag)
Bit 77~73: 00010 (5 bits) (temperature format)
Bit 72~71: XX (2 bits) (00: CUC temperature; 01: MB temperature; 10: DUC temperature; 11: DB temperature)
Bit 70~68: MB address bits (3 bits)
Bit 67~65: DUC address bits (3 bits)
Bit 64~623: DB address bits (3 bits)
Bit 61~58: x10 temperature (4 bits ASIC code)
Bit 57~54: x1 temperature (4 bits ASIC code)
Bit 53~50: x0.1 temperature (4 bits ASIC code)
Bit 49~0: reserved (50 bits)

Single events data format

Status words format

Reserved status words (13 modes)
Bit 79: 0 (0: not coincidence event data format)
Bit 78: 0 (0: status words format flag)
Bit 77~73: 00011 ~01111 (5 bits) (reserved)
Bit 72~0: to be defined

User-defined status word (16 modes)
Bit 79: 0 (0: not coincidence event data format)
Bit 78: 0 (0: status words format flag)
Bit 77~73: 10000 ~11111 (4 bits) (user defined)
Bit 72~0: user defined

System software framework models

- ***Host computer software functions***

- system configuration, calibration and monitoring,
- data acquisition
- data analysis

Host computer software functions

- 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

Host computer software functions

- Coincidence events data analysis

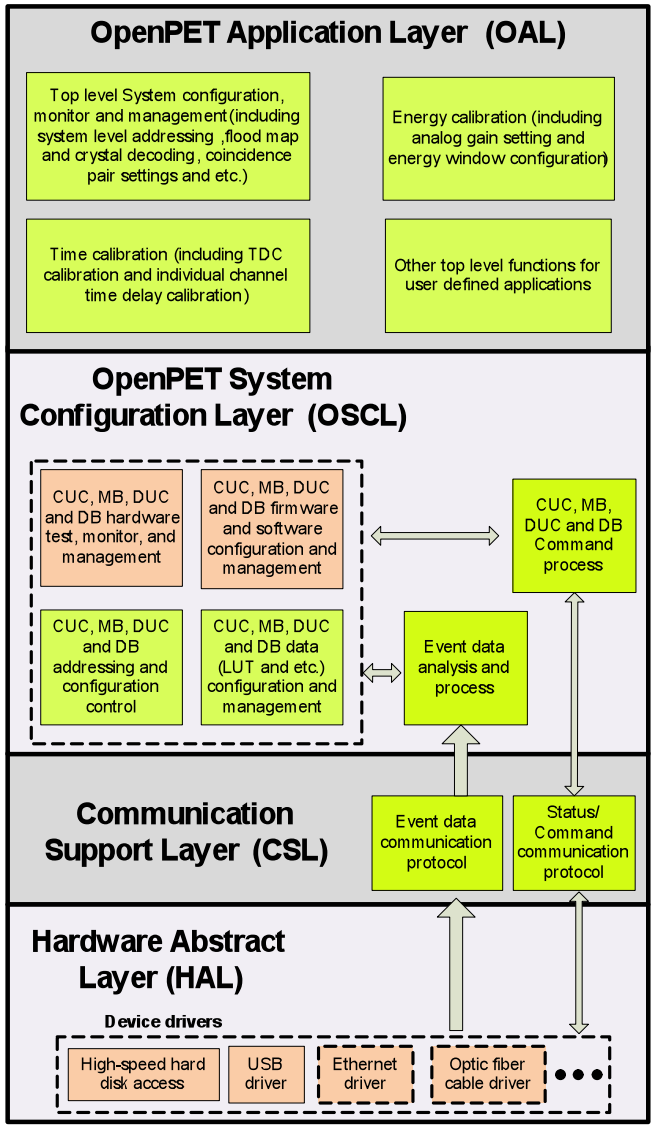
  a. Coincidence pair addressing analysis
  b. Coincidence event analysis
     - Sinogram
     - Random correction and etc.

Host computer software functions

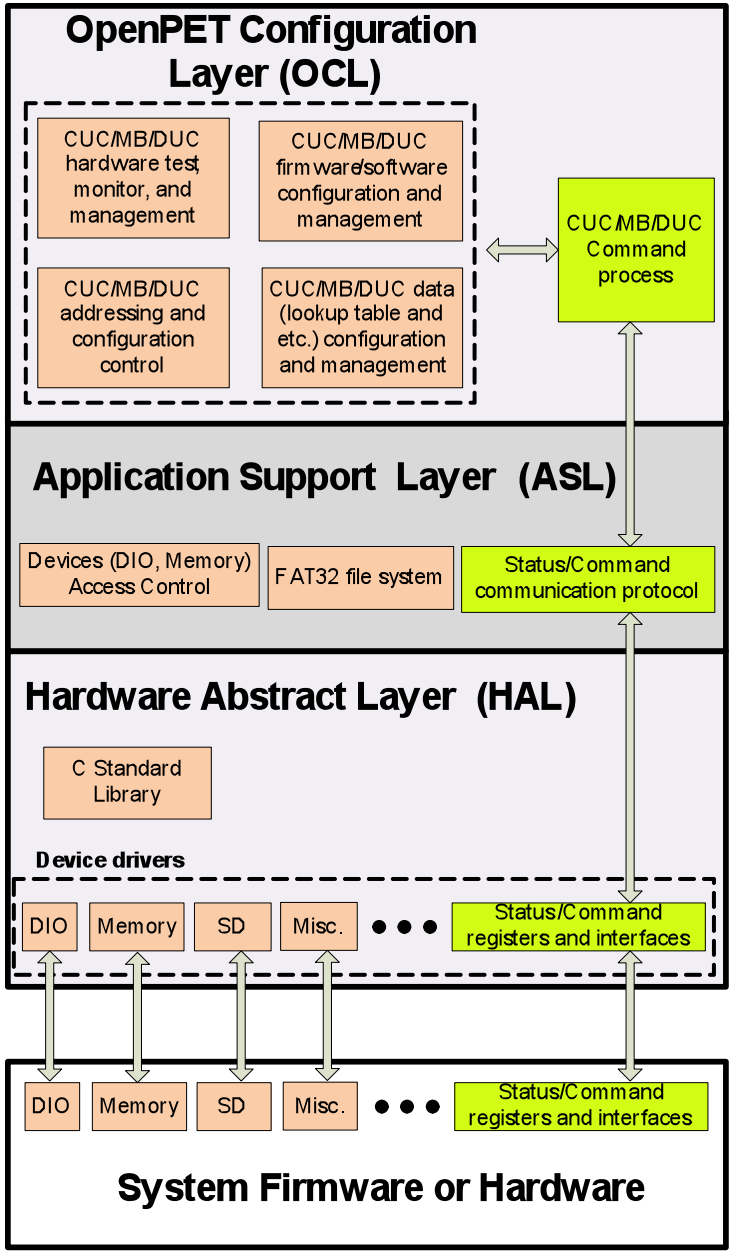
- Status words data analysis

a. Time word analysis
b. Event rate analysis
c. Temperature and voltage monitoring
d. User-defined status processing

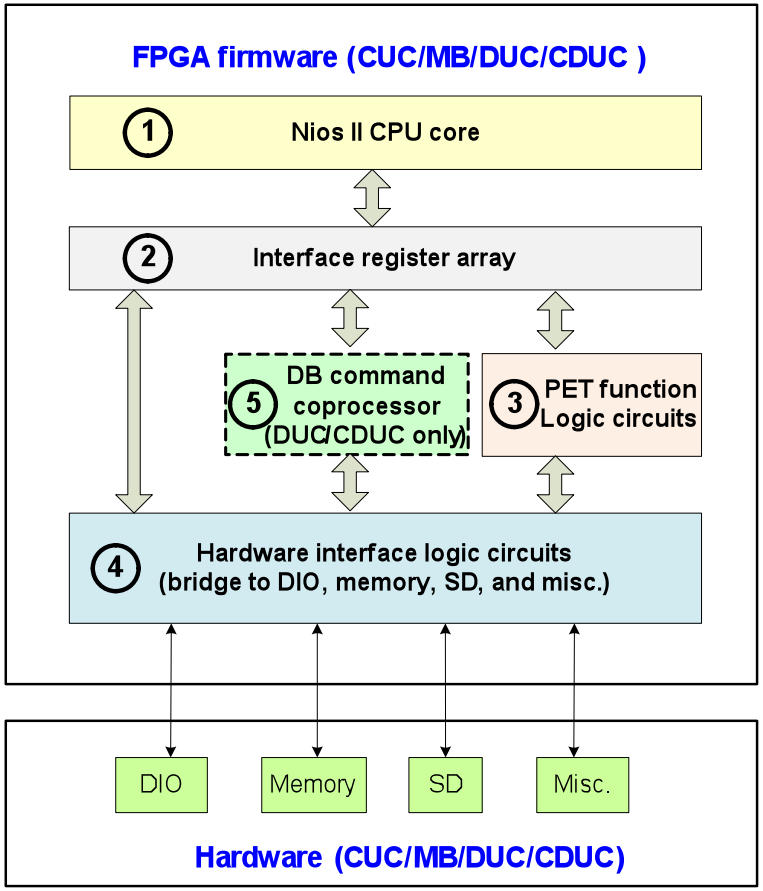
Host computer software model



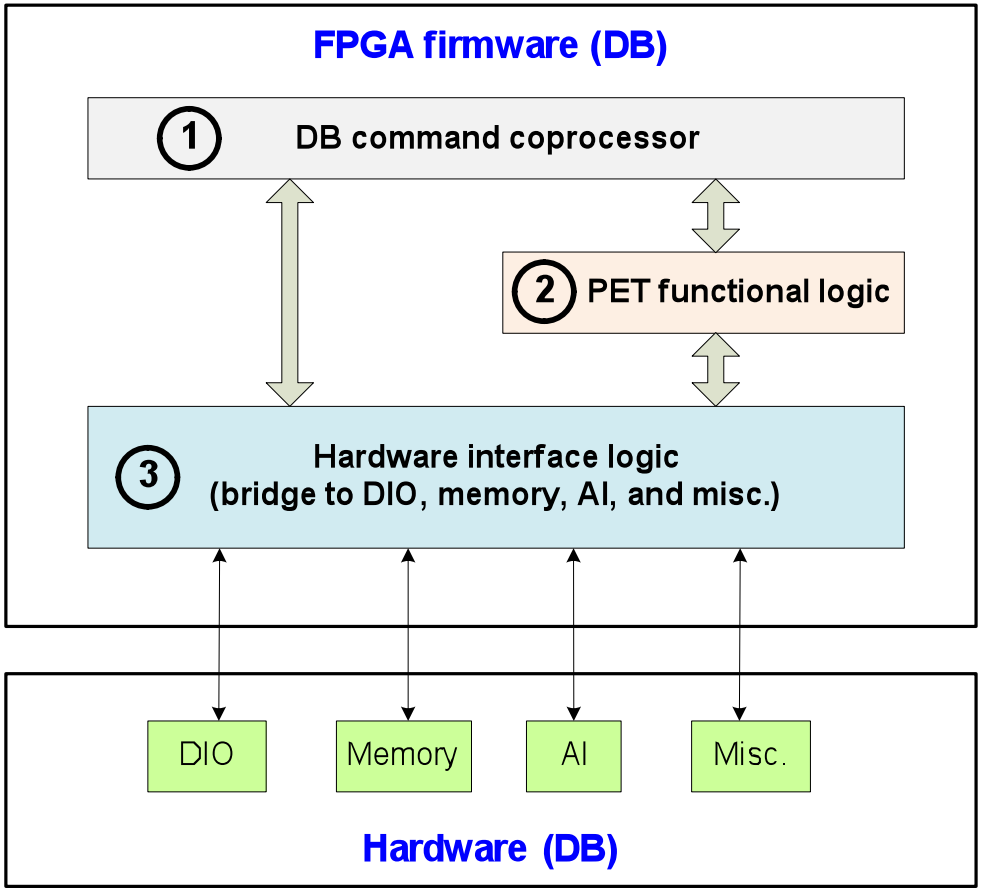
CUC/MB/DUC/CDUC software

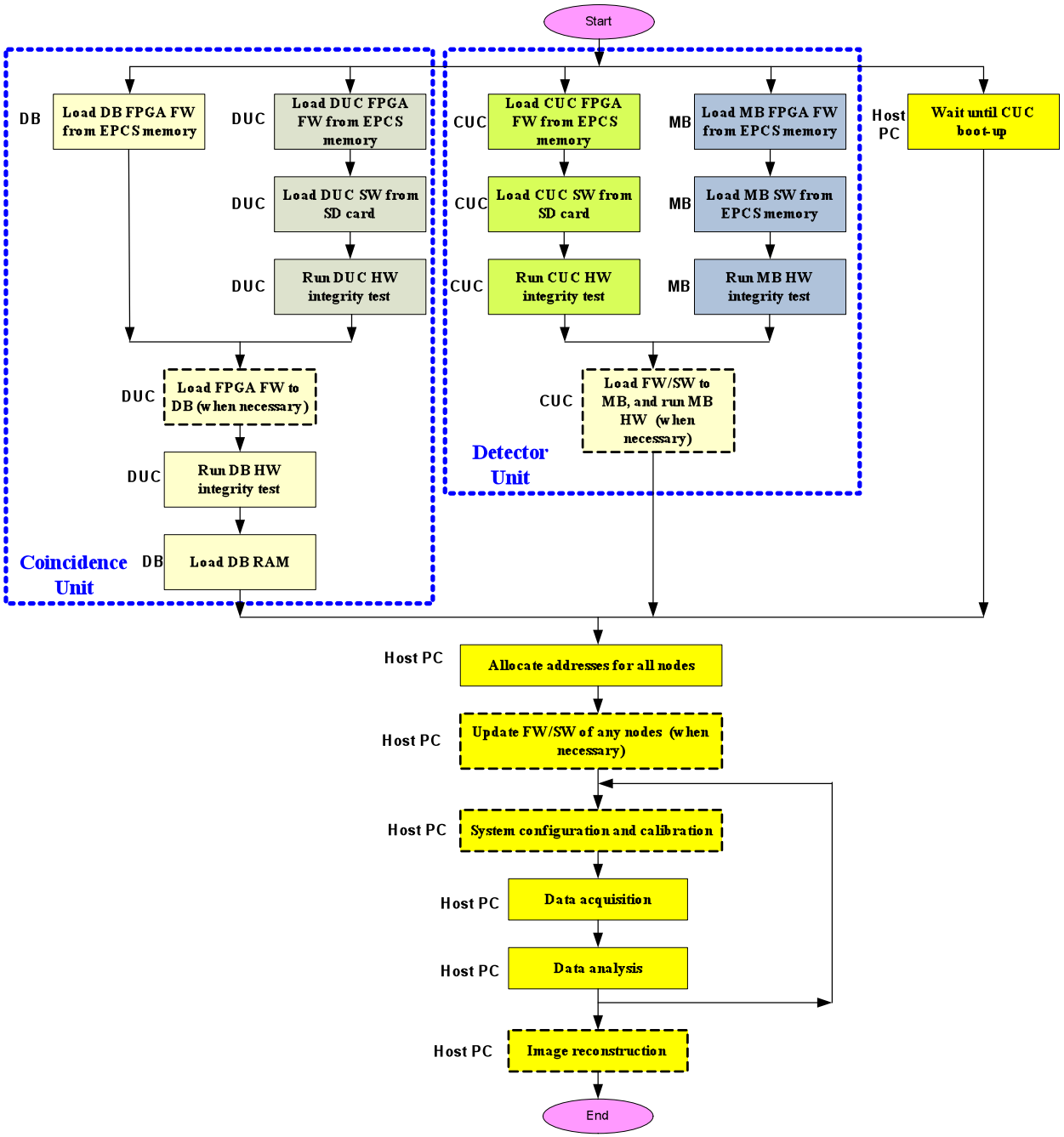


CUC/MB/DUC/CDUC firmware (FPGA) framework model



DB (FPGA) framework model




Start
DB
Load DB FPGA FW from EPCS memory
DUC
Load DUC FPGA FW from EPCS memory
DUC
Load DUC SW from SD card
DUC
Run DUC HW integrity test
DUC
Load FPGA FW to DB (when necessary)
DUC
Run DB HW integrity test
DB
Load DB RAM
Coincidence Unit
CUC
Load CUC FPGA FW from EPCS memory
CUC
Load CUC SW from SD card
CUC
Run CUC HW integrity test
MB
Load MB FPGA FW from EPCS memory
MB
Load MB SW from EPCS memory
MB
Run MB HW integrity test
CUC
Load FW/SW to MB, and run MB HW (when necessary)
Detector Unit
Host PC
Wait until CUC boot-up
Host PC
Allocate addresses for all nodes
Host PC
Update FW/SW of any nodes (when necessary)
Host PC
System configuration and calibration
Host PC
Data acquisition
Host PC
Data analysis
Host PC
Image reconstruction
End

OpenPET Software and Firmware Management

Martin S. Judenhofer, PhD

Department of Biomedical Engineering, University of California-Davis, Davis, CA

OpenPET Meeting, Berkeley, May 10-11, 2012

UCDAVIS
UNIVERSITY OF CALIFORNIA

cmgi

How to manage the software/firmware and associated source codes

Categories of OpenPET users?

- **End user with limited programming skills**
- **End user with programing skills (FPGA & software)**
- **Developers**

How to manage the software/firmware and associated source codes

What would users like to see ?

- **Provide packages for users to download**
- **Provide regular updates**
- **Have ticketing system for bug reporting**
- **Provide documentation for framework usage**
- **Provide binaries for advanced users**

Structure of Software and Firmware

System component	Firmware (FPGA)	uC source (NIOS II)	API (on host)
DB	Yes	No	No
DUC	Yes	Yes	No
MB	Yes	Yes	No
CUC	Yes	Yes	No
Host PC	No	No	Yes

What should/would users like to be able to modify

System component	Simple End user	Software & FPGA End user	Developer
DB FPGA	No	Yes	Yes
DUC FPGA	No	No[1]	Yes
DUC software	No	Yes	Yes
MB FPGA	No	No[1]	Yes
MB Software	No	No[1]	Yes
CUC FGPA	No	No[1]	Yes
CUC software	No	No[1]	Yes
Host PC API use	No	Yes	Yes
Host PC API modification	No	No[1]	Yes

[1]Making changes here requires great attention and may require recompilation and correction of other backbone FPGA code

Deployment of packages (simple user)

In the beginning

- **Simply download binaries**
- **User has to take care that hex files get downloaded properly**
- **Prepare "Boot-SD card"**

Future

- **Download one software package**
- **Main software should be able to configure and flash complete system by downloading the correct binaries (included in software deployment)**
- **Only have to flash the MB FPGA once**
- **Automatically prepare "Boot-SD card"**

Deployment of packages (advance user)

Download binaries source code as needed

- **A software user is not interested in modifying the FPGA code**
- **A non developer user may not need/want to modify the backbone/ framework**
- **Developer should receive all sources**

How the keep a current version of all the source code?

- Since several groups will participate in generating the source codes which will be used in the OpenPET framework, we will need some means of version control

What is version control

- Version control allows tracking of changes done to source codes
- Version control allows seamless switching between versions of source codes (“time machine”)
- Source codes can be branched (parallel version) and later be merged back to the main trunk
- Usually, changes made between versions can be conveniently visualized
- Used for large community projects (e.g. Linux kernel) and in software companies
- Allows web hosting of source code. → easy accessible

What can be used for version control?

- There is several tools available (many free)
  - Subversion
  - GIT
  - Bittracker
  - …
- Web hosting is offered as well

What should we use?

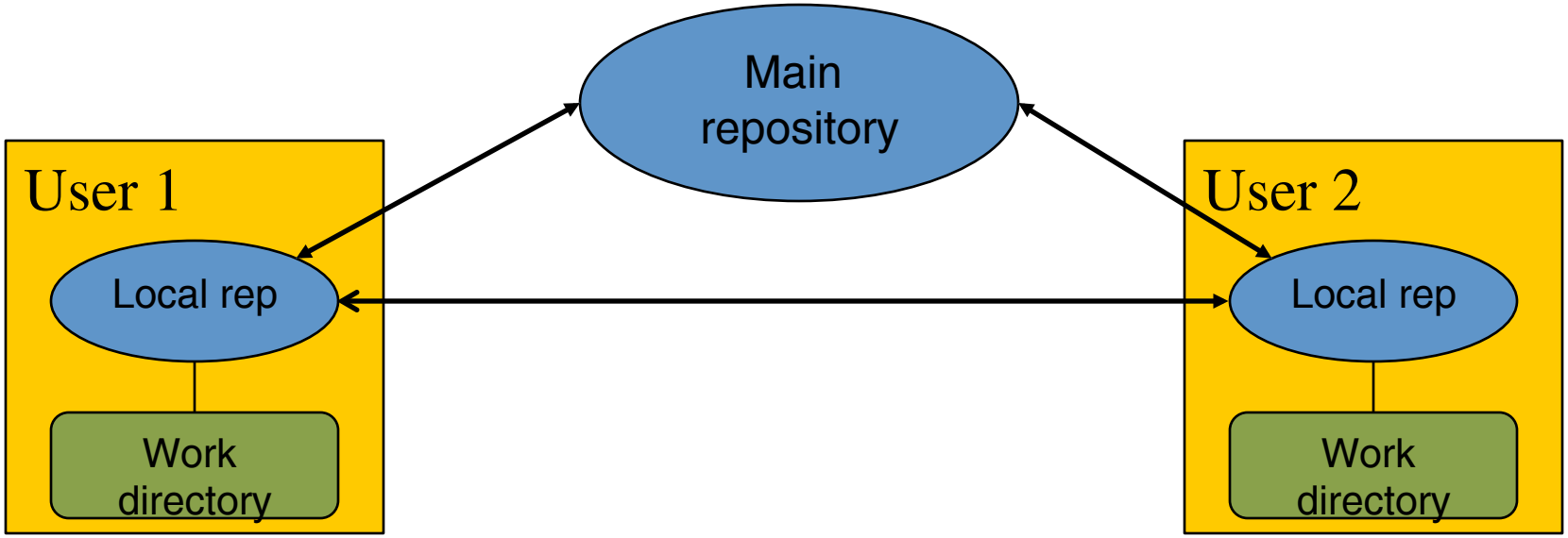
GIT

- Free software
- Easy to use on Windows (great GUI, TortoiseGIT)
- Easy to install on web servers (no external hosting required)
- Uses SSL encoding for transfers
- Can be made available to download from web pages
- Can monitor user access by means of SSL keys

Why GIT?

GIT provides some nice features

- Uses local repositories
  - Each user has to whole history on his desktop
  - User can work independently
  - Sub groups can work independently



Why GIT?

GIT provides some nice features

- Is very fast (quick transfers)
- Provides features to branch and merge
- Great data security, each user has a backup!
- Each commit will have to have some comment which will be very helpful for documentation

Documentation of Source code

- Documentation of source code is essential to make long term use of it and to have others use it
- Documentation essentially requires discipline of the programmer
- Using the versioning software can provide some documentation on specific bug fixes and progress

Documentation of source code

- Use Doxygen

It can help you in three ways:

- It can generate an on-line documentation browser (in HTML) and/or an off-line reference manual (in) from of a set of documented source files. There is also support for generating output in RTF (MS-Word), PostScript, hyperlinked PDF, compressed HTML, and Unix man pages. The documentation is extracted directly from the sources, which makes it much easier to keep the documentation consistent with the source code.
- You can configure doxygen to extract the code structure from undocumented source files. This is very useful to quickly find your way in large source distributions. You can also visualize the relations between the various elements by means of include dependency graphs, inheritance diagrams, and collaboration diagrams, which are all generated automatically.
- You can also use doxygen for creating normal documentation

From : www.doxygen.org

Documentation of source code

- To be most efficient, Doxygen uses tags which are inserted in the source code comments

```
/** <A short one line description>
<Longer description>
<May span multiple lines or paragraphs as needed>

@param Description of method's or function's input parameter
@param ...
@return Description of the return value
*/
```