

UM1935 CAENDigitizer User & Reference Manual

Rev. 5 - 08 May 2013

Purpose of this User Manual

This User Manual contains the full description of the C version of CAENDigitizer library, updated to the last rel. 2.3.1.

Change Document Record

Date	Revision	Changes
16 February 2012	01	Fully revised and implemented §5
18 June 2012	02	Fully revised to document the software library 2.2.1
08 October 2012	03	Removed LabVIEW content
10 December 2012	04	Revised functions at pp. 58 - 59
08 May 2013	05	Revised DPP-CI and DPP-PSD digital probes

Symbols, abbreviated terms and notation

ADC	Analog to Digital Converter
DPP	Digital Pulse Processing
FFT	Fast Fourier Transform
FSR	Full Scale Range
OS	Operating System
SBC	Single Board Computer

Reference Document

- [RD1] UM1934 - CAENComm User & Reference Manual
- [RD2] AN2472 - CONET1 to CONET2 migration
- [RD3] UM2784 - CAENDigitizer LabView User & Reference Manual

CAEN S.p.A.
Via Vetraia, 11 55049 Viareggio (LU) - ITALY
Tel. +39.0584.388.398 Fax +39.0584.388.959
info@caen.it
www.caen.it

© CAEN SpA – 2012

Disclaimer

No part of this manual may be reproduced in any form or by any means, electronic, mechanical, recording, or otherwise, without the prior written permission of CAEN SpA.

The information contained herein has been carefully checked and is believed to be accurate; however, no responsibility is assumed for inaccuracies. CAEN SpA reserves the right to modify its products specifications without giving any notice; for up to date information please visit www.caen.it.

Index

Purpose of this User Manual	2
Change Document Record	2
Symbols, abbreviated terms and notation	2
Reference Document.....	2
Index	3
List of Figures.....	4
List of Tables	4
1 Introduction.....	6
System requirements & installation setup	7
Return Codes.....	8
2 Communication	9
OpenDigitizer.....	9
CloseDigitizer	10
WriteRegister.....	10
ReadRegister	10
Reset.....	11
GetInfo	12
Interrupt configuration.....	14
Set / GetInterruptConfig.....	15
IRQWait.....	16
VMEIRQWait	16
VMEIRQCheck.....	17
IACKCycle	17
VMEIACKCycle.....	18
RearmInterrupt.....	18
Data Readout	19
ClearData	19
DisableEventAlignedReadout	19
Set / GetMaxNumEventsBLT	20
MallocReadoutBuffer.....	20
FreeReadoutBuffer.....	21
ReadData	21
GetNumEvents	22
GetEventInfo.....	22
DecodeEvent	23
AllocateEvent	23
FreeEvent	24
LoadDRS4CorrectionData.....	24
Enable/Disable DRS4Correction.....	25
3 Trigger configuration.....	26
SendSWtrigger.....	26
Set / GetSWTriggerMode.....	27
Set / GetExtTriggerInputMode.....	27
Set / GetChannelSelfTrigger	28
Set / GetGroupSelfTrigger.....	29
Set / GetChannelGroupMask.....	30
Set / GetChannelTriggerThreshold	30
Set / GetGroupTriggerThreshold.....	31
Set / GetChannelPulsePolarity.....	31
Set / GetRunSynchronizationMode.....	32
Set / GetIOLevel	32
Set / GetTriggerPolarity	33
Set / GetGroupFastTriggerThreshold.....	33
Set / GetGroupFastTriggerDCOffset.....	34
Set / GetFastTriggerDigitizing.....	34
Set / GetFastTriggerMode	35
Set / GetDRS4SamplingFrequency.....	35
Set / GetOutputSignalMode	36

4	Acquisition	37
	Set / GetChannelEnableMask.....	37
	Set / GetGroupEnableMask.....	37
	SWStartAcquisition.....	38
	SWStopAcquisition.....	38
	Set / GetRecordLength.....	38
	Set / GetPostTriggerSize.....	39
	Set / GetAcquisitionMode.....	39
	Set / GetChannelDCOffset.....	40
	Set / GetGroupDCOffset.....	40
	Set / GetDESMODE.....	41
	Set / GetZeroSuppressionMode.....	42
	Set / GetChannelZSParams.....	43
	Set / GetAnalogMonOutput.....	44
	Set / GetAnalogInspectionMonParams.....	45
	Set / GetEventPackaging.....	46
	Acquisition example.....	47
	x742 Offline data correction functions.....	50
	LoadCorrectionTables.....	50
	ApplyDataCorrection.....	51
	GetNumEvents.....	51
	GetEventPtr.....	52
	X742_DecodeEvent.....	52
5	DPP specific functions	53
	Set / GetDPPPreTriggerSize.....	53
	GetDPPEvents.....	53
	MallocDPPEvents.....	54
	FreeDPPEvents.....	55
	MallocDPPWaveforms.....	56
	FreeDPPWaveforms.....	57
	DecodeDPPWaveforms.....	57
	SetDPPEventAggregation.....	58
	Set / GetNumEventsPerAggregate.....	58
	Set / GetMaxNumAggregatesBLT.....	59
	SetDPPParameters.....	60
	Set / GetDPPAcquisitionMode.....	62
	Set / GetDPPTriggerMode.....	63
	Set / GetDPP_PHA_VirtualProbe.....	64
	Set / GetDPP_PSD_VirtualProbe.....	65
	Set / GetDPP_CI_VirtualProbe.....	67
	DPP code examples.....	68
6	ZLE x751 specific functions	92
7	Examples of communication settings	93
	Example No.1.....	93
	Example No.2.....	94
	Example No.3.....	95
	Example No.4.....	97
	Example No.5.....	99

List of Figures

Fig. 1.1: Hardware and Software layers.....	6
Fig. 7.1: Connection example no.1.....	93
Fig. 7.2: Connection example no.2.....	94
Fig. 7.3: Connection example no.3.....	95
Fig. 7.4: A2818 network scheme.....	96
Fig. 7.5: Connection example no.4.....	97
Fig. 7.6: Connection example no.5.....	99

List of Tables

Tab. 1: Host PC requirements.....	7
-----------------------------------	---

Tab. 2: Return codes table8

1 Introduction

CAEN has developed a family of Sampling ADCs modules with different form factors (VME, NIM and Desktop). They all provide the possibility to be handled and readout by a host PC via different communication channels.

The CAENDigitizer is a library of functions designed specifically for the digitizer family and it supports also the boards running special DPP (Digital Pulse Processing) firmware. The purpose of this library is to allow the user to open the digitizer, program it and manage the data acquisition in an easy way: with few lines of code the user can make a simple readout program without the necessity to know the details of the registers and the event data format.

The CAENDigitizer library implements a common interface to the higher software layers, masking the details of the physical channel and its protocol, thus making the libraries and applications that rely on the CAENDigitizer independent from the physical layer.

Supported platforms are Windows and Linux 32 and 64 bit. A specific version of CAENDigitizer library has been developed for LabVIEW and is documented in the soon to be released **[RD3]**.

The CAENDigitizer library is based on the CAENComm library that manages the communication at low level (read and write access). The CAENComm requires the CAENVMElib library (access to the VME bus) even in the cases where the VME is not used. For this reason, the CAENVMElib and CAENComm libraries must be already installed on the host PC before installing the CAENDigitizer; however, both CAENVMElib and CAENComm libraries are completely transparent to the user.

Concerning the access through the VME bus, the CAENComm and the CAENDigitizer libraries have been designed to work with CAEN's VME bridges V1718 and V2718. It is possible to make the CAENDigitizer compatible for different types of VME controllers (such as a SBC); for this purpose, the user must provide a library that exports the functions used by the CAENComm. Refer to **[RD1]** for documentation.

Currently, the CAENComm (and so the CAENDigitizer) supports the following communication channels:

- PC → USB → Digitizer (either Desktop or NIM models)
- PC → USB → V1718 → VME → Digitizers (VME models only)
- PC → PCI (A2818) → CONET → Digitizers (all models)
- PC → PCI (A2818) → CONET → V2718 → VME → Digitizers (VME models only)
- PC → PCIe (A3818) → CONET → Digitizers (all models)
- PC → PCIe (A3818) → CONET → V2718 → VME → Digitizers (VME models only)

CONET (Chainable Optical NETWORK) indicates the CAEN proprietary protocol for communication on Optical Link. Refer to **[RD2]** for useful information.

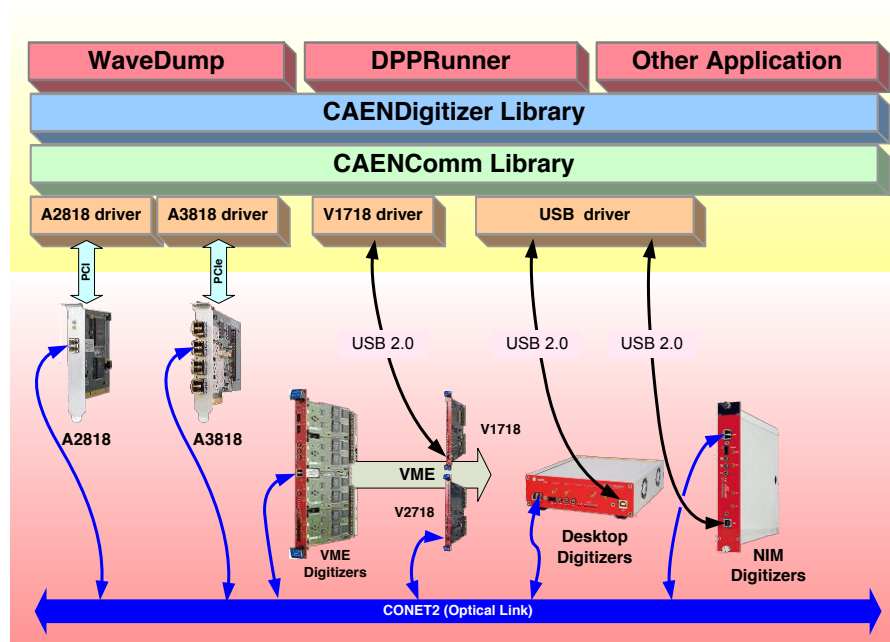




Fig. 1.1: Hardware and Software layers.

System requirements & installation setup

OS	OS version	CAEN Library required	Third-party software required
	2000/XP/Vista/7	CAENComm CAENVMELib	n/a
	kernel Rel. 2.4/2.6 with gnu C/C++ compiler		n/a

Tab. 1: Host PC requirements

Follow the next steps for the installation of CAENDigitizer library:

- Go to CAEN web site in the “Download” area of *CAENDigitizer* page.
- Download the **CAENDigitizer installation package** related to your OS and unpack it.
- Click on the red link under the library links and download the **CAEN required libraries**.

Software Libraries					
	CAENDigitizer library	2.2.1	June, 2012	XP/Vista/7 (32 - 64 bit)	C, LABVIEW
	Library 10.69 MB - Type: .zip				 
	Release Notes 5.33 KB - Type: .txt				 
	[-] CAENDigitizer library requires additional Libraries				
	CAENComm Library			Library 8.67 MB - Type: .zip	 
	CAENVMELib (*)			Library 13.54 MB - Type: .zip	 
	CAENDigitizer library	2.2.1	June, 2012	Kernel 2.4, Kernel 2.6	C, LABVIEW
	Library 158.59 KB - Type: .tgz				 
	Release Notes 5.23 KB - Type: .txt				 
	[-] CAENDigitizer library requires additional Libraries				
	CAENComm library			Library 21.64 KB - Type: .tgz	 
	CAENVMELib & demo			Library & Demo 412.18 KB - Type: .tgz	 

- Install the required libraries starting from CAENVMELib.
- For Windows users: run the *CAENDigitizer* setup executable file and follow the installer instructions.
- For Linux users: follow the instructions in the README file within the library package.



Note: Installation of *CAENDigitizer* also includes a folder “Samples” with a set of source files and projects for readout with standard firmware (p. 47) and DPP firmware (p. 68) to be available for user practice, as well as functions for the offline data correction (p.50) of x742 digitizers.

Return Codes

Error code	Value	Meaning
CAEN_DGTZ_Success	0	Operation completed successfully
CAEN_DGTZ_CommError	-1	Communication error
CAEN_DGTZ_GenericError	-2	Unspecified error
CAEN_DGTZ_InvalidParam	-3	Invalid parameter
CAEN_DGTZ_InvalidLinkType	-4	Invalid Link Type
CAEN_DGTZ_InvalidHandler	-5	Invalid device handler
CAEN_DGTZ_MaxDevicesError	-6	Maximum number of devices exceeded
CAEN_DGTZ_BadBoardType	-7	Operation not allowed on this type of board
CAEN_DGTZ_BadInterruptLev	-8	The interrupt level is not allowed
CAEN_DGTZ_BadEventNumber	-9	The event number is bad
CAEN_DGTZ_ReadDeviceRegisterFail	-10	Unable to read the registry
CAEN_DGTZ_WriteDeviceRegisterFail	-11	Unable to write into the registry
CAEN_DGTZ_InvalidChannelNumber	-13	The Channel is busy
CAEN_DGTZ_ChannelBusy	-14	The channel number is invalid
CAEN_DGTZ_FPIOModelInvalid	-15	Invalid FPIO Mode
CAEN_DGTZ_WrongAcqMode	-16	Wrong acquisition mode
CAEN_DGTZ_FunctionNotAllowed	-17	This function is not allowed for this module
CAEN_DGTZ_Timeout	-18	Communication Timeout
CAEN_DGTZ_InvalidBuffer	-19	The buffer is invalid
CAEN_DGTZ_EventNotFound	-20	The event is not found
CAEN_DGTZ_InvalidEvent	-21	The event is invalid
CAEN_DGTZ_OutOfMemory	-22	Out of memory
CAEN_DGTZ_CalibrationError	-23	Unable to calibrate the board
CAEN_DGTZ_DigitizerNotFound	-24	Unable to open the digitizer
CAEN_DGTZ_DigitizerAlreadyOpen	-25	The Digitizer is already open
CAEN_DGTZ_DigitizerNotReady	-26	The Digitizer is not ready to operate
CAEN_DGTZ_InterruptNotConfigured	-27	The Digitizer has not the IRQ configured
CAEN_DGTZ_DigitizerMemoryCorrupted	-28	The digitizer flash memory is corrupted
CAEN_DGTZ_DPPFirmwareNotSupported	-29	The digitizer DPP firmware is not supported in this lib version
CAEN_DGTZ_NotYetImplemented	-99	The function is not yet implemented

Tab. 2: Return codes table

2 Communication

These functions allow to open and close the connection with the digitizer as well as get board information such as the serial number, the model, the firmware revision, etc. To open one board is necessary to describe the physical communication channel from the PC to the device (as already indicated in the introduction). Once the device is opened, the function returns a handle that becomes the unique identifier of that device; any access operation to the device (except for VME IRQ management) will take place according to its handle, thus making transparent the physical channel.

OpenDigitizer

Desktop and NIM versions can be directly handled via USB, just connecting the digitizer to the host PC via the USB cable (the USB driver is available on Digitizer web page).

Description

Opens the digitizer and gets the device handle. See the examples below for the different types of communication channels and the relevant parameters.

Synopsis

```
CAEN DGTZ ErrorCode CAENDGTZ API
CAEN DGTZ OpenDigitizer (CAEN DGTZ ConnectionType LinkType,
                        int LinkNum,
                        int ConetNode,
                        uint32_t VMEBaseAddress,
                        int *handle
                        );

typedef enum {
    CAEN_DGTZ_USB = 0, // USB (direct or V1718->VME)
    CAEN_DGTZ_PCI_OpticalLink = 1, // PCI-A2818 (direct or V2718->VME)
    CAEN_DGTZ_PCIE_OpticalLink = 2, // PCIe-A3818 (direct or V2718->VME)
    CAEN_DGTZ_PCIE_EmbeddedDigitizer = 3, // future development
} CAEN_DGTZ_ConnectionType;
```

Arguments

Name	Description
LinkType	Indicates the physical communication channel. It can be CAEN_DGTZ_USB (either direct connection or VME through V1718), CAEN_DGTZ_PCI_OpticalLink (A2818 -> Optical Link, either direct connection or VME through V2718) or CAEN_DGTZ_PCIE_OpticalLink (same as A2818 but using A3818).
LinkNum	Link number: in case of USB, the link numbers are assigned by the PC when you connect the cable to the device; it is 0 for the first device, 1 for the second and so on. There is not a fixed correspondence between the USB port and the link number. For the CONET, the link number indicates which link of A2818 or A3818 is used; Link index start from 0 (1 st Optical link port in the 1 st slot used). It is not known a priori which is the first slot used (it depends on the motherboard of the PC used.). IMPORTANT NOTE: if A2818 and A3818 are installed together, the A2818 have lower index assigned.
ConetNode	The CONET node identifies which device in the Daisy chain is being addressed. The node is 0 for the first device in the chain, 1 for the second and so on. In case of USB, <i>ConetNode</i> must be 0.
VMEBaseAddress	VME Base Address of the board (rotary switches setting) expressed as a 32-bit number. This argument is used only for the VME models accessed through the VME bus and MUST BE 0 in all other cases.
*handle	Pointer to the handler returned by the open function

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Examples

See the various examples described at p. 92 - 99.

CloseDigitizer

Description

This function closes the digitizer.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_CloseDigitizer (int handle);
```

Arguments

Name	Description
handle	Device handler

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

WriteRegister

Description

Generic write access to one register of the digitizer. The CAENDigitizer library provides specific functions for most of the parameters settings; in the case where there is not a specific function for accessing a particular register or the user wants to force the writing of a datum, this function makes it possible to perform a direct access to the registers. It is worth noticing that the overwriting of some settings can cause inconsistency of the operations.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_WriteRegister(int handle,
                        uint32_t Address,
                        uint32_t Data
                        );
```

Arguments

Name	Description
handle	Device handler
Address	Register address. For the VME access, this is the lower 16 bit part of the VME address bus
Data	32 bit data to write

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

ReadRegister

Description

Generic read access to one register of the digitizer (see **WriteRegister** for more details).

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_ReadRegister(int handle,
                       uint32_t Address,
                       uint32_t *Data
                       );
```

Arguments

Name	Description
handle	Device handler
Address	Register address. For the VME access, this is the lower 16 bit part of the VME address bus
*Data	Data read from the board (32 bit)

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Reset

Description

This function resets the Digitizer. All internal registers and states are restored to default.

Synopsis

```
CAEN DGTZ ErrorCode CAENDGTZ API  
CAEN_DGTZ_Reset (int handle);
```

Arguments

Name	Description
<code>handle</code>	Device handler

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

GetInfo

Description

The function reads from the board some information such as serial number, model, number of channels, firmware release and other parameters of the device.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetInfo(int handle,
                  CAEN_DGTZ_BoardInfo_t *BoardInfo
                  );

typedef struct {
char                               ModelName[12];
uint32_t                           Model;
uint32_t                           Channels;
uint32_t                           FormFactor;
uint32_t                           FamilyCode;
char                               ROC_FirmwareRel[20];
char                               AMC_FirmwareRel[20];
uint32_t                           SerialNumber;
uint32_t                           PCB_Revision;
uint32_t                           ADC_NBits;
} CAEN_DGTZ_BoardInfo_t;

typedef enum {
CAEN_DGTZ_V1724                    =0L,
CAEN_DGTZ_V1721                    =1L,
CAEN_DGTZ_V1731                    =2L,
CAEN_DGTZ_V1720                    =3L,
CAEN_DGTZ_V1740                    =4L,
CAEN_DGTZ_V1751                    =5L,
CAEN_DGTZ_DT5724                   =6L,
CAEN_DGTZ_DT5721                   =7L,
CAEN_DGTZ_DT5731                   =8L,
CAEN_DGTZ_DT5720                   =9L,
CAEN_DGTZ_DT5740                   =10L,
CAEN_DGTZ_DT5751                   =11L,
CAEN_DGTZ_N6724                    =12L,
CAEN_DGTZ_N6721                    =13L,
CAEN_DGTZ_N6731                    =14L,
CAEN_DGTZ_N6720                    =15L,
CAEN_DGTZ_N6740                    =16L,
CAEN_DGTZ_N6751                    =17L,
CAEN_DGTZ_DT5742                   =18L,
CAEN_DGTZ_N6742                    =19L,
CAEN_DGTZ_V1742                    =20L,
CAEN_DGTZ_DT5780                   =21L,
CAEN_DGTZ_N6780                    =22L,
CAEN_DGTZ_V1780                    =23L,
CAEN_DGTZ_DT5761                   =24L,
CAEN_DGTZ_N6761                    =25L,
CAEN_DGTZ_V1761                    =26L,
} CAEN_DGTZ_BoardModel_t;

typedef enum {
CAEN_DGTZ_VME64_FORM_FACTOR        = 0L,
CAEN_DGTZ_VME64X_FORM_FACTOR       = 1L,
CAEN_DGTZ_DESKTOP_FORM_FACTOR      = 2L,
CAEN_DGTZ_NIM_FORM_FACTOR          = 3L,
} CAEN_DGTZ_BoardFormFactor_t;

typedef enum {
CAEN_DGTZ_XX724_FAMILY_CODE        = 0L, // 14 bit, 100 MS/s
CAEN_DGTZ_XX721_FAMILY_CODE        = 1L, // 8 bit, 500 MS/s
CAEN_DGTZ_XX731_FAMILY_CODE        = 2L, // 8 bit, 500-1000 MS/s
CAEN_DGTZ_XX720_FAMILY_CODE        = 3L, // 12 bit, 250 MS/s
CAEN_DGTZ_XX740_FAMILY_CODE        = 4L, // 12 bit, 65 MS/s
CAEN_DGTZ_XX751_FAMILY_CODE        = 5L, // 10 bit, 1000-2000 MS/s
CAEN_DGTZ_XX742_FAMILY_CODE        = 6L, // 12 bit, 5 GS/s, switched capacitor
CAEN_DGTZ_XX780_FAMILY_CODE        = 7L, // 14 bit, 100 MS/s Multi Channel Analyser
CAEN_DGTZ_XX761_FAMILY_CODE        = 8L, // 10 bit, 4 GS/s
} CAEN_DGTZ_BoardFamilyCode_t;
```

Arguments

Name	Description
handle	Device handler
*Board Info	Pointer to the structure containing the Board Info filled by the CAEN_DGTZ_GetInfo

BoardInfo Fields

Name	Description
ModelName	Model name: for example "V1724"
Model	See type enum CAEN_DGTZ_BoardModel_t
Channels	Number of channels
FormFactor	Format Factor (VME, NIM, Desktop); see type CAEN_DGTZ_BoardFormFactor_t
FamilyCode	Family (ADC type); see type CAEN_DGTZ_FamilyCode_t
ROC_FirmwareRel	Firmware Revision of the FPGA on the mother board (ROC); for example "01.02"
AMC_FirmwareRel	Firmware Revision of the FPGA on the daughter board (AMC)
SerialNumber	Serial number of the board
PCB_Revision	PCB Revision number
ADC_NBits	Number of bits of the ADC

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Interrupt configuration

All digitizers can generate interrupt requests (IRQ) to the PC to the occurrence of a particular condition: if the memory contains at least N_e events ready for reading, where N_e is a programmable parameter.

This allows to create programs that build the process of readout (read access to the memory buffer) on interrupts: they perform passive wait cycles, until they are awakened by the driver at the arrival of an interrupt from the digitizer; at such point, the process can read data, aware to find at least N_e events in memory, without having to check in advance the presence of data, as in the case of the readout based on polling.

The readout based on the interrupts is therefore more efficient, in terms of employment of the PC resources, compared to the one based on polling.

The interrupt requests are transferred from the digitizer to the PC via the optical link, in one of the following ways:

- Direct connection to the optical link (all models): the digitizer sends the interrupt request on the optical link to the A2818 PCI or A3818 PCIe connected to the PC, and these, in their turn, assert the interrupt request on the PCI bus or PCIe respectively. In this case, the interrupt request coming to the PC is uniquely associated with the digitizer which sent it.
- Connection via VME bus: in this case, the digitizer asserts the interrupt request on the VME bus on one of the 7 IRQ lines, and this request is detected by the VME master (V2718), which sends it via optical link to the PC, in the same manner described above. In this case, since the lines IRQ [7 .. 1] of the VME are shared with all modules on VME bus, it is necessary to identify the module that sent the request, as explained farther.



Note: interrupts cannot be used in case of communication via USB (either directly or through V1718 and VME)

Set / GetInterruptConfig

Description

Enable / Disable the digitizer to generate an interrupt request when the memory contains at least N_e events ready for reading, where N_e is the parameter `event_number`.

- In the case of VME models, the IRQ level to be activated on VME bus can be set from 1 to 7;
- in the case of the optical link, level should be 1.

The `status_id`, according to the specifications of the VME bus, is the value returned by the card during the interrupt acknowledge cycle and allows the operator to see which digitizer has asserted the interrupt request on the VME bus; in the programming stage, the user must set different `status_id` values for each digitizer. In the case of the optical link, the `status_id` is meaningless.

The mode parameter sets the interrupt release policy of the digitizer: in particular, **Roak** (Release On Acknowledge) mode foresees that the request is issued immediately after the interrupt acknowledge cycle (IACK), while in the case of **Rora** (Release on Register Access) mode, the interrupt request is not released by the digitizer until the user accesses a particular registry to disable it; in the case of the digitizer, the release occurs by setting to zero the level in the VME Control register, by calling the "Set" function of **Set / GetInterruptConfig** with `status = disabled`.

The methods Rora and Roak, arising from the VME specifications, are implemented also in the CONET protocol of the optical link, with the exception that the Interrupt Acknowledge cycle with CONET is required only to release the interrupt and not to identify the device that has generated it, since this information is already determined from the handle

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetInterruptConfig (int handle,
                             CAEN_DGTZ_EnaDis_t state,
                             uint8_t level,
                             uint32_t status_id,
                             uint16_t event_number,
                             CAEN_DGTZ_IRQMode_t mode
                             );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetInterruptConfig (int handle,
                              CAEN_DGTZ_EnaDis_t *state,
                              uint8_t *level,
                              uint32_t *status_id,
                              uint16_t *event_number,
                              CAEN_DGTZ_IRQMode_t *mode
                              );

typedef enum {
    CAEN_DGTZ_ENABLE = 1L,
    CAEN_DGTZ_DISABLE = 0L,
} CAEN_DGTZ_EnaDis_t;

typedef enum {
    CAEN_DGTZ_IRQ_MODE_RORA = 0,
    CAEN_DGTZ_IRQ_MODE_ROAK = 1,
} CAEN_DGTZ_IRQMode_t;
```

Arguments

Name	Description
<code>handle</code>	Device handler
<code>state</code>	Enable/Disable
<code>level</code>	VME IRQ Level (from 1 to 7). Must be 1 for direct connection through CONET
<code>status_id</code>	32 bit number assigned to the device and returned by the device during the Interrupt Acknowledge
<code>event_number</code>	If the number of events ready for the readout is equal to or greater than <code>event_number</code> , then the digitizer asserts the interrupt request
<code>mode</code>	Interrupt release mode: <code>CAEN_DGTZ_IRQ_MODE_RORA</code> (release on register access) or <code>CAEN_DGTZ_IRQ_MODE_ROAK</code> (release on acknowledge)

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

IRQWait

Description

Once set up the digitizer to generate an interrupt request by the function described above, the reading process can enter a state of passive waiting to be woken up as the interrupt request from the digitizer which is communicating with (the one identified uniquely from the handle passed as a parameter), is sent. This function is valid only for direct connection to link optical digitizer, in the case of communication via the VME, use **VMEIRQWait**. The timeout parameter indicates the maximum waiting time before being forced to wake up even without interrupt. In this case, the value returned by the function is 18.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_IRQWait(int handle,
                  uint32_t timeout
                  );
```

Arguments

Name	Description
handle	Device handler
timeout	Timeout (max wait time) in ms

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

VMEIRQWait

Description

This function, as the one described above, implements the passive waiting from which the waking occurs up in response to an interrupt request from the digitizer. The main difference is that in this case, the digitizer asserts a IRQ (1 to 7) on the VME bus and this is transferred to the PC by the master VME V2718. Since other digitizers could be on the VME bus (and therefore different handles that identify them within the program), and each one can generate interrupts, even on the same IRQ line, the management of interrupts cannot take place through the handle of the digitizer (which cannot be uniquely associated with the request arrived at the PC) but must be performed through the handle of the master VME V2718 which is the unique collector of interrupt requests to the PC. Once awakened from the waiting status, the process of reading can understand what digitizer has actually sent the request via the interrupt acknowledge cycle.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_VMEIRQWait(int LinkType,
                    int LinkNum,
                    int ConetNode,
                    uint32_t timeout,
                    int *VMEHandle
                    );
```

Arguments

Name	Description
LinkType	Indicates the physical communication channel used to connect the CAEN VME bridge that handles the interrupts on the VME bus. It can be <code>CAENComm_USB</code> for the V1718 or <code>CAENComm_PCI_OpticalLink</code> for the A2818 -> Optical Link -> V2718) or <code>CAENComm_PCIE_OpticalLink</code> (same as A2818 but using A3818)
LinkNum	Link number: in case of USB, the link numbers are assigned by the PC when you connect the cable to the device; it is 0 for the first device, 1 for the second and so on. There is not a fixed correspondence between the USB port and the link number. For the CONET, the link number indicates which A2818 or A3818 is used; also in this case, it is not known a priori which PCI/PCle card is assigned to which number.
ConetNode	The CONET node identifies which device in the Daisy chain is being addressed. The node is 0 for the first device in the chain, 1 for the second and so on. In case of USB, ConetNode must be 0.
timeout	Timeout (max wait time) in msec
*VMEHandle	Device handler of the CAEN VME Bridge that received the interrupt request

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

VMEIRQCheck

Description

This function allows to read the status of interrupt requests on the VME bus (IRQ1-7) and, for this reason, the handle to be passed is the VME master one, not the digitizer one. This function can only be used for digitizer that communicate via the VME bus. The purpose of this function is almost exclusively for debugging.

Synopsis

```
CAEN DGTZ ErrorCode CAENDGTZ API
CAEN DGTZ VMEIRQCheck(int VMEHandle,
                      uint8_t *Mask
                      );
```

Arguments

Name	Description
VMEHandle	Device handler of the VME bridge that handles the interrupts
*Mask	Mask of the IRQ lines read from the VME bus (1=IRQ active, 0=IRQ not active)

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

IACKCycle

Description

This function performs an interrupt acknowledge cycle on the digitizer identified by the handle. This function can only be used for direct communications via optical links; in case of communication via the VME, it should be used **VMEIACKCycle** described farther. Although in the case of direct connection to the optical link there is not need to identify the digitizer that generated the interrupt request, the IACK cycle is still executed in the case of mode ROAK (release on acknowledge) to release the request

Synopsis

```
CAEN DGTZ ErrorCode CAENDGTZ API
CAEN DGTZ IACKCycle(int handle,
                    int32_t *board_id
                    );
```

Arguments

Name	Description
handle	Device handler of the digitizer
*board_id	Data (status_id) returned by the digitizer that asserted the interrupt request.

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

VMEIACKCycle

Description

This function performs an interrupt acknowledge cycle to know the `board_id` of the board that raised an interrupt. As described previously, in the case of interrupt requests on the VME bus, it is not possible to know in advance which digitizer asserted a certain IRQ line. Indeed, it could also happen that a line is asserted by any other slave on the VME bus with which no communication is established. For this reason, when the reading process on hold in a specific IRQ is awakened, it must perform an interrupt acknowledge cycle to see which one generated the interrupt. The identification is as follows: during acknowledge cycle (which is very similar to a read cycle), the slave that caused the interruption puts on his bus `status_id`, actually the value previously programmed by the user through the “Set” function of **Set / GetInterruptConfig** function. In the case of multiple cards having different values of the programmed `status_id`, the user will be able to figure out who sent the request, and then which one is to be read. It should be noted that in the case of multiple cards on the bus (even inhomogeneous), the interrupt management must be centralized, as the acknowledge cycle should be performed only once. It is therefore not recommended (although possible) to have more process waiting on the same IRQ line.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_VMEIACKCycle(int VMEHandle,
                        uint8 t level,
                        int32 t *board id
                        );
```

Arguments

Name	Description
<code>VMEHandle</code>	Device handler of the CAEN VME bridge that handles the interrupts
<code>level</code>	IRQ level (from 1 to 7) on which to perform the interrupt acknowledge cycle
<code>*board_id</code>	Data (<code>status_id</code>) returned by the digitizer that asserted the interrupt request

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Examples

WAVEDUMP Code *(To be implemented)*

RearmInterrupt

Description

Rearm the Interrupt.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_RearmInterrupt (int handle);
```

Arguments

Name	Description
<code>handle</code>	Device handler

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Data Readout

The data reading from the memories of the digitizer is done through BlockRead cycles (although it is possible also to run cycles to read each buffer). In the case of direct communication via USB or optical link, the protocol that manages the blocks transfer is CAEN proprietary and therefore there are no ambiguities or special options to be decided. Conversely, if reading takes place through the VME bus, since the standard provides different types of access and not all VME masters support all modes (or do it differently), the reading mode may need to be adapted according to the master features. The library foresees the use of master CAEN V1718 and V2718 and the readout mode is optimized for these modules.

ClearData

Description

This function Clears the data stored in the buffers of the Digitizer.



Note: generally it is not necessary to call this function, because the digitizer runs automatically a clear cycle when an acquisition starts. The function can be used during an acquisition when aware that the data stored in memory are not interesting and not going to be read

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_ClearData(int handle);
```

Arguments

Name	Description
handle	Device handler

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

DisableEventAlignedReadout

Description

By default, in the data transfer from the memory of the digitizer to the PC, regardless of the type of link used, events are aligned: the digitizer stop the transfer after transferring an integer number N_e of events, where N_e is user programmable through the "Set" function of **Set / GetMaxNumEventsBLT**, even if the user has requested the transfer of more data. In the case of communication via USB and optical links, the premature termination of the transfer is foreseen by the protocol; instead, for the VME Block Transfer, the transfer is interrupted by the digitizer asserting the bus error (if enabled, see above).

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_DisableEventAlignedReadout(int handle);
```

Arguments

Name	Description
handle	Device handler

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetMaxNumEventsBLT

Description

Concerning the Digitizers running the standard firmware, this function sets/gets the maximum number of events for each transfer. Regardless of the type of link, during a block transfer cycle, the digitizer stops the transfer after a predetermined number of events (or when the memory is empty). The greater the number of events transferred (and thus the size of the block read), the greater the efficiency of the readout, since the protocol overhead is smaller. In contrast, higher values for **MaxNumEventsBLT** imply the need to allocate a memory buffer for very large the readout.



Note: If using DPP-PHA, DPP-PSD or DPP-CI firmware, you have to refer to the **SetDPPEventAggregation** function.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetMaxNumEventsBLT(int handle,
                             uint32_t numEvents
                             );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetMaxNumEventsBLT(int handle,
                              uint32_t *numEvents
                              );
```

Arguments

Name	Description
handle	Device handler
numEvents	Maximum number of events to transfer in a BlockRead

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

MallocReadoutBuffer

Description

This function allocates the memory buffer for the data block transfer from the digitizer to the PC. The size of the buffer allocated is calculated by the function according to the size of the event, the number of enabled channels and the maximum number of events transferred by each block transfer (see previous function). For this reason, the function must be called after having programmed the digitizer, if the parameters that determine the size of the buffer change, it is necessary to free it by calling the **FreeReadoutBuffer** function and then reallocated.



Note: the buffer pointer must be initialized to NULL.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_MallocReadoutBuffer(int handle,
                              char **buffer
                              uint32_t *size);
```

Arguments

Name	Description
handle	Device handler
**buffer	Pointer to the readout buffer allocated (WARNING: **buffer MUST be initialized to NULL)
*size	The size in byte of the buffer allocated

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

FreeReadoutBuffer

Description

Frees memory allocated by the **MallocReadoutBuffer** function.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_FreeReadoutBuffer(char **buffer);
```

Arguments

Name	Description
**buffer	Pointer to the readout buffer to free, returned by the MallocReadoutBuffer function.

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

ReadData

Description

This function performs a block transfer of data from the digitizer to the computer. The size of the block to be transferred is determined by the function according to parameters set and the mode of readout. The block can contain one or more events. The data is transferred into the buffer memory previously allocated by **MallocReadoutBuffer** function. The function returns in *bufferSize* the size of the data block read from the card, expressed in long words.



Note:

CAEN_DGTZ_SLAVE_TERMINATED_READOUT_MBLT for VME accesses:

In this case the digitizer is programmed to assert the VME Bus Error during a Block Transfer cycle to prematurely end the cycle when it no longer has data to transfer or has completed the transfer of the maximum number of events planned (see *BLT_EVENT_NUM* register, or **Set / GetMaxNumEventsBLT** function). This use of the Bus Error, though not specifically provided by the VME standard for this purpose, it is actually very common. However, some VME masters have a Bus Error management not suitable for this purpose.

CAEN_DGTZ_POLLING_MBLT for VME accesses:

The VME Bus Error generation is disabled, the transfer always continues until the completion of the number of bytes required and, if there are no data to be transferred, the digitizer will insert filler words (0xFFFFFFFF)

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_ReadData(int handle,
                  CAEN_DGTZ_ReadMode_t mode,
                  char *buffer,
                  uint32_t *bufferSize
                  );

typedef enum {
    CAEN_DGTZ_SLAVE_TERMINATED_READOUT_MBLT = 0,
    CAEN_DGTZ_SLAVE_TERMINATED_READOUT_2eVME = 1,
    CAEN_DGTZ_SLAVE_TERMINATED_READOUT_2eSST = 2,
    CAEN_DGTZ_POLLING_MBLT = 3,
    CAEN_DGTZ_POLLING_2eVME = 4,
    CAEN_DGTZ_POLLING_2eSST = 5,
} CAEN_DGTZ_ReadMode_t;
```

Arguments

Name	Description
handle	Device handler
mode	CAEN_DGTZ_SLAVE_TERMINATED_READOUT_MBLT = 0 CAEN_DGTZ_SLAVE_TERMINATED_READOUT_2eVME = 1 CAEN_DGTZ_SLAVE_TERMINATED_READOUT_2eSST = 2 CAEN_DGTZ_POLLING_MBLT = 3 CAEN_DGTZ_POLLING_2eVME = 4 CAEN_DGTZ_POLLING_2eSST = 5
*buffer	Pointer to the readout buffer
*bufferSize	Size of the data block read from the board (expressed in bytes)

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

GetNumEvents

Description

This function scans the readout buffer and gets the number of events contained in the data block previously read by the **ReadData** function. The number of events is returned in the parameter *numEvents*.



Note: If using DPP-PHA, DPP-PSD or DPP-CI firmware, you have to refer to the **GetDPPEvents** function.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetNumEvents(int handle,
                       char *buffer,
                       uint32_t buffsize,
                       uint32_t *numEvents
                       );
```

Arguments

Name	Description
handle	Device handler
*buffer	Pointer to the readout buffer
buffsize	Size of the data block contained in the readout buffer. This value is given by the ReadData function.
*numEvents	Number of events contained in the readout buffer

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

GetEventInfo

Description

This function retrieves the information (trigger time stamp, event number, channel mask, etc.) associated to one event contained in the readout buffer. This function reads the header of the *numEvent* event in the buffer, fills the *eventInfo* structure and set the data pointer *EventPtr* to the first word of the event data in the readout buffer. This pointer will be passed to the **DecodeEvent** function described below.



Note: If using DPP-PHA, DPP-PSD or DPP-CI firmware, you have to refer to the **GetDPPEvents** function.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetEventInfo(int handle,
                       char *buffer,
                       uint32_t buffsize,
                       int32_t numEvent,
                       CAEN_DGTZ_EventInfo_t *eventInfo,
                       char **EventPtr
                       );

typedef struct
{
    uint32_t EventSize;
    uint32_t BoardId;
    uint32_t Pattern;
    uint32_t ChannelMask;
    uint32_t EventCounter;
    uint32_t TriggerTimeTag;
} CAEN_DGTZ_EventInfo_t;
```

Arguments

Name	Description
handle	Device handler
*buffer	Pointer to the readout buffer
buffsize	Size of the data block contained in the readout buffer
numEvent	Number of the requested event in the readout buffer (0 is the first event in the buffer)
*eventInfo	Pointer to the structure that contains the information about the requested event
**EventPtr	Pointer to the requested event data in the readout buffer

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

DecodeEvent

Description

Each type of digitizer has a different event data format. This function decodes (unpacks) the data of a specified event and fills the event structure containing the data of each channel (i.e. the waveform and/or other parameters in case of DPP) separately. There are two ways to allocate the memory for the unpacked event data:

- If the pointer `**Evt` to the event structure passed to the function is initialized to NULL, then the event is automatically allocated by the **DecodeEvent** function that knows the exact size of the decoded event data, hence there is no waste in the memory usage. In this case, the user must free the event memory buffer once it has been used.
- The memory buffer for the decoded event can be allocated once at the beginning of the acquisition; this is done by the **AllocateEvent** function. This solution is more efficient in terms of readout rate (no waste of time to allocate and free the memory) but requires more memory because the buffer must be able to contain the maximum event size. In this mode, the memory free must be done at the end of the acquisition.



Note: If using DPP-PHA, DPP-PSD or DPP-CI firmware, you have to refer to the **GetDPPEvents** function.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_DecodeEvent(int handle,
                      char *evtPtr,
                      void **Evt
                      );
```

Arguments

Name	Description
<code>handle</code>	Device handler
<code>*evtPtr</code>	Pointer to the event data in the readout buffer (this is the pointer returned by the GetEventInfo function).
<code>**Evt</code>	Pointer to the decoded event structure. This pointer must be initialized to NULL if you want the function to allocate the memory buffer automatically. Conversely, if the memory buffer has been already allocated, this is the pointer to that memory buffer. The latter case is more efficient in terms of readout rate.

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

AllocateEvent

Description

This function allocates the memory buffer for the decoded event data. The size of the buffer is calculated in order to keep the maximum event size.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_AllocateEvent(int handle,
                        void **Evt
                        );
```

Arguments

Name	Description
<code>handle</code>	Device handler
<code>**Evt</code>	Pointer to memory buffer for the event structure.

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

FreeEvent

Description

This function releases the event memory buffer allocated by either the **DecodeEvent** or **AllocateEvent** function.



Note: If using DPP-PHA, DPP-PSD or DPP-CI firmware, you have to refer to the **GetDPPEvents** function.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_FreeEvent(int handle,
                    void **Evt
                    );
```

Arguments

Name	Description
handle	Device handler
**Evt	Pointer to memory buffer for the event structure.

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

LoadDRS4CorrectionData

Description

Regarding the x742 series, in order to compensate for unavoidable construction differences in the DRS4 chips, a data correction is required (for details, please refer to the User Manual of the board). This function loads the correction parameters stored on board, while a **DecodeEvent** function is then needed to apply them. The correction parameters to load depend on the operating sampling frequency.



Note: to be used only with x742 series.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_LoadDRS4CorrectionData (int handle,
                                  CAEN_DGTZ_DRS4Frequency_t frequency
                                  );

typedef enum
{
    CAEN_DGTZ_DRS4_5GHz      = 0L,
    CAEN_DGTZ_DRS4_2_5GHz   = 1L,
    CAEN_DGTZ_DRS4_1GHz     = 2L,
} CAEN_DGTZ_DRS4Frequency_t;
```

Arguments

Name	Description
handle	Device handler
frequency	The DRS4 sampling frequency.

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Enable/Disable DRS4Correction

Description

Enables/disables the data correction in the x742 series.



Note: to be used only with x742 series.

Note: If enabled, the data correction through the **DecodeEvent** function only applies if a **LoadDRS4CorrectionData** has been previously called, otherwise the **DecodeEvent** runs the same, but data will be provided out not compensated.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_EnableDRS4Correction (int handle);

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_DisableDRS4Correction (int handle);
```

Arguments

Name	Description
handle	Device handler

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

3 Trigger configuration

The acquisition in the digitizer is ruled by the trigger, which is a signal that decides when to start the acquisition window and save samples of the ADC or the values of interest calculated on line (DPP) in the digitizer memory.

The digitizer can have the following trigger sources: External Trigger (digital signal from the panel), Software Trigger (write access to the specific register), Self Trigger Channel (internal signal generated by a digitizer channel under certain conditions, for example when the input signal exceeds a programmable threshold).

All trigger sources can be enabled or not to generate the acquisition trigger for the channels. Similarly, it is possible to decide what triggers should participate in the generation of the Trigger Output (NIM or TTL digital output of the digitizer panel). Trigger Output can not necessarily coincide with the acquisition trigger: for example, in order to trigger multiple cards at once, as one of their channel has "auto triggered"; for this purpose, the auto triggering channel is used only to generate the Trigger Outputs (but not for the acquisition trigger); all Trigger Outputs are ORed externally to the cards and the resulting signal is sent in parallel to all cards Trigger Inputs, which are programmed to enable only the Trigger Input to generate the acquisition Trigger.



Note: in digitizer series X740, the auto trigger channel is divided into two levels: each 8-channel group generates a "group local trigger", given by the OR a of channel triggers enabled to generate them. The group triggers, in their turn, may participate or not to generate the acquisition trigger and / or trigger output.

SendSWtrigger

Description

This function sends a Software trigger to the Digitizer. The SW trigger can be used to save an acquisition window on all channels at the same time and/or to generate a pulse on the Trigger Output of the board, according to the SW trigger mode set by the "Set" function of the **Set / GetSWTriggerMode**.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SendSWtrigger (int handle);
```

Arguments

Name	Description
handle	Device handler

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Examples

WAVEDUMP Code *(To be implemented)*

Set / GetSWTriggerMode

Description

This function decides whether the trigger software should only be used to generate the acquisition trigger, only to generate the trigger output, or both.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetSWTriggerMode(int handle,
                           CAEN_DGTZ_TriggerMode_t mode
                           );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetSWTriggerMode(int handle,
                           CAEN_DGTZ_TriggerMode_t *mode);

typedef enum
{
    CAEN_DGTZ_TRGMODE_DISABLED           = 0,
    CAEN_DGTZ_TRGMODE_EXTOUT_ONLY       = 2,
    CAEN_DGTZ_TRGMODE_ACQ_ONLY          = 1,
    CAEN_DGTZ_TRGMODE_ACQ_AND_EXTOUT    = 3,
}CAEN_DGTZ_TriggerMode_t;
```

Arguments

Name	Description
handle	Device handler
mode	SW Trigger mode: CAEN_DGTZ_TRGMODE_DISABLED = 0, CAEN_DGTZ_TRGMODE_EXTOUT_ONLY = 2, CAEN_DGTZ_TRGMODE_ACQ_ONLY = 1, CAEN_DGTZ_TRGMODE_ACQ_AND_EXTOUT = 3,

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetExtTriggerInputMode

Description

This function decides whether the external trigger should only be used to generate the acquisition trigger, only to generate the trigger output, or both.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetExtTriggerInputMode(int handle,
                                  CAEN_DGTZ_TriggerMode_t mode
                                  );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetExtTriggerInputMode(int handle,
                                  CAEN_DGTZ_TriggerMode_t *mode);

typedef enum
{
    CAEN_DGTZ_TRGMODE_DISABLED           = 0,
    CAEN_DGTZ_TRGMODE_EXTOUT_ONLY       = 2,
    CAEN_DGTZ_TRGMODE_ACQ_ONLY          = 1,
    CAEN_DGTZ_TRGMODE_ACQ_AND_EXTOUT    = 3,
}CAEN_DGTZ_TriggerMode_t;
```

Arguments

Name	Description
handle	Device handler
mode	External Trigger mode CAEN_DGTZ_TRGMODE_DISABLED = 0, CAEN_DGTZ_TRGMODE_EXTOUT_ONLY = 2, CAEN_DGTZ_TRGMODE_ACQ_ONLY = 1, CAEN_DGTZ_TRGMODE_ACQ_AND_EXTOUT = 3,

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetChannelSelfTrigger

Description

This function decides whether the trigger of a channel should be used only to generate the acquisition trigger, only to generate the trigger output, or both.

For the x740 series, use the **Set / GetGroupSelfTrigger** function.

Synopsis

```
CAEN DGTZ ErrorCode CAENDGTZ API
CAEN DGTZ SetChannelSelfTrigger(int handle,
                                CAEN DGTZ TriggerMode t mode,
                                uint32_t channelmask
                                );

CAEN DGTZ ErrorCode CAENDGTZ API
CAEN DGTZ GetChannelSelfTrigger(int handle,
                                uint32_t channel,
                                CAEN_DGTZ_TriggerMode_t *mode
                                );

typedef enum
{
    CAEN_DGTZ_TRGMODE_DISABLED           = 0,
    CAEN_DGTZ_TRGMODE_EXTOUT_ONLY       = 2,
    CAEN_DGTZ_TRGMODE_ACQ_ONLY          = 1,
    CAEN_DGTZ_TRGMODE_ACQ_AND_EXTOUT    = 3,
}CAEN_DGTZ_TriggerMode_t;
```

Arguments

Name	Description
handle	Device handler
mode	Channel Self Trigger mode CAEN_DGTZ_TRGMODE_DISABLED = 0, CAEN_DGTZ_TRGMODE_EXTOUT_ONLY = 2, CAEN_DGTZ_TRGMODE_ACQ_ONLY = 1, CAEN_DGTZ_TRGMODE_ACQ_AND_EXTOUT = 3,
channelmask	(only for Set): the function applies only to those channels that have the relevant bit in the mask equal to 1
channel	(only for Get): channel for which the mode is get

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetGroupSelfTrigger

Description

This function is valid only for the x740 series. In fact, in this type of digitizer, the channels are grouped 8 by 8. The trigger properties are referred to the groups and cannot be set individually channel by channel. Each group of 8 channels generates one single self trigger which is the OR of the 8 self triggers in the group (with a programmable trigger enable mask, see next function). The group self trigger can generate the acquisition trigger for the board and/or a pulse on the Trigger Output.



Note: to be used only with x740 series.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetGroupSelfTrigger(int handle,
                              CAEN_DGTZ_TriggerMode t mode,
                              uint32 t groupmask
                              );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetGroupSelfTrigger(int handle,
                              uint32 t group,
                              CAEN_DGTZ_TriggerMode t *mode
                              );

typedef enum
{
    CAEN_DGTZ_TRGMODE_DISABLED           = 0,
    CAEN_DGTZ_TRGMODE_EXTOUT_ONLY       = 2,
    CAEN_DGTZ_TRGMODE_ACQ_ONLY          = 1,
    CAEN_DGTZ_TRGMODE_ACQ_AND_EXTOUT    = 3,
}CAEN_DGTZ_TriggerMode_t;
```

Arguments

Name	Description
handle	Device handler
mode	Group Self Trigger mode CAEN_DGTZ_TRGMODE_DISABLED = 0, CAEN_DGTZ_TRGMODE_EXTOUT_ONLY = 2, CAEN_DGTZ_TRGMODE_ACQ_ONLY = 1, CAEN_DGTZ_TRGMODE_ACQ_AND_EXTOUT = 3,
groupmask	(only for Set): the function applies only to those groups that have the relevant bit in the mask equal to 1
group	(only for Get): group for which the mode is get

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetChannelGroupMask

Description

This function decides which channels in a group of 8 participate to the generation of the self-trigger of that group. The self-trigger is the OR of the channels enabled by this function that are above the threshold. **WARNING:** the channels that are not connected must be disabled here, otherwise it may happen that one channel has a DC offset higher than the threshold and it keeps the OR always active.



Note: to be used only with x740 series.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetChannelGroupMask(int handle,
                               uint32_t group,
                               uint32_t channelmask
                               );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetChannelGroupMask(int handle,
                               uint32_t group,
                               uint32_t *channelmask
                               );
```

Arguments

Name	Description
handle	Device handler
group	Group for which the mask is set
channelmask	Channels Trigger mask for the group (8 bits)

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetChannelTriggerThreshold

Description

This function sets the Trigger Threshold for a specific channel. The threshold is applied to the digital signal after the ADC and it is expressed in ADC counts. The user should take care of the DC offset adjust when converting the digital threshold in the corresponding voltage level on the analog input.

For the x740 series, use the **Set / GetGroupTriggerThreshold** function. For the DPP firmware, use the **SetDPPParameters** function.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetChannelTriggerThreshold(int handle,
                                     uint32_t channel,
                                     uint32_t Tvalue
                                     );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetChannelTriggerThreshold(int handle,
                                     uint32_t channel,
                                     uint32_t *Tvalue
                                     );
```

Arguments

Name	Description
handle	Device handler
channel	Channel to set
Tvalue	Threshold value (in ADC counts)

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetGroupTriggerThreshold

Description

This function sets/gets the Trigger Threshold for a specified group of channel. The threshold is common to the 8 channels in the group. See the **Set / GetChannelTriggerThreshold** function for further details.



Note: to be used only with x740 series.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetGroupTriggerThreshold(int handle,
                                   uint32_t group,
                                   uint32_t Tvalue
                                   );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetGroupTriggerThreshold(int handle,
                                   uint32_t group,
                                   uint32_t *Tvalue
                                   );
```

Arguments

Name	Description
handle	Device handler
group	Group to set
Tvalue	Threshold value

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetChannelPulsePolarity

Description

Sets/gets the value of the pulse polarity for the specified channel.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetChannelPulsePolarity (int handle,
                                   uint32_t channel,
                                   CAEN_DGTZ_PulsePolarity t pol
                                   );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetChannelPulsePolarity (int handle,
                                   uint32_t channel,
                                   CAEN_DGTZ_PulsePolarity t *pol
                                   );

typedef enum
{
    CAEN_DGTZ_PulsePolarityPositive = 0,
    CAEN_DGTZ_PulsePolarityNegative = 1,
} CAEN_DGTZ_PulsePolarity_t;
```

Arguments

Name	Description
handle	Device handler
channel	The channel to set/get information for
pol/*pol	Value of the pulse polarity

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetRunSynchronizationMode

Description

Sets/gets the run synchronization mode of the digitizer, used to synchronize an acquisition on multiple boards.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetRunSynchronizationMode (int handle,
                                     CAEN_DGTZ_RunSyncMode_t mode
                                    );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetRunSynchronizationMode (int handle,
                                     CAEN_DGTZ_RunSyncMode_t *mode
                                    );

typedef enum
{
    CAEN_DGTZ_RUN_SYNC_Disabled,
    CAEN_DGTZ_RUN_SYNC_TriggerOutTriggerInDaisyChain,
    CAEN_DGTZ_RUN_SYNC_TriggerOutSinDaisyChain,
    CAEN_DGTZ_RUN_SYNC_SinFanout,
    CAEN_DGTZ_RUN_SYNC_GpioGpioDaisyChain
} CAEN_DGTZ_RunSyncMode_t;
```

Arguments

Name	Description
handle	Device handler
mode/*mode	The run synchronization mode to set/get

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetIOLevel

Description

Sets/gets the I/O level.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetIOLevel (int handle,
                     CAEN_DGTZ_IOLevel_t level
                    );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetIOLevel (int handle,
                     CAEN_DGTZ_IOLevel_t *level
                    );

typedef enum
{
    CAEN_DGTZ_IOLevel_NIM = 0L,
    CAEN_DGTZ_IOLevel_TTL = 1L,
} CAEN_DGTZ_IOLevel_t;
```

Arguments

Name	Description
handle	Device handler
level/*level	The I/O level of the digitizer to set/get

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetTriggerPolarity

Description

Sets/gets the trigger polarity of a specified channel.



Note: not to be used with DPP firmware.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetTriggerPolarity (int handle,
                              uint32_t channel,
                              CAEN_DGTZ_TriggerPolarity_t Polarity
                              );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetTriggerPolarity (int handle,
                              uint32_t channel,
                              CAEN_DGTZ_TriggerPolarity_t *Polarity
                              );

typedef enum
{
    CAEN_DGTZ_TriggerOnRisingEdge = 0L,
    CAEN_DGTZ_TriggerOnFallingEdge = 1L,
} CAEN_DGTZ_TriggerPolarity_t;
```

Arguments

Name	Description
handle	Device handler
channel/*channel	Selects the channel to set/get the trigger polarity for
Polarity/*Polarity	The polarity of the trigger to set/get

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetGroupFastTriggerThreshold

Description

Sets/gets the threshold value on TRn input (used as external trigger) for the local trigger generation in x742 series. As the threshold is an hardware threshold (input of a programmable 16-bit DAC, whose voltage output goes to a comparator), it is not easy to set and the user can refer to the board User Manual for setting examples.



Note: to be used only with x742 series.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetGroupFastTriggerThreshold (int handle,
                                         uint32_t group,
                                         uint32_t Tvalue
                                         );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetGroupFastTriggerThreshold (int handle,
                                         uint32_t group,
                                         uint32_t *Tvalue
                                         );
```

Arguments

Name	Description
handle	Device handler
group	The channels group the threshold is applied to
Tvalue/*Tvalue	The value of the TRn threshold to set/get

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetGroupFastTriggerDCOffset

Description

Regarding the x742 series, sets/gets the TRn signal DC offset when it is sampled in the DRS4 chips in order to make positive, negative or bipolar input signals to be compliant with the DRS4 input dynamics. The DC offset also affects the TRn when used as trigger, in this case it relates to the threshold setting above described (please refer to the board User Manual for setting examples).



Note: to be used only with x742 series.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetGroupFastTriggerDCOffset (int handle,
                                       uint32_t group,
                                       uint32_t DCvalue
                                       );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetGroupFastTriggerDCOffset (int handle,
                                       uint32_t group,
                                       uint32_t *DCvalue
                                       );
```

Arguments

Name	Description
handle	Device handler
group	The channels group the DC offset is applied to
DCvalue/*DCvalue	The value of the TRn DC offset to set/get

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetFastTriggerDigitizing

Description

Regarding the x742 series, enables/disables (set) the presence of the TRn signal in the data readout as well as allows for checking the status of the setting (get).



Note: to be used only with x742 series.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetFastTriggerDigitizing (int handle,
                                    CAEN_DGTZ_EnaDis_t enable
                                    );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetFastTriggerDigitizing (int handle,
                                    CAEN_DGTZ_EnaDis_t *enable
                                    );

typedef enum
{
    CAEN_DGTZ_ENABLE = 1L,
    CAEN_DGTZ_DISABLE = 0L,
} CAEN_DGTZ_EnaDis_t;
```

Arguments

Name	Description
handle	Device handler
enable/*enable	The enable flag to set/get

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetFastTriggerMode

Description

Enables/disables (set) the TRn input as local trigger in x742 series, as wells allows for checking the status of the setting (get).



Note: to be used only with x742 series.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetFastTriggerMode (int handle,
                              CAEN_DGTZ_TriggerMode t mode
                              );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetFastTriggerMode (int handle,
                              CAEN_DGTZ_TriggerMode t *mode
                              );

typedef enum
{
    CAEN_DGTZ_TRGMODE_DISABLED = 0L,
    CAEN_DGTZ_TRGMODE_ACQ_ONLY = 1L,
}CAEN_DGTZ_TriggerMode_t;
```

Arguments

Name	Description
handle	Device handler
mode/*mode	The fast trigger value to set/get.

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetDRS4SamplingFrequency

Description

Regarding the x742 series, sets/gets the sampling frequency of the DRS4 chips which sample the input analog signal and the fast trigger signal.



Note: to be used only with x742 series.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetDRS4SamplingFrequency (int handle,
                                     CAEN_DGTZ_DRS4Frequency_t frequency
                                     );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetDRS4SamplingFrequency (int handle,
                                     CAEN_DGTZ_DRS4Frequency_t *frequency
                                     );

typedef enum
{
    CAEN_DGTZ_DRS4_5GHz = 0L,
    CAEN_DGTZ_DRS4_2_5GHz = 1L,
    CAEN_DGTZ_DRS4_1GHz = 2L,
} CAEN_DGTZ_DRS4Frequency_t;
```

Arguments

Name	Description
handle	Device handler
frequency/*frequency	The sampling frequency value to set/get

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetOutputSignalMode

Description

Sets/gets the signal to be provided out over the TRG-OUT output channel in the x742 series.

Note: to be used only with x742 series.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetOutputSignalMode (int handle,
                               CAEN_DGTZ_OutputSignalMode_t mode
                              );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetOutputSignalMode (int handle,
                               CAEN_DGTZ_OutputSignalMode_t *mode
                              );

typedef enum
{
    CAEN_DGTZ_TRIGGER                = 0L,
    CAEN_DGTZ_FASTTRG_ALL            = 1L,
    CAEN_DGTZ_FASTTRG_ACCEPTED      = 2L,
    CAEN_DGTZ_BUSY                   = 3L,
} CAEN_DGTZ_OutputSignalMode_t;
```

Arguments

Name	Description
<code>handle</code>	Device handler
<code>mode/*mode</code>	The output signal mode to set/get.

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

4 Acquisition

Set / GetChannelEnableMask

Description

This function enables/disables the channels for the acquisition. Disabled channels don't give any trigger and don't participate to the event data.

For the x740 and x742 series, use the **Set / GetGroupEnableMask** function

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetChannelEnableMask(int handle,
                               uint32_t mask
                              );
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetChannelEnableMask(int handle,
                               uint32_t *mask
                              );
```

Arguments

Name	Description
handle	Device handler
mask	Enable Mask. Bit n (with 0 <= n <= 7) corresponds to channel n.

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Examples

WAVEDUMP Code *(To be implemented)*

Set / GetGroupEnableMask

Description

This function enables/disables the groups for the acquisition. This function is valid only for the x740 and x742 series. Disabled groups don't give any trigger and don't participate to the event data. The 8 channel in a group are all enabled/disabled according to the relevant bit in the enable mask.



Note: to be used only with x740 and x742 series.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetGroupEnableMask(int handle,
                              uint32_t mask
                             );
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetGroupEnableMask(int handle,
                              uint32_t *mask
                              );
```

Arguments

Name	Description
handle	Device handler
mask	Enable Mask. Bit n (with 0 <= n <= 7) corresponds to group n.

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

SWStartAcquisition

Description

This function starts the acquisition in a board using a software command. When the acquisition starts, the relevant RUN LED on the front panel lights up. It is worth noticing that in case of multiple board systems, the software start doesn't allow the digitizer to start synchronously. For this purpose, it is necessary to use to start the acquisition using a physical signal, such as the S-IN or GPI as well as the TRG-IN-TRG-OUT Daisy chain. Please refer to Digitizer manual for more details on this issue.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SWStartAcquisition(int handle);
```

Arguments

Name	Description
handle	Device handler

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

SWStopAcquisition

Description

This function stops the acquisition in a board using a software command.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SWStopAcquisition(int handle);
```

Arguments

Name	Description
handle	Device handler

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetRecordLength

Description

This function sets the size of the acquisition window, that is the number of samples that belong to it. Due to the way the samples are written into the memory (more samples are put in parallel), there is a specific granularity of the record length depending on the board model. For example, in the x720 series, the samples are written 4 by 4, hence the record length must be a multiple of 4. The function accepts any value for the parameter size and then takes the closest value multiple of the granularity. The function **GetRecordLength** returns the exact value.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetRecordLength (int handle,
                          uint32_t size,
                          ...
                          );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetRecordLength (int handle,
                          uint32_t *size,
                          ...
                          );
```

Arguments

Name	Description
handle	Device handler
size/*size	The size of the record (in samples) to set/get
channel (optional)	A <i>int</i> specifying the channel to set/get the record length for. Used only for digitizers running DPP firmware, in particular DPP-PSD and DPP-CI

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetPostTriggerSize

Description

This function sets the post trigger size, that is the position of the trigger within the acquisition window. The size is expressed in percentage of the record length. 0% means that the trigger is at the end of the window, while 100% means that it is at the beginning.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetPostTriggerSize(int handle,
                             uint32_t percent
                             );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetPostTriggerSize(int handle,
                              uint32_t *percent
                              );
```

Arguments

Name	Description
handle	Device handler
percent	Post trigger in percent of the record length

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetAcquisitionMode

Description

Gets/Sets digitizer acquisition mode.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetAcquisitionMode(int handle,
                              CAEN_DGTZ_AcqMode_t mode
                              );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetAcquisitionMode(int handle,
                              CAEN_DGTZ_AcqMode_t *mode
                              );

typedef enum
{
    CAEN_DGTZ_SW_CONTROLLED = 0,
    CAEN_DGTZ_S_IN_CONTROLLED = 1,
}CAEN_DGTZ_AcqMode_t;
```

Arguments

Name	Description
handle	Device handler
mode	The acquisition mode CAEN_DGTZ_SW_CONTROLLED or CAEN_DGTZ_S_IN_CONTROLLED

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetChannelDCOffset

Description

This function sets the 16 bit DAC that adds a DC offset to the input signal in order to adapt it to the dynamic range of the ADC. By default, the DAC is set to middle scale (0x8000) which corresponds to a DC offset of $-V_{pp}/2$, where V_{pp} is the voltage range (peak to peak) of the ADC. This means that the input signal can range from $-V_{pp}/2$ to $+V_{pp}/2$. If the DAC is set to 0x0000, then no DC offset is added and the range of the input signal goes from 0 to $+V_{pp}$. Conversely, when the DAC is set to 0xFFFF, the DC offset is $-V_{pp}$ and the range goes from $-V_{pp}$ to 0. The DC offset can be set on channel basis except for the x740 in which it is set on group basis; in this case, you must use the **Set / GetGroupDCOffset** functions.

Synopsis

```
CAEN DGTZ ErrorCode CAENDGTZ API
CAEN DGTZ SetChannelDCOffset(int handle,
                             uint32_t channel,
                             uint32_t Tvalue
                             );

CAEN DGTZ ErrorCode CAENDGTZ API
CAEN DGTZ GetChannelDCOffset(int handle,
                              uint32_t channel,
                              uint32_t *Tvalue
                              );
```

Arguments

Name	Description
handle	Device handler
channel	Channel to which the DAC setting is applied. Use -1 for all channels
Tvalue	DAC value (from 0x0000 to 0xFFFF)

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetGroupDCOffset

Description

The same as Set/Get ChannelDCOffset, but in this case it is applied to the groups of the x740 series.



Note: to be used only with x740 series.

Synopsis

```
CAEN DGTZ ErrorCode CAENDGTZ API
CAEN DGTZ SetGroupDCOffset(int handle,
                            uint32_t group,
                            uint32_t Tvalue
                            );

CAEN DGTZ ErrorCode CAENDGTZ API
CAEN_DGTZ_GetGroupDCOffset(int handle,
                             uint32_t group,
                             uint32_t *Tvalue
                             );
```

Arguments

Name	Description
handle	Device handler
group	Group to which the DAC setting is applied. Use -1 for all groups
Tvalue	DAC value (from 0x0000 to 0xFFFF)

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetDESMode

Description

This function enables or disables the Dual Edge Sampling mode, that is the channel interleaving option to double the sampling frequency. This option is available in the x731 and x751 series only.



WARNING: when the DES mode is enabled, only the odd channels (for the x751) or the even channels (for the x731) will work; the other channels must be left unconnected

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetDESMode(int handle,
                     CAEN_DGTZ_EnaDis_t mode);

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetDESMode(int handle,
                     CAEN_DGTZ_EnaDis_t *mode);

typedef enum
{
    CAEN_DGTZ_ENABLE      = 1,
    CAEN_DGTZ_DISABLE    = 0,
} CAEN_DGTZ_EnaDis_t;
```

Arguments

Name	Description
<code>handle</code>	Device handler
<code>mode</code>	CAEN_DGTZ_ENABLE to enable the DES mode, CAEN_DGTZ_DISABLE to disable the DES mode

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetZeroSuppressionMode

Description

sets/gets the Zero Suppression mode.

Synopsis

```
CAEN DGTZ ErrorCode CAENDGTZ API
CAEN_DGTZ_SetZeroSuppressionMode(int handle,
                                  CAEN_DGTZ_ZS_Mode_t mode
                                  );

CAEN DGTZ ErrorCode CAENDGTZ API
CAEN DGTZ GetZeroSuppressionMode (int handle,
                                   CAEN_DGTZ_ZS_Mode_t *mode
                                   );

typedef enum
{
    CAEN_DGTZ_ZS_NO           = 0,
    CAEN_DGTZ_ZS_INT         = 1,
    CAEN_DGTZ_ZS_ZLE         = 2,
    CAEN_DGTZ_ZS_AMP         = 3,
} CAEN_DGTZ_ZS_Mode_t;
```

Arguments

Name	Description
handle	Device handler
mode	Zero Suppression Mode : CAEN_DGTZ_ZS_NO = 0 (no Zero suppression), CAEN_DGTZ_ZS_INT = 1 (Full Suppression based on the integral of the signal), CAEN_DGTZ_ZS_ZLE = 2 (Zero Length Encoding), CAEN_DGTZ_ZS_AMP = 3 (Full Suppression based on the signal amplitude),

Supported digitizers and permitted zero suppression modes

Digitizer	0	1	2	3
x720	X		X	X
v1721/v1731	X		X	X
x724	X	X	X	X

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetChannelZSParams

Description

Sets/Gets Zero Suppression parameters for a specific channel in the supported digitizers (see the table in the **Set / GetZeroSuppressionMode** functions).



Note: the **Set / GetChannelZSParams** functions are to be used in combination with **Set / GetTriggerPolarity** and **Set / GetZeroSuppressionMode** functions which relate to the trigger polarity logic and the zero suppression algorithm.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetChannelZSParams(int handle,
                             uint32_t channel,
                             CAEN_DGTZ_ThresholdWeight_t weight,
                             int32_t threshold,
                             int32_t nsamp
                             );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetChannelZSParams(int handle,
                             uint32_t channel,
                             CAEN_DGTZ_ThresholdWeight_t *weight,
                             int32_t *threshold,
                             int32_t *nsamp
                             );

typedef enum
{
    CAEN_DGTZ_ZS_FINE = 0,
    CAEN_DGTZ_ZS_COARSE = 1,
}CAEN_DGTZ_ThresholdWeight_t;
```

Arguments

Name	Description
handle	Device handler
channel	Channel to which the ZS settings are applied. Use -1 for all channels
weight	Zero Suppression weight*. Used in “Full Suppression based on the integral of the signal” supported only by x724 series. CAEN_DGTZ_ZS_FINE = 0 (Fine threshold step; the threshold is the <i>threshold</i> parameter), CAEN_DGTZ_ZS_COARSE = 1 (Coarse threshold step; the threshold is <i>threshold</i> × 64) For “Full Suppression based on the signal amplitude” and “Zero Length Encoding” algorithms, the value of <i>weight</i> doesn’t affect the function working.
threshold	Zero Suppression Threshold to be used depending on the ZS algorithm*.
nsamp	Number of samples to be used by the ZS algorithm*.

*Refer to the digitizer User Manual for definition and representation.

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetAnalogMonOutput

Description

Sets/Gets the signal to output on the Analog Monitor Front Panel output in VME digitizers running standard firmware.



Note: the function is not supported by V1742 and digitizers running DPP firmware.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetAnalogMonOutput(int handle,
                             CAEN_DGTZ_AnalogMonitorOutputMode t mode
                             );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetAnalogMonOutput(int handle,
                              CAEN_DGTZ_AnalogMonitorOutputMode_t *mode
                              );

typedef enum
{
    CAEN_DGTZ_AM_TRIGGER_MAJORITY    = 0,
    CAEN_DGTZ_AM_TEST                = 1,
    CAEN_DGTZ_AM_ANALOG_INSPECTION   = 2,
    CAEN_DGTZ_AM_BUFFER_OCCUPANCY    = 3,
    CAEN_DGTZ_AM_VOLTAGE_LEVEL       = 4,
}CAEN_DGTZ_AnalogMonitorOutputMode_t;
```

Arguments

Name	Description
handle	Device handler
mode	Analog Monitor Mode CAEN_DGTZ_AM_TRIGGER_MAJORITY = 0 (Trigger Majority Mode), CAEN_DGTZ_AM_TEST = 1 (Test Mode), CAEN_DGTZ_AM_ANALOG_INSPECTION = 2 (Analog Inspection Mode), CAEN_DGTZ_AM_BUFFER_OCCUPANCY = 3 (Buffer Occupancy Mode), CAEN_DGTZ_AM_VOLTAGE_LEVEL = 4 (Voltage Level Mode),

Supported digitizers and permitted AM modes

Digitizer	0	1	2	3	4
V1720-V1721-V1731-V1740-V1751	X	X		X	X
V1724	X	X	X	X	X

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetAnalogInspectionMonParams

Description

Sets/Gets the Analog Inspection Monitor parameters for a V1724 digitizer running standard firmware.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetAnalogInspectionMonParams(int handle,
                                        uint32_t channelmask,
                                        uint32_t offset,
                                        CAEN_DGTZ_AnalogMonitorMagnify_t mf,
                                        CAEN_DGTZ_AnalogMonitorInspectorInverter_t ami
                                        );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetAnalogInspectionMonParams(int handle,
                                        uint32_t channelmask,
                                        uint32_t *offset,
                                        CAEN_DGTZ_AnalogMonitorMagnify_t *mf,
                                        CAEN_DGTZ_AnalogMonitorInspectorInverter_t *ami
                                        );

typedef enum
{
    CAEN_DGTZ_AM_MAGNIFY_1X           = 0,
    CAEN_DGTZ_AM_MAGNIFY_2X           = 1,
    CAEN_DGTZ_AM_MAGNIFY_4X           = 2,
    CAEN_DGTZ_AM_MAGNIFY_8X           = 3,
}CAEN_DGTZ_AnalogMonitorMagnify_t;

typedef enum
{
    CAEN_DGTZ_AM_INSPECTORINVERTER_P_1X = 0,
    CAEN_DGTZ_AM_INSPECTORINVERTER_N_1X = 1,
}CAEN_DGTZ_AnalogMonitorInspectorInverter_t;
```

Arguments

Name	Description
handle	Device handler
channelmask	channel enable mask
offset	DC Offset for the analog output signal
mf	Multiply factor
ami	Invert Output

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Set / GetEventPackaging

Description

This function allows to enable or disable the Pack 2.5 mode of V1720/DT5720 Digitizers

Synopsis

```
CAEN DGTZ ErrorCode CAENDGTZ API
CAEN_DGTZ_SetEventPackaging(int handle,
                             CAEN_DGTZ_EnaDis_t mode
                             );

CAEN DGTZ ErrorCode CAENDGTZ API
CAEN DGTZ GetEventPackaging(int handle,
                             CAEN_DGTZ_EnaDis_t *mode
                             );

typedef enum {
                                CAEN_DGTZ_ENABLE = 1L,
                                CAEN_DGTZ_DISABLE = 0L,
} CAEN_DGTZ_EnaDis_t;
```

Arguments

Name	Description
handle	Device handler
mode	Enable/Disable the Pack 2.5 mode

Return Values

0: Success; Negative numbers are error codes (see Return Codes).

Acquisition example

The following example shows how to perform a data acquisition cycle. CAEN provides this example also as source files and project inside the CAENDigitizer full installation package, compliant to Visual Studio Professional 2010.

```
#include <stdio.h>
#include "CAENDigitizer.h"

#include "keyb.h"

#define CAEN USE DIGITIZERS
#define IGNORE DPP DEPRECATED

#define MAXNB 1 /* Number of connected boards */

int checkCommand() {
    int c = 0;
    if(!kbhit())
        return 0;

    c = getch();
    switch (c) {
        case 's':
            return 9;
            break;
        case 'k':
            return 1;
            break;
        case 'q':
            return 2;
            break;
    }
    return 0;
}

int main(int argc, char* argv[])
{
    /* The following variable is the type returned from most of CAENDigitizer
    library functions and is used to check if there was an error in function
    execution. For example:
    ret = CAEN_DGTZ_some_function(some_args);
    if(ret) printf("Some error"); */
    CAEN_DGTZ_ErrorCode ret;

    /* The following variable will be used to get an handler for the digitizer. The
    handler will be used for most of CAENDigitizer functions to identify the board */
    int handle[MAXNB];

    CAEN_DGTZ_BoardInfo t BoardInfo;
    CAEN_DGTZ_EventInfo t eventInfo;
    CAEN_DGTZ_UINT16_EVENT_t *Evt = NULL;
    char *buffer = NULL;

    int i,b;
    int c = 0, count[MAXNB];
    char * evtptr = NULL;
    uint32_t size,bsize;
    uint32_t numEvents;
    i = sizeof(CAEN_DGTZ_TriggerMode_t);

    for(b=0; b<MAXNB; b++){
        /* IMPORTANT: The following function identifies the different boards with a system
        which may change
        for different connection methods (USB, Conet, ecc). Refer to CAENDigitizer user
        manual for more info.
        brief:
        CAEN_DGTZ_OpenDigitizer(<LikType>, <LinkNum>, <ConetNode>, <VMEBaseAddress>,
        <*handler>);
        Some examples below */

        /* The following is for b boards connected via b USB direct links
        in this case you must set <LikType> = CAEN_DGTZ_USB and <ConetNode> =
        <VMEBaseAddress> = 0 */
        //ret = CAEN_DGTZ_OpenDigitizer(CAEN_DGTZ_USB, b, 0, 0, &handle[b]);
    }
}
```

```

/* The following is for b boards connected via 1 opticalLink in dasy chain
in this case you must set <LikType> = CAEN DGTZ PCI OpticalLink and <LinkNum> =
<VMEBaseAddress> = 0 */
//ret = CAEN DGTZ OpenDigitizer(Params[b].LinkType, 0, b, Params[b].VMEBaseAddress,
&handle[b]);

/* The following is for b boards connected to A2818 (or A3818) via opticalLink (or
USB with A1718)
in this case the boards are accessed through VME bus, and you must specify the VME
address of each board:
<LikType> = CAEN_DGTZ_PCI_OpticalLink (CAEN_DGTZ_PCIE_OpticalLink for A3818 or
CAEN_DGTZ_USB for A1718)
<LinkNum> must be the bridge identifier
<ConetNode> must be the port identifier in case of A2818 or A3818 (or 0 in case of
A1718)
<VMEBaseAddress>[0] = <0XXXXXXXX> (address of first board)
<VMEBaseAddress>[1] = <0YYYYYYYY> (address of second board)
...
<VMEBaseAddress>[b-1] = <0ZZZZZZZZ> (address of last board)
See the manual for details */
ret = CAEN DGTZ OpenDigitizer(CAEN DGTZ USB,0,0,0x11110000,&handle[b]);
if(ret != CAEN_DGTZ_Success) {
    printf("Can't open digitizer\n");
    goto QuitProgram;
}
/* Once we have the handler to the digitizer, we use it to call the other functions
*/
ret = CAEN_DGTZ_GetInfo(handle[b], &BoardInfo);
printf("\nConnected to CAEN Digitizer Model %s, recognized as board %d\n",
BoardInfo.ModelName, b);
printf("\tROC FPGA Release is %s\n", BoardInfo.ROC FirmwareRel);
printf("\tAMC FPGA Release is %s\n", BoardInfo.AMC FirmwareRel);

ret = CAEN_DGTZ_Reset(handle[b]); /*
Reset Digitizer */
ret = CAEN DGTZ GetInfo(handle[b], &BoardInfo);
/* Get Board Info */
ret = CAEN DGTZ SetRecordLength(handle[b],4096);
/* Set the length of each waveform (in samples) */
ret = CAEN_DGTZ_SetChannelEnableMask(handle[b],1);
/* Enable channel 0 */
ret = CAEN DGTZ SetChannelTriggerThreshold(handle[b],0,32768);
/* Set self trigger threshold */
ret = CAEN_DGTZ_SetChannelSelfTrigger(handle[b],CAEN_DGTZ_TRGMODE
_ACQ_ONLY,1); /* Set trigger on channel 0 to be ACQ_ONLY
*/
ret = CAEN DGTZ SetSWTriggerMode(handle[b],CAEN DGTZ TRGMODE ACQ ONLY);
/* Set the behaviour when a SW trigger arrives */
ret = CAEN_DGTZ_SetMaxNumEventsBLT(handle[b],3);
/* Set the max number of events to transfer in a single
readout */
ret = CAEN DGTZ SetAcquisitionMode(handle[b],CAEN DGTZ SW CONTROLLED); /*
Set the acquisition mode */
if(ret != CAEN_DGTZ_Success) {
    printf("Errors during Digitizer Configuration.\n");
    goto QuitProgram;
}
}
printf("\n\nPress 's' to start the acquisition\n");
printf("Press 'k' to stop the acquisition\n");
printf("Press 'q' to quit the application\n\n");
while (1) {
    c = checkCommand();
    if (c == 9) break;
    if (c == 2) return;
    Sleep(100);
}
/* Malloc Readout Buffer.
NOTE1: The mallocs must be done AFTER digitizer's configuration!
NOTE2: In this example we use the same buffer, for every board. We
Use the first board to allocate the buffer, so if the configuration
is different for different boards (or you use different board models), may be
that the size to allocate must be different for each one. */
ret = CAEN DGTZ MallocReadoutBuffer(handle[0],&buffer,&size);

for(b=0; b<MAXNB; b++)

```



```

/* Start Acquisition
NB: the acquisition for each board starts when the following line is executed
so in general the acquisition does NOT starts synchronously for different boards
*/
ret = CAEN_DGTZ_SWStartAcquisition(handle[b]);

// Start acquisition loop
while(1) {
for(b=0; b<MAXNB; b++) {
    ret = CAEN DGTZ SendSWtrigger(handle[b]); /* Send a SW
    Trigger */

    ret
    CAEN DGTZ ReadData(handle[b],CAEN DGTZ SLAVE TERMINATED REA
    DOUT MBLT,buffer,&bsize); /* Read the buffer from the
    digitizer */

/* The buffer red from the digitizer is used in the other functions to get the
event data
The following function returns the number of events in the buffer */
ret
CAEN_DGTZ_GetNumEvents (handle[b],buffer,bsize,&numEvents);

    printf(".");
    count[b] +=numEvents;
    for (i=0;i<numEvents;i++) {
/* Get the Infos and pointer to the event */
ret
CAEN_DGTZ_GetEventInfo (handle[b],buffer,bsize,i,&eventInfo,
&evtptr);

/* Decode the event to get the data */
ret = CAEN_DGTZ_DecodeEvent (handle[b],evtptr,&Evt);
//*****
// Event Elaboration
//*****
ret = CAEN DGTZ FreeEvent (handle[b],&Evt);
    }
    c = checkCommand();
    if (c == 1) goto Continue;
    if (c == 2) goto Continue;
    } // end of loop on boards
} // end of readout loop

Continue:
for(b=0; b<MAXNB; b++)
    printf("\nBoard %d: Retrieved %d Events\n",b, count[b]);
goto QuitProgram;

/* Quit program routine */
QuitProgram:
// Free the buffers and close the digitizers
ret = CAEN DGTZ FreeReadoutBuffer (&buffer);
for(b=0; b<MAXNB; b++)
    ret = CAEN_DGTZ_CloseDigitizer (handle[b]);
    printf("Press 'Enter' key to exit\n");
    c = getchar();
    return 0;
}

```

x742 Offline data correction functions

In the installation package of CAENDigitizer library, additional functions are provided inside the “Sample” folder to let the user perform offline the correction of raw data acquired with x742 digitizers.



Note: The functions are not included in the CAENDigitizer run time library and are intended also for offline use.

LoadCorrectionTables

Description

Loads the correction table stored onto the board into a user defined structure.

Synopsis

```
int32_t LoadCorrectionTables(int handle, DataCorrection_t *Table,
                             uint8_t group,
                             uint32_t frequency
                             );

typedef struct {
    int16_t      cell[MAX_X742_CHANNELS+1][1024];
    int8_t       nsample[MAX_X742_CHANNELS+1][1024];
    float        time[1024];
} DataCorrection_t;
```

Arguments

Name	Description
handle	Device handle
Table	Pointer to the structure to be filled with the correction table values
group	Channel group
frequency	DSR4 sampling frequency

Return Values

0: Success.

ApplyDataCorrection

Description

Applies the desired correction data (configured through a mask) to the raw data acquired by the user.

Synopsis

```
void ApplyDataCorrection(DataCorrection_t* CTable,
                        CAEN_DGTZ_DRS4Frequency_t frequency,
                        int CorrectionLevelMask,
                        CAEN_DGTZ_X742_GROUP_t *data
                        );

typedef struct {
    int16_t      cell[MAX_X742_CHANNELS+1][1024];
    int8_t       nsample[MAX_X742_CHANNELS+1][1024];
    float        time[1024];
} DataCorrection_t;
```

Arguments

Name	Description
CTable	Pointer to the structure filled with the correction data
frequency	DSR4 sampling frequency
CorrectionLevelMask	Mask for the correction type to be applied (3-bit): Bit0 = Cell Offset correction Bit1 = Index Sampling correction Bit2= Time correction
data	Raw acquired data to be corrected

Return Values

0: Success.

GetNumEvents

Description

Gets the current number of events stored in the acquisition buffer.

Synopsis

```
int32_t GetNumEvents(char *buffer,
                    uint32_t buffersize,
                    uint32_t *numEvents
                    );
```

Arguments

Name	Description
buffer	Address of the acquisition buffer
buffersize	Size of the data stored in the acquisition buffer
numEvents	Number of events stored in the acquisition buffer

Return Values

0: Success.

GetEventPtr

Description

Retrieves the event pointer of a specified event in the acquisition buffer.

Synopsis

```
int32_t GetEventPtr(char *buffer,
                   uint32_t buffersize,
                   int32_t numEvent,
                   char **EventPtr
                   );
```

Arguments

Name	Description
buffer	Address of the acquisition buffer
buffersize	Acquisition buffer size
numEvent	Index of the requested event
EventPtr	Pointer to the requested event in the acquisition buffer

Return Values

0: Success.

X742_DecodeEvent

Description

Decodes a specified event stored in the acquisition buffer writing data in Evt memory.



Note: Once used, the Evt memory MUST be deallocated by the caller.

Synopsis

```
int32_t X742_DecodeEvent(char *evtPtr,
                        void **Evt
                        );
```

Arguments

Name	Description
evtPtr	Pointer to the requested event in the acquisition buffer
Evt	Pointer to the event structure with decoded event (MUST BE NULL)

Return Values

0: Success.

5 DPP specific functions

In order to handle acquisitions with the DPP firmware (PHA, PSD, CI), the C functions described in this chapter can be used.

Set / GetDPPPreTriggerSize

Description

Sets/gets the pre-trigger size, which is the portion of acquisition window visible before a trigger.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetDPPPreTriggerSize (int handle,
                                int ch,
                                uint32_t samples
                                );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetDPPPreTriggerSize (int handle,
                                int ch,
                                uint32_t *samples
                                );
```

Arguments

Name	Description
handle	Device handler
samples/*samples	The size of the record (in samples)
ch	The channel whose pre-trigger has to be set/get. ch=-1 writes the same value for all channels. DPP-CI only supports ch=-1 (different channels must have the same pre-trigger)

Return Values

0: Success; negative numbers are error codes (see Return Codes).

GetDPPEvents

Description

Decodes and returns all the DPP events stored in the acquisition buffers.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetDPPEvents (int handle,
                        char *buffer,
                        uint32_t buffsize,
                        void **events,
                        uint32_t *numEventsArray
                        );
```

Arguments

Name	Description
handle	Device handler
*buffer	The address of the acquisition buffer
buffsize	The acquisition buffer size (in samples)
**events	The pointer to the event list (allocated via MallocDPPEvents)
*numEventsArray	The pointer to an array of int which will contain the number of events found per channel

Return Values

0: Success; negative numbers are error codes (see Return Codes).

MallocDPPEvents

Description

Allocates the event buffer matrix which is handled by the **GetDPPEvents** function. The matrix has one event array per channel and must be declared as a MAX_CH-sized array of pointers.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_MallocDPPEvents (int handle,
                          void **events,
                          uint32_t *allocatedSize
                          );

typedef struct
{
    uint32_t Format;
    uint64_t TimeTag;
    uint16_t Energy;
    int16_t Extras;
    uint32_t *Waveforms; /*!< pointer to coded data inside the readout buffer. only meant
                        to be supplied to CAEN DGTZ DecodeDPPWaveforms */
} CAEN_DGTZ_DPP_PHA_Event_t;

typedef struct
{
    uint32_t Format;
    uint32_t TimeTag;
    int16_t ChargeShort;
    int16_t ChargeLong;
    int16_t Baseline;
    int16_t Pur;
    uint32_t *Waveforms; /*!< pointer to coded data inside the readout buffer. only meant
                        to be supplied to CAEN DGTZ DecodeDPPWaveforms */
} CAEN_DGTZ_DPP_PSD_Event_t;

typedef struct
{
    uint32_t Format;
    uint32_t TimeTag;
    int16_t Charge;
    int16_t Baseline;
    uint32_t *Waveforms; /*!< pointer to coded data inside the readout buffer. only meant
                        to be supplied to CAEN_DGTZ_DecodeDPPWaveforms */
} CAEN_DGTZ_DPP_CI_Event_t;
```

Arguments

Name	Description
handle	Device handler
**events	The pointer to the event matrix, which shall be of type: CAEN_DGTZ_DPP_PHA_Event_t, for DPP-PHA, CAEN_DGTZ_DPP_PSD_Event_t, for DPP-PSD CAEN_DGTZ_DPP_CI_Event_t, for DPP-CI
*allocatedSize	The size in bytes of the event list

Return Values

0: Success; negative numbers are error codes (see Return Codes).

FreeDPPEvents

Description

Deallocates the event buffer matrix.

Synopsis

```
CAEN DGTZ ErrorCode CAENDGTZ API
CAEN_DGTZ_FreeDPPEvents (int handle,
                        void **events
                        );
```

Arguments

Name	Description
handle	Device handler
**events	The pointer to the event buffer

Return Values

0: Success; negative numbers are error codes (see Return Codes).

MallocDPPWaveforms

Description

Allocates the waveform buffer, which is used by **CAEN_DGTZ_DecodeDPPWaveforms**.

Synopsis

```
CAEN DGTZ ErrorCode CAENDGTZ API
CAEN_DGTZ_MallocDPPWaveforms (int handle,
                               void **waveforms,
                               uint32_t *allocatedSize
                               );

typedef struct
{
    uint32_t Ns;
    uint8_t DualTrace;
    uint8_t VProbe1;
    uint8_t VProbe2;
    uint8_t VProbe;
    int16_t *Trace1;
    int16_t *Trace2;
    uint8_t *DTrace1;
    uint8_t *DTrace2;
} CAEN DGTZ DPP PHA Waveforms t;

typedef struct
{
    uint32_t Ns;
    uint8_t dualTrace;
    uint8_t anlgProbe;
    uint8_t dgtProbe1;
    uint8_t dgtProbe2;
    int16_t *Trace1;
    int16_t *Trace2;
    uint8_t *DTrace1;
    uint8_t *DTrace2;
    uint8_t *DTrace3;
    uint8_t *DTrace4;
} CAEN DGTZ DPP PSD Waveforms t;

#define CAEN DGTZ DPP CI Waveforms t CAEN DGTZ DPP PSD Waveforms t /*!< \brief Waveform
                               types for DPP-CI and DPP-PSD are the same, hence this
                               define */
```

Arguments

Name	Description
handle	Device handler
**waveforms	The pointer to the waveform buffer, which shall be of type: CAEN_DGTZ_DPP_PHA_Waveforms_t, for DPP-PHA CAEN_DGTZ_DPP_PSD_Waveforms_t, for DPP-PSD CAEN_DGTZ_DPP_CI_Waveforms_t, for DPP-CI
*allocatedSize	The size in bytes of the waveform buffer

Return Values

0: Success; negative numbers are error codes (see Return Codes).

FreeDPPWaveforms

Description

Deallocates the waveform buffer.

Synopsis

```
CAEN DGTZ ErrorCode CAENDGTZ API
CAEN_DGTZ_FreeDPPWaveforms (int handle,
                             void *waveforms
                             );
```

Arguments

Name	Description
handle	Device handler
*waveforms	The pointer to the waveform buffer

Return Values

0: Success; negative numbers are error codes (see Return Codes).

DecodeDPPWaveforms

Description

Decodes the waveforms contained inside an event.

Synopsis

```
CAEN DGTZ ErrorCode CAENDGTZ API
CAEN_DGTZ_DecodeDPPWaveforms (int handle,
                               void *event,
                               void *waveforms
                               );
```

Arguments

Name	Description
handle	Device handler
*event	The pointer to the event
*waveforms	The pointer to the (preallocated) waveform list

Return Values

0: Success; negative numbers are error codes (see Return Codes).

SetDPPEventAggregation

Description

Sets event aggregation parameters.



Note: This function has to be used only after the record length parameter has been set (by the “Set” function of the **Set / GetRecordLength**).

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetDPPEventAggregation (int handle,
                                   int threshold,
                                   int maxsize
                                   );
```

Arguments

Name	Description
handle	Device handler
threshold	Specifies how many events to let accumulate in the board memory before they are rendered available for readout. A low number maximizes responsiveness, since data are read as soon as they are stored in memory, while a high number maximizes efficiency, since fewer transfers are made. Supplying 0 will let the library choose the most reasonable value depending on acquisition mode and other parameters.
maxsize	Specifies the maximum size in bytes of the event buffer on the PC side. This parameter might be useful in case the computer has very low RAM. Normally, though, it is safe to supply 0 as this parameter, so that the library will choose an appropriate value automatically.

Return Values

0: Success; negative numbers are error codes (see Return Codes).

Set / GetNumEventsPerAggregate

Description

Sets/Gets the number of events that each aggregate will contain.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetNumEventsPerAggregate (int handle,
                                     uint32_t numEvents,
                                     ...
                                     );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetNumEventsPerAggregate (int handle,
                                     uint32_t *numEvents,
                                     ...
                                     );
```

Arguments

Name	Description
handle	Device handler
numEvents	Number of events per aggregator.
channel	Optional parameter in the form of an int to specify the channel (required for DPP-PSD and DPP-CI, ignored by DPP-PHA).

Return Values

0: Success; negative numbers are error codes (see Return Codes).

Set / GetMaxNumAggregatesBLT

Description

Sets/Gets the maximum number of aggregates for each transfer.



Note: with DPP-PHA, DPP-PSD and DPP-CI, also the *maxsize* parameter of **SetDPPEventAggregation** can be used.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetMaxNumAggregatesBLT (int handle,
                                   uint32_t numAggr
                                   );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetMaxNumAggregatesBLT (int handle,
                                   uint32_t *numAggr
                                   );
```

Arguments

Name	Description
handle	Device handler
numAggr	Max number of aggregates per block transfer (BLT)

Return Values

0: Success; negative numbers are error codes (see Return Codes).

SetDPPParameters

Description

Sets DPP configuration parameters for DPP-PHA, DPP-PSD or DPP-CI.

Synopsis

```
CAEN DGTZ ErrorCode CAENDGTZ API
CAEN_DGTZ_SetDPPParameters (int handle,
                            uint32_t channelMask,
                            void *params
                            );

/* DPP parameter structure to be initialized and passed to CAEN DGTZ SetDPPParameters.
   To be used only for DPP-PHA */

typedef struct{
    int M          [MAX_V1724DPP_CHANNEL_SIZE]; //Signal Decay Time Constant
    int m          [MAX_V1724DPP_CHANNEL_SIZE]; //Trapezoid Flat Top
    int k          [MAX_V1724DPP_CHANNEL_SIZE]; //Trapezoid Rise Time
    int ftd       [MAX_V1724DPP_CHANNEL_SIZE]; //Flat Top Delay
    int a         [MAX_V1724DPP_CHANNEL_SIZE]; //Trigger Filter smoothing factor
    int b         [MAX_V1724DPP_CHANNEL_SIZE]; //Input Signal Rise time
    int thr       [MAX_V1724DPP_CHANNEL_SIZE]; //Trigger Threshold
    int nsbl      [MAX_V1724DPP_CHANNEL_SIZE]; //Number of Samples for Baseline Mean
    int nspk      [MAX_V1724DPP_CHANNEL_SIZE]; //Number of Samples for Peak Mean
    int pkho      [MAX_V1724DPP_CHANNEL_SIZE]; //Peak Hold Off
    int blho      [MAX_V1724DPP_CHANNEL_SIZE]; //Base Line Hold Off
    int otrej     [MAX_V1724DPP_CHANNEL_SIZE]; //
    int trgho     [MAX_V1724DPP_CHANNEL_SIZE]; //Trigger Hold Off
    int twwdt     [MAX_V1724DPP_CHANNEL_SIZE]; //
    int trgwin    [MAX_V1724DPP_CHANNEL_SIZE]; //
    int dgain     [MAX_V1724DPP_CHANNEL_SIZE]; //Digital Probe Gain
    float enf     [MAX_V1724DPP_CHANNEL_SIZE]; //Energy Normalization Factor
    int decimation [MAX_V1724DPP_CHANNEL_SIZE]; //Decimation of Input Signal
} CAEN_DGTZ_DPP_PHA_Params_t;

/* DPP parameter structure to be initialized and passed to CAEN_DGTZ_SetDPPParameters.
   To be used only for DPP-PSD */

typedef struct {
    int blthr;
    int bltmo;
    int thr          [MAX_V1720DPP_CHANNEL_SIZE];
    int selft       [MAX_V1720DPP_CHANNEL_SIZE];
    int csens       [MAX_V1720DPP_CHANNEL_SIZE];
    int sgate       [MAX_V1720DPP_CHANNEL_SIZE];
    int lgate       [MAX_V1720DPP_CHANNEL_SIZE];
    int pgate       [MAX_V1720DPP_CHANNEL_SIZE];
    int tvaw        [MAX_V1720DPP_CHANNEL_SIZE];
    int nsbl        [MAX_V1720DPP_CHANNEL_SIZE];
    CAEN_DGTZ_DPP_TriggerConfig_t trgc [MAX_V1720DPP_CHANNEL_SIZE]; // Ignored for x751
    CAEN_DGTZ_DPP_PUR_t purh; // Ignored for x751
    int purgap; // Ignored for x751
} CAEN_DGTZ_DPP_PSD_Params_t;

/* DPP parameter structure to be initialized and passed to CAEN DGTZ SetDPPParameters.
   To be used only for DPP-CI */

typedef struct {
    int blthr;
    int bltmo;
    int thr          [MAX_V1720DPP_CHANNEL_SIZE];
    int selft       [MAX_V1720DPP_CHANNEL_SIZE];
    int csens       [MAX_V1720DPP_CHANNEL_SIZE];
    int gate        [MAX_V1720DPP_CHANNEL_SIZE];
    int pgate       [MAX_V1720DPP_CHANNEL_SIZE];
    int tvaw        [MAX_V1720DPP_CHANNEL_SIZE];
    int nsbl        [MAX_V1720DPP_CHANNEL_SIZE];
    CAEN_DGTZ_DPP_TriggerConfig_t trgc [MAX_V1720DPP_CHANNEL_SIZE];
} CAEN_DGTZ_DPP_CI_Params_t;
```

Arguments

Name	Description
handle	Device handler
channelMask	A bit mask indicating the channels to apply the DPP parameters
*params	The pointer to a preallocated struct of type: CAEN_DGTZ_DPP_PHA_Params_t, in case of DPP-PHA CAEN_DGTZ_DPP_PSD_Params_t, in case of DPP-PSD CAEN_DGTZ_DPP_CI_Params_t, in case of DPP-CI

Return Values

0: Success; negative numbers are error codes (see Return Codes).

Set / GetDPPAcquisitionMode

Description

Sets/gets the DPP acquisition mode.

Synopsis

```
CAEN DGTZ ErrorCode CAENDGTZ API
CAEN_DGTZ_SetDPPAcquisitionMode (int handle,
                                  CAEN_DGTZ_DPP_AcqMode_t mode,
                                  CAEN_DGTZ_DPP_SaveParam_t param
                                  );

CAEN DGTZ ErrorCode CAENDGTZ API
CAEN_DGTZ_GetDPPAcquisitionMode (int handle,
                                  CAEN_DGTZ_DPP_AcqMode_t * mode,
                                  CAEN_DGTZ_DPP_SaveParam_t *param
                                  );

typedef enum
{
    CAEN_DGTZ_DPP_ACQ_MODE_Oscilloscope = 0L,
    CAEN_DGTZ_DPP_ACQ_MODE_List          = 1L,
    CAEN_DGTZ_DPP_ACQ_MODE_Mixed        = 2L,
} CAEN_DGTZ_DPP_AcqMode_t;

typedef enum
{
    CAEN_DGTZ_DPP_SAVE_PARAM_EnergyOnly   = 0L,
    CAEN_DGTZ_DPP_SAVE_PARAM_TimeOnly     = 1L,
    CAEN_DGTZ_DPP_SAVE_PARAM_EnergyAndTime = 2L,
    CAEN_DGTZ_DPP_SAVE_PARAM_ChargeAndTime = 4L,
    CAEN_DGTZ_DPP_SAVE_PARAM_None        = 3L,
} CAEN_DGTZ_DPP_SaveParam_t;
```

Arguments

Name	Description
handle	Device handler
mode/*mode	The DPP acquisition mode to set/get. CAEN_DGTZ_DPP_ACQ_MODE_Oscilloscope = 0L: enables the acquisition of the samples of the digitized waveforms. CAEN_DGTZ_DPP_ACQ_MODE_List = 1L: enables the acquisition of time stamps and energy values in case of DPP-PHA, or charge in case of DPP-CI and DPP-PSD. CAEN_DGTZ_DPP_ACQ_MODE_Mixed = 2L: enables the acquisition of both waveforms, energies or charges, and time stamps.
param/*param	The acquisition data to retrieve in acquisition

Return Values

0: Success; negative numbers are error codes (see Return Codes).

Set / GetDPPTriggerMode

Description

Sets/gets the DPP Trigger mode.



Note: to be used only with DPP-PSD and DPP-CI enabled firmware.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetDPPTriggerMode (int handle,
                             CAEN_DGTZ_DPP_TriggerMode t mode
                             );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetDPPTriggerMode (int handle,
                             CAEN_DGTZ_DPP_TriggerMode_t *mode
                             );

typedef enum
{
    CAEN_DGTZ_DPP_TriggerMode_Normal,
    CAEN_DGTZ_DPP_TriggerMode_Coincidence,
} CAEN_DGTZ_DPP_TriggerMode_t;
```

Arguments

Name	Description
handle	Device handler
mode/*mode	For SetDPPTriggerMode, it is the desired trigger mode which can be set: CAEN_DGTZ_DPP_TriggerMode_Normal CAEN_DGTZ_DPP_TriggerMode_Coincidence For GetDPPTriggerMode, it is the current trigger mode.

Return Values

0: Success; negative numbers are error codes (see Return Codes).

Set / GetDPP_PHA_VirtualProbe

Description

Set/gets the information about the output signal of the DPP-PHA acquisition mode.

Synopsis

```
CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetDPP_PHA_VirtualProbe (int handle,
                                   CAEN_DGTZ_DPP_VirtualProbe_t mode,
                                   CAEN_DGTZ_DPP_PHA_VirtualProbe1_t vp1,
                                   CAEN_DGTZ_DPP_PHA_VirtualProbe2_t vp2,
                                   CAEN_DGTZ_DPP_PHA_DigitalProbe_t dp
                                   );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetDPP_PHA_VirtualProbe (int handle,
                                   CAEN_DGTZ_DPP_VirtualProbe_t *mode,
                                   CAEN_DGTZ_DPP_PHA_VirtualProbe1_t *vp1,
                                   CAEN_DGTZ_DPP_PHA_VirtualProbe2_t *vp2,
                                   CAEN_DGTZ_DPP_PHA_DigitalProbe_t *dp
                                   );

typedef enum
{
    CAEN_DGTZ_DPP_VIRTUALPROBE_SINGLE = 0L,
    CAEN_DGTZ_DPP_VIRTUALPROBE_DUAL   = 1L,
} CAEN_DGTZ_DPP_VirtualProbe_t;

typedef enum
{
    CAEN_DGTZ_DPP_PHA_VIRTUALPROBE1_Input           = 0L,
    CAEN_DGTZ_DPP_PHA_VIRTUALPROBE1_Delta          = 1L,
    CAEN_DGTZ_DPP_PHA_VIRTUALPROBE1_Delta2         = 2L,
    CAEN_DGTZ_DPP_PHA_VIRTUALPROBE1_trapezoid      = 3L,
} CAEN_DGTZ_DPP_PHA_VirtualProbe1_t;

typedef enum
{
    CAEN_DGTZ_DPP_PHA_VIRTUALPROBE2_Input           = 0L,
    CAEN_DGTZ_DPP_PHA_VIRTUALPROBE2_S3              = 1L,
    CAEN_DGTZ_DPP_PHA_VIRTUALPROBE2_DigitalCombo    = 2L,
    CAEN_DGTZ_DPP_PHA_VIRTUALPROBE2_trapBaseline    = 3L,
    CAEN_DGTZ_DPP_PHA_VIRTUALPROBE2_None            = 4L,
} CAEN_DGTZ_DPP_PHA_VirtualProbe2_t;

typedef enum
{
    CAEN_DGTZ_DPP_PHA_DIGITAL_PROBE_trgKln          = 0L,
    CAEN_DGTZ_DPP_PHA_DIGITAL_PROBE_Armed           = 1L,
    CAEN_DGTZ_DPP_PHA_DIGITAL_PROBE_PkRun           = 2L,
    CAEN_DGTZ_DPP_PHA_DIGITAL_PROBE_PkAbort         = 3L,
    CAEN_DGTZ_DPP_PHA_DIGITAL_PROBE_Peaking         = 4L,
    CAEN_DGTZ_DPP_PHA_DIGITAL_PROBE_PkHoldOff       = 5L,
    CAEN_DGTZ_DPP_PHA_DIGITAL_PROBE_Flat            = 6L,
    CAEN_DGTZ_DPP_PHA_DIGITAL_PROBE_trgHoldOff      = 7L,
} CAEN_DGTZ_DPP_PHA_DigitalProbe_t;
```

Arguments

Name	Description
handle	Device handler
mode/*mode	The Virtual Probe mode to set/get.
vp1/*vp1	The Virtual Probe1 mode to set/get
vp2/*vp2	The Virtual Probe2 mode to set/get
dp/*dp	The Digital Probe mode to set/get

Return Values

0: Success; negative numbers are error codes (see Return Codes).

Set / GetDPP_PSD_VirtualProbe

Description

Sets/gets the information about the output signal of the DPP-PSD acquisition mode.

Synopsis

```
CAEN DGTZ ErrorCode CAENDGTZ API
CAEN_DGTZ_SetDPP_PSD_VirtualProbe (int handle,
                                   CAEN_DGTZ_DPP_VirtualProbe_t mode,
                                   CAEN_DGTZ_DPP_PSD_VirtualProbe_t vp,
                                   CAEN_DGTZ_DPP_PSD_DigitalProbe1_t dp1,
                                   CAEN_DGTZ_DPP_PSD_DigitalProbe2_t dp2
                                   );

CAEN DGTZ ErrorCode CAENDGTZ API
CAEN_DGTZ_GetDPP_PSD_VirtualProbe (int handle,
                                   CAEN_DGTZ_DPP_VirtualProbe_t *mode,
                                   CAEN_DGTZ_DPP_PSD_VirtualProbe_t *vp,
                                   CAEN_DGTZ_DPP_PSD_DigitalProbe1_t *dp1,
                                   CAEN_DGTZ_DPP_PSD_DigitalProbe2_t *dp2
                                   );

typedef enum
{
    CAEN_DGTZ_DPP_VIRTUALPROBE_SINGLE = 0L,
    CAEN_DGTZ_DPP_VIRTUALPROBE_DUAL   = 1L,
} CAEN_DGTZ_DPP_VirtualProbe_t;

typedef enum
{
    CAEN_DGTZ_DPP_PSD_VIRTUALPROBE_Baseline = 0L,
    CAEN_DGTZ_DPP_PSD_VIRTUALPROBE_Threshold = 1L,
} CAEN_DGTZ_DPP_PSD_VirtualProbe_t;

typedef enum
{
    /*
    /*****
    * WARNING WARNING WARNING WARNING WARNING WARNING WARNING *
    * The following values are valid for the following DPP-PSD *
    * Firmwares: *
    *   x720 Boards: AMC REL <= 131.5 *
    *   x751 Boards: AMC_REL <= 132.5 *
    * For newer firmwares, use the values marked with 'R4' in *
    * the name. *
    * WARNING WARNING WARNING WARNING WARNING WARNING WARNING *
    *****/
    /* x720 Digital Probes Types */
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_Armed           = 0L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_Trigger        = 1L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_ChargeReady    = 2L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_PileUp         = 3L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_BlOutSafeBand  = 4L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_BlTimeout      = 5L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_CoincidenceMet = 6L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_Tvaw          = 7L,

    /* x751 Digital Probes Types */
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_OverThr       = 8L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_GateShort      = 9L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_None          = 10L,

    /*
    /*****
    * WARNING WARNING WARNING WARNING WARNING WARNING WARNING *
    * The following values are valid for the following DPP-PSD *
    * Firmwares: *
    *   x720 Boards: AMC REL >= 131.6 *
    *   x751 Boards: AMC_REL >= 132.6 *
    * For older firmwares, use the values above. *
    * WARNING WARNING WARNING WARNING WARNING WARNING WARNING *
    *****/
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_R6_ExtTrg     = 11L, /*x720 only*/
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_R6_OverThr    = 12L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_R6_TrigOut    = 13L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_R6_CoincWin   = 14L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_R6_PileUp    = 15L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_R6_Coincidence = 16L,
    */
}
```

```

CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_R6_GateLong = 17L, /*x751 only*/
} CAEN_DGTZ_DPP_PSD_DigitalProbe1_t;

typedef enum
{
    /******
    * WARNING WARNING WARNING WARNING WARNING WARNING WARNING *
    * The following values are valid for the following DPP-PSD *
    * Firmwares: *
    *     x720 Boards: AMC_REL <= 131.5 *
    *     x751 Boards: AMC_REL <= 132.5 *
    * For newer firmwares, use the values marked with 'R4' in *
    * the name. *
    * WARNING WARNING WARNING WARNING WARNING WARNING WARNING *
    *****/

    /* x720 Digital Probes Types */
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_Armed = 0L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_Trigger = 1L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_ChargeReady = 2L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_PileUp = 3L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_BlOutSafeBand = 4L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_BlTimeout = 5L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_CoincidenceMet = 6L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_Tvaw = 7L,

    /* x751 Digital Probes Types */
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_GateShort = 8L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_GateLong = 9L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_None = 10L,

    /******
    * WARNING WARNING WARNING WARNING WARNING WARNING WARNING *
    * The following values are valid for the following DPP-PSD *
    * Firmwares: *
    *     x720 Boards: AMC_REL >= 131.6 *
    *     x751 Boards: AMC_REL >= 132.6 *
    * For older firmwares, use the values above. *
    * WARNING WARNING WARNING WARNING WARNING WARNING WARNING *
    *****/
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_R6_GateShort = 11L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_R6_OverThr = 12L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_R6_TrqVal = 13L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_R6_TrqHO = 14L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_R6_PileUp = 15L,
    CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_R6_Coincidence = 16L,
} CAEN_DGTZ_DPP_PSD_DigitalProbe2_t;

```

Arguments

Name	Description
handle	Device handler
mode/*mode	The Virtual Probe mode to set/get.
vp/*vp	The Virtual Probe to set/get. NOTE: ignored for x751; VirtualProbes are always Input and Baseline
dp1/*dp1	The Digital Probe1 to set/get
dp2/*dp2	The Digital Probe2 to set/get

Return Values

0: Success; negative numbers are error codes (see Return Codes).

Set / GetDPP_CI_VirtualProbe

Description

Sets/gets the information about the output signal of the DPP-CI acquisition mode.



Note: this function is supported only by DPP-CI firmware from release 3.4_130.16 on.

Synopsis

```

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_SetDPP_CI_VirtualProbe (int handle,
                                   CAEN_DGTZ_DPP_VirtualProbe_t mode,
                                   CAEN_DGTZ_DPP_CI_VirtualProbe_t vp,
                                   CAEN_DGTZ_DPP_CI_DigitalProbe1_t dp1,
                                   CAEN_DGTZ_DPP_CI_DigitalProbe2_t dp2
                                   );

CAEN_DGTZ_ErrorCode CAENDGTZ_API
CAEN_DGTZ_GetDPP_CI_VirtualProbe (int handle,
                                   CAEN_DGTZ_DPP_VirtualProbe_t *mode,
                                   CAEN_DGTZ_DPP_CI_VirtualProbe_t *vp,
                                   CAEN_DGTZ_DPP_CI_DigitalProbe1_t *dp1,
                                   CAEN_DGTZ_DPP_CI_DigitalProbe2_t *dp2
                                   );

typedef enum
{
    CAEN_DGTZ_DPP_VIRTUALPROBE_SINGLE = 0L,
    CAEN_DGTZ_DPP_VIRTUALPROBE_DUAL   = 1L,
} CAEN_DGTZ_DPP_VirtualProbe_t;

typedef enum
{
    CAEN_DGTZ_DPP_CI_VIRTUALPROBE_Baseline = 0L,
} CAEN_DGTZ_DPP_CI_VirtualProbe_t;

typedef enum
{
    /*
    * WARNING WARNING WARNING WARNING WARNING WARNING WARNING
    * The following values are valid for the following DPP-CI
    * Firmwares:
    *   x720 Boards: AMC_REL <= 130.20
    * For newer firmwares, use the values marked with 'R21' in
    * the name.
    * WARNING WARNING WARNING WARNING WARNING WARNING WARNING
    */
    CAEN_DGTZ_DPP_CI_DIGITALPROBE1_B1OutSafeBand = 0L,
    CAEN_DGTZ_DPP_CI_DIGITALPROBE1_B1Timeout    = 1L,
    CAEN_DGTZ_DPP_CI_DIGITALPROBE1_CoincidenceMet = 2L,
    CAEN_DGTZ_DPP_CI_DIGITALPROBE1_Tvaw         = 3L,

    /*
    * WARNING WARNING WARNING WARNING WARNING WARNING WARNING
    * The following values are valid for the following DPP-CI
    * Firmwares:
    *   x720 Boards: AMC_REL >= 130.22
    * For older firmwares, use the values above.
    * WARNING WARNING WARNING WARNING WARNING WARNING WARNING
    */
    CAEN_DGTZ_DPP_CI_DIGITALPROBE1_R21_ExtTrg      = 4L,
    CAEN_DGTZ_DPP_CI_DIGITALPROBE1_R21_OverThr     = 5L,
    CAEN_DGTZ_DPP_CI_DIGITALPROBE1_R21_TrigOut     = 6L,
    CAEN_DGTZ_DPP_CI_DIGITALPROBE1_R21_CoincWin    = 7L,
    CAEN_DGTZ_DPP_CI_DIGITALPROBE1_R21_Coincidence = 9L,
} CAEN_DGTZ_DPP_CI_DigitalProbe1_t;

typedef enum
{
    /*
    * WARNING WARNING WARNING WARNING WARNING WARNING WARNING
    * The following values are valid for the following DPP-CI
    * Firmwares:
    *   x720 Boards: AMC_REL <= 130.20
    * For newer firmwares, use the values marked with 'R21' in
    * the name.
    * WARNING WARNING WARNING WARNING WARNING WARNING WARNING
    */

```

```

*****/
CAEN DGTZ DPP CI DIGITALPROBE2 BlOutSafeBand = 0L,
CAEN DGTZ DPP CI DIGITALPROBE2 BlTimeout = 1L,
CAEN DGTZ DPP CI DIGITALPROBE2 CoincidenceMet = 2L,
CAEN_DGTZ_DPP_CI_DIGITALPROBE2_Tvaw = 3L,

/*****
* WARNING WARNING WARNING WARNING WARNING WARNING WARNING *
* The following values are valid for the following DPP-CI *
* Firmwares: *
* x720 Boards: AMC_REL >= 130.22 *
* For older firmwares, use the values above. *
* WARNING WARNING WARNING WARNING WARNING WARNING WARNING *
*****/
CAEN DGTZ DPP CI DIGITALPROBE2 R22 OverThr = 5L,
CAEN DGTZ DPP CI DIGITALPROBE2 R22 TrgVal = 6L,
CAEN_DGTZ_DPP_CI_DIGITALPROBE2_R22_TrgHO = 7L,
CAEN_DGTZ_DPP_CI_DIGITALPROBE2_R22_Coincidence = 9L,
} CAEN DGTZ DPP CI DigitalProbe2 t;

```

Arguments

Name	Description
handle	Device handler
mode/*mode	The Virtual Probe mode to set/get.
vp/*vp	The Virtual Probe to set/get
dp1/*dp1	The Digital Probe1 to set/get
dp2/*dp2	The Digital Probe2 to set/get

Return Values

0: Success; negative numbers are error codes (see Return Codes).

DPP code examples

The following example codes are intended to let the developer deal with the library C functions to build up a readout cycle when using DPP-PHA, DPP-CI and DPP-PSD firmware. **CAEN provides these examples also as source files and projects inside the CAENDigitizer full installation package, compliant to Visual Studio Professional 2010.**



Note: the DPP-CI example code works only with the DPP-CI firmware from release 3.4_130.16 on.

DPP-PHA EXAMPLE CODE

```

#include <CAENDigitizer.h>
#include <stdio.h>
#include <stdlib.h>

// #define INDIVIDUAL_TRIGGER_INPUTS
// NB: the following define MUST specify the ACTUAL max allowed number of boards
#define MAXNB 1

/* NB: the following define MUST specify the ACTUAL max allowed number of board's channels
it is needed for consistency inside the CAENDigitizer's functions used to allocate the
memory*/
#define MaxNChannels 8

#define MAXNBITS 14

/* include some useful functions from file Functions.c
you can find this file in the src directory */
#include "Functions.h"

/* #####
* Functions
* ##### */

/*-----*/
/*! \fn int ProgramDigitizer(int handle, DigitizerParams t Params,
CAEN_DGTZ_DPPParamsPHA_t DPPParams)

```

```

* \brief Program the registers of the digitizer with the relevant parameters
* \return 0=success; -1=error */
/*-----*/
int ProgramDigitizer(int handle, DigitizerParams t Params, CAEN_DGTZ_DPP_PHA_Params t
                    DPPParams)
{
    /* This function uses the CAENDigitizer API functions to perform the digitizer's initial
    configuration */
    int i, ret = 0;

    /* Reset the digitizer */
    ret |= CAEN_DGTZ_Reset(handle);

    if (ret) {
        printf("ERROR: can't reset the digitizer.\n");
        return -1;
    }

    ret |= CAEN_DGTZ_WriteRegister(handle, 0x8000, 0x01000114); /*Channel Control Reg (indiv
    trg, seq readout)*/

    /* Set the DPP acquisition mode.
    This setting affects the modes Mixed and List (see CAEN_DGTZ_DPP_AcqMode_t definition
    for details)*/
    CAEN_DGTZ_DPP_SAVE_PARAM_EnergyOnly;    Only energy (DPP-PHA) or charge (DPP-PSD/DPP-CI)
    is returned
    CAEN_DGTZ_DPP_SAVE_PARAM_TimeOnly;      Only time is returned
    CAEN_DGTZ_DPP_SAVE_PARAM_EnergyAndTime; Both energy/charge and time are returned
    CAEN_DGTZ_DPP_SAVE_PARAM_None;         No histogram data is returned */
    ret |= CAEN_DGTZ_SetDPPAcquisitionMode(handle, Params.AcqMode,
        CAEN_DGTZ_DPP_SAVE_PARAM_EnergyAndTime);

    // Set the digitizer acquisition mode (CAEN_DGTZ_SW_CONTROLLED or
    CAEN_DGTZ_S_IN_CONTROLLED)
    ret |= CAEN_DGTZ_SetAcquisitionMode(handle, CAEN_DGTZ_SW_CONTROLLED);

    // Set the number of samples for each waveform
    ret |= CAEN_DGTZ_SetRecordLength(handle, Params.RecordLength);

    // Set the I/O level (CAEN_DGTZ_IOLevel_NIM or CAEN_DGTZ_IOLevel_TTL)
    ret |= CAEN_DGTZ_SetIOLevel(handle, Params.IOlev);

    /* Set the digitizer's behaviour when an external trigger arrives:

    CAEN_DGTZ_TRGMODE_DISABLED: do nothing
    CAEN_DGTZ_TRGMODE_EXTOUT_ONLY: generate the Trigger Output signal
    CAEN_DGTZ_TRGMODE_ACQ_ONLY = generate acquisition trigger
    CAEN_DGTZ_TRGMODE_ACQ_AND_EXTOUT = generate both Trigger Output and acquisition trigger

    see CAENDigitizer user manual, chapter "Trigger configuration" for details */
    ret |= CAEN_DGTZ_SetExtTriggerInputMode(handle, CAEN_DGTZ_TRGMODE_ACQ_ONLY);

    // Set the enabled channels
    ret |= CAEN_DGTZ_SetChannelEnableMask(handle, Params.ChannelMask);

    // Set how many events to accumulate in the board memory before being available for
    readout
    ret |= CAEN_DGTZ_SetDPPEventAggregation(handle, Params.EventAggr, 0);

    /* Set the mode used to synchronize the acquisition between different boards.
    In this example the sync is disabled */
    ret |= CAEN_DGTZ_SetRunSynchronizationMode(handle, CAEN_DGTZ_RUN_SYNC_Disabled);

    // Set the DPP specific parameters for the channels in the given channelMask
    ret |= CAEN_DGTZ_SetDPPParameters(handle, Params.ChannelMask, &DPPParams);

    for(i=0; i<MaxNChannels; i++) {
        if (Params.ChannelMask & (1<<i)) {
            // Set a DC offset to the input signal to adapt it to digitizer's dynamic range
            ret |= CAEN_DGTZ_SetChannelDCOffset(handle, i, 0x8000);

            // Set the Pre-Trigger size (in samples)
            ret |= CAEN_DGTZ_SetDPPPreTriggerSize(handle, i, 80);

            // Set the polarity for the given channel (CAEN_DGTZ_PulsePolarityPositive or
            CAEN_DGTZ_PulsePolarityNegative)
            ret |= CAEN_DGTZ_SetChannelPulsePolarity(handle, i, Params.PulsePolarity);
        }
    }
}

```

```

/* Set the virtual probes settings
DPP-PHA can save:
2 analog waveforms:
    the first and the second can be specified with the VIRTUALPROBE 1 and 2 parameters

4 digital waveforms:
    the first is always the trigger
    the second is always the gate
    the third and fourth can be specified with the DIGITALPROBE 1 and 2 parameters

CAEN_DGTZ_DPP_VIRTUALPROBE_SINGLE -> Save only the Input Signal waveform
CAEN_DGTZ_DPP_VIRTUALPROBE_DUAL    -> Save also the waveform specified in
    VIRTUALPROBE

Virtual Probes 1 types:
CAEN_DGTZ_DPP_PHA_VIRTUALPROBE1_trapezoid
CAEN_DGTZ_DPP_PHA_VIRTUALPROBE1_Delta
CAEN_DGTZ_DPP_PHA_VIRTUALPROBE1_Delta2
CAEN_DGTZ_DPP_PHA_VIRTUALPROBE1_Input

Virtual Probes 2 types:
CAEN_DGTZ_DPP_PHA_VIRTUALPROBE2_Input
CAEN_DGTZ_DPP_PHA_VIRTUALPROBE2_S3
CAEN_DGTZ_DPP_PHA_VIRTUALPROBE2_DigitalCombo
CAEN_DGTZ_DPP_PHA_VIRTUALPROBE2_trapBaseline
CAEN_DGTZ_DPP_PHA_VIRTUALPROBE2_None

Digital Probes types:
CAEN_DGTZ_DPP_PHA_DIGITAL_PROBE_trgKln
CAEN_DGTZ_DPP_PHA_DIGITAL_PROBE_Armed
CAEN_DGTZ_DPP_PHA_DIGITAL_PROBE_PkRun
CAEN_DGTZ_DPP_PHA_DIGITAL_PROBE_PkAbort
CAEN_DGTZ_DPP_PHA_DIGITAL_PROBE_Peaking
CAEN_DGTZ_DPP_PHA_DIGITAL_PROBE_PkHoldOff
CAEN_DGTZ_DPP_PHA_DIGITAL_PROBE_Flat
CAEN_DGTZ_DPP_PHA_DIGITAL_PROBE_trgHoldOff */
ret |= CAEN_DGTZ_SetDPP_PHA_VirtualProbe(handle, CAEN_DGTZ_DPP_VIRTUALPROBE_DUAL,
    CAEN_DGTZ_DPP_PHA_VIRTUALPROBE1_Delta2,
    CAEN_DGTZ_DPP_PHA_VIRTUALPROBE2_Input,
    CAEN_DGTZ_DPP_PHA_DIGITAL_PROBE_trgHoldOff);

    if (ret) {
printf("Warning: errors found during the programming of the digitizer.\nSome settings may
    not be executed\n");
return ret;
    } else {
        return 0;
    }
}

/* ##### */
/* MAIN */
/* ##### */
int main(int argc, char *argv[])
{
    /* The following variable is the type returned from most of CAENDigitizer
    library functions and is used to check if there was an error in function
    execution. For example:
    ret = CAEN_DGTZ_some_function(some_args);
    if(ret) printf("Some error"); */
    CAEN_DGTZ_ErrorCode ret;

    /* Buffers to store the data. The memory must be allocated using the appropriate
    CAENDigitizer API functions (see below), so they must not be initialized here
    NB: you must use the right type for different DPP analysis (in this case PHA) */
    char *buffer = NULL; // readout buffer
    CAEN_DGTZ_DPP_PHA_Event_t *Events[MaxNChannels]; // events buffer
    CAEN_DGTZ_DPP_PHA_Waveforms_t *Waveform=NULL; // waveforms buffer

    /* The following variables will store the digitizer configuration parameters */
    CAEN_DGTZ_DPP_PHA_Params_t DPPParams[MAXNB];
    DigitizerParams_t Params[MAXNB];

    /* Arrays for data analysis */
    uint64_t PrevTime[MAXNB][MaxNChannels];
    uint64_t ExtendedTT[MAXNB][MaxNChannels];
    uint32_t *EHisto[MAXNB][MaxNChannels]; // Energy Histograms

```

```

int ECnt[MAXNB][MaxNChannels];
int TrgCnt[MAXNB][MaxNChannels];
int PurCnt[MAXNB][MaxNChannels];

/* The following variable will be used to get an handler for the digitizer. The
handler will be used for most of CAENDigitizer functions to identify the board */
int handle[MAXNB];

/* Other variables */
int i, b, ch, ev;
int Quit=0;
int AcqRun = 0;
uint32 t AllocatedSize, BufferSize;
int Nb=0;
int DoSaveWave[MAXNB][MaxNChannels];
int MajorNumber;
int BitMask = 0;
uint64_t CurrentTime, PrevRateTime, ElapsedTime;
uint32 t NumEvents[MaxNChannels];
CAEN DGTZ BoardInfo t BoardInfo;

memset(DoSaveWave, 0, MAXNB*MaxNChannels*sizeof(int));
for (i=0; i<MAXNBITS; i++)
    BitMask |= 1<<i; /* Create a bit mask based on number of bits of the board */

/* ***** */
/* Set Parameters */
/* ***** */
memset(&Params, 0, MAXNB*sizeof(DigitizerParams_t));
memset(&DPPParams, 0, MAXNB*sizeof(CAEN DGTZ DPP PHA Params t));
for(b=0; b<MAXNB; b++) {
    for(ch=0; ch<MaxNChannels; ch++)
        EHisto[b][ch] = NULL; //set all histograms pointers to NULL (we will allocate
                                them later)

    /******\
    * Communication Parameters *
    \*****/
    // Direct USB connection
    //Params[b].LinkType = CAEN_DGTZ_USB; // Link Type
    //Params[b].VMEBaseAddress = 0; // For direct USB connection, VMEBaseAddress must
        be 0

    // Direct optical connection
    //Params[b].LinkType = CAEN_DGTZ_PCI_OpticalLink; // Link Type
    //Params[b].VMEBaseAddress = 0; // For direct CONET connection, VMEBaseAddress
        must be 0

    // Optical connection to A2818 (or A3818) and access to the board with VME bus
    //Params[b].LinkType = CAEN_DGTZ_PCI_OpticalLink; // Link Type
        (CAEN_DGTZ_PCIE_OpticalLink for A3818)
    //Params[b].VMEBaseAddress = 0x32100000; // VME Base Address (only for VME bus
        access; must be 0 for direct connection (CONET or USB)

    // USB connection to V1718 bridge and access to the board with VME bus
    Params[b].LinkType = CAEN_DGTZ_USB; // Link Type
    Params[b].VMEBaseAddress = 0x32110000; // VME Base Address (only for VME bus
        access; must be 0 for direct connection (CONET or USB)

    Params[b].IOlev = CAEN_DGTZ_IOLevel_TTL;
    /******\
    * Acquisition parameters *
    \*****/
    Params[b].AcqMode = CAEN_DGTZ_DPP_ACQ_MODE_Mixed; //
        CAEN_DGTZ_DPP_ACQ_MODE_List or
        CAEN_DGTZ_DPP_ACQ_MODE_Oscilloscope
    Params[b].RecordLength = 400; // Num of samples of the
        waveforms (only for Oscilloscope mode)
    Params[b].ChannelMask = 0xFF; // Channel enable mask
    Params[b].EventAggr = 0; // number of events in
        one aggregate (0=automatic)
    Params[b].PulsePolarity = CAEN_DGTZ_PulsePolarityNegative; // Pulse Polarity (this
        parameter can be individual)

    /******\
    * DPP parameters *
    \*****/
    for(ch=0; ch<MaxNChannels; ch++) {

```

```

        DPPPParams[b].thr[ch] = 200; // Trigger Threshold
        DPPPParams[b].k[ch] = 1000; // Trapezoid Rise Time (N*10ns)
        DPPPParams[b].m[ch] = 500; // Trapezoid Flat Top (N*10ns)
        DPPPParams[b].M[ch] = 200; // Decay Time Constant (N*10ns) HACK-FPEP the
                                one expected from fitting algorithm?
        DPPPParams[b].ftd[ch] = 30; // Flat top delay (peaking time) (N*10ns) ??
        DPPPParams[b].a[ch] = 2; // Trigger Filter smoothing factor
        DPPPParams[b].b[ch] = 100; // Input Signal Rise time (N*10ns)
        DPPPParams[b].trgho[ch] = 600; // Trigger Hold Off
        DPPPParams[b].nsbl[ch] = 2; // 3 = bx10 = 64 samples
        DPPPParams[b].nspk[ch] = 2;
        DPPPParams[b].pkho[ch] = 770;
        DPPPParams[b].blho[ch] = 100;
        DPPPParams[b].enf[ch] = 1.0; // Energy Normalization Factor
        //DPPPParams[b].tsampl[ch] = 10;
        //DPPPParams[b].dgain[ch] = 1;
    }
}

/* ***** */
/* Open the digitizer and read board information */
/* ***** */
/* The following function is used to open the digitizer with the given connection
parameters and get the handler to it */
for(b=0; b<MAXNB; b++) {
    /* IMPORTANT: The following function identifies the different boards with a system
    which may change for different connection methods (USB, Conet, ecc). Refer to
    CAENDigitizer user manual for more info. Some examples below */

    /* The following is for b boards connected via b USB direct links.
    In this case you must set Params[b].LinkType = CAEN DGTZ USB and
    Params[b].VMEBaseAddress = 0 */
    //ret = CAEN_DGTZ_OpenDigitizer(Params[b].LinkType, b, 0,
    Params[b].VMEBaseAddress, &handle[b]);

    /* The following is for b boards connected via 1 opticalLink in dasy chain.
    In this case you must set Params[b].LinkType = CAEN DGTZ PCI OpticalLink and
    Params[b].VMEBaseAddress = 0 */
    //ret = CAEN_DGTZ_OpenDigitizer(Params[b].LinkType, 0, b,
    Params[b].VMEBaseAddress, &handle[b]);

    /* The following is for b boards connected to A2818 (or A3818) via opticalLink (or
    USB with A1718). In this case the boards are accessed throught VME bus, and you
    must specify the VME address of each board:
    Params[b].LinkType = CAEN_DGTZ_PCI_OpticalLink (CAEN_DGTZ_PCI_OpticalLink for
    A3818 or CAEN_DGTZ_USB for A1718)
    Params[0].VMEBaseAddress = <0XXXXXXXX> (address of first board)
    Params[1].VMEBaseAddress = <0YYYYYYYY> (address of second board)
    etc */
    ret = CAEN_DGTZ_OpenDigitizer(Params[b].LinkType, 0, 0,
    Params[b].VMEBaseAddress, &handle[b]);

    if (ret) {
        printf("Can't open digitizer\n");
        goto QuitProgram;
    }

    /* Once we have the handler to the digitizer, we use it to call the other functions
    */
    ret = CAEN_DGTZ_GetInfo(handle[b], &BoardInfo);
    if (ret) {
        printf("Can't read board info\n");
        goto QuitProgram;
    }
    printf("\nConnected to CAEN Digitizer Model %s, recognized as board %d\n",
    BoardInfo.ModelName, b);
    printf("ROC FPGA Release is %s\n", BoardInfo.ROC_FirmwareRel);
    printf("AMC FPGA Release is %s\n", BoardInfo.AMC_FirmwareRel);

    /* Check firmware revision (only DPP firmwares can be used with this Demo) */
    sscanf(BoardInfo.AMC_FirmwareRel, "%d", &MajorNumber);
    if (MajorNumber != 128) {
        printf("This digitizer has not a DPP-PHA firmware\n");
        goto QuitProgram;
    }
}
}

```



```

/* ***** */
/* Program the digitizer (see function ProgramDigitizer) */
/* ***** */
for(b=0; b<MAXNB; b++) {
    ret = ProgramDigitizer(handle[b], Params[b], DPPParams[b]);
    if (ret) {
        printf("Failed to program the digitizer\n");
        goto QuitProgram;
    }
}

/* WARNING: The mallocs MUST be done after the digitizer programming,
because the following functions needs to know the digitizer configuration
to allocate the right memory amount */
/* Allocate memory for the readout buffer */
ret = CAEN DGTZ MallocReadoutBuffer(handle[0], &buffer, &AllocatedSize);
/* Allocate memory for the events */
ret |= CAEN_DGTZ_MallocDPPEvents(handle[0], Events, &AllocatedSize);
/* Allocate memory for the waveforms */
ret |= CAEN DGTZ MallocDPPWaveforms(handle[0], &Waveform, &AllocatedSize);
if (ret) {
    printf("Can't allocate memory buffers\n");
    goto QuitProgram;
}

/* ***** */
/* Readout Loop */
/* ***** */
// Clear Histograms and counters
for(b=0; b<MAXNB; b++) {
    for(ch=0; ch<MaxNChannels; ch++) {
        EHisto[b][ch] = (uint32_t *)malloc( (1<<MAXNBITS)*sizeof(uint32_t) );
        memset(EHisto[b][ch], 0, (1<<MAXNBITS)*sizeof(uint32_t));
        TrgCnt[b][ch] = 0;
        ECnt[b][ch] = 0;
        PrevTime[b][ch] = 0;
        ExtendedTT[b][ch] = 0;
        PurCnt[b][ch] = 0;
    }
}
PrevRateTime = get time();
AcqRun = 0;
PrintInterface();
printf("Type a command: ");
while(!Quit) {

    // Check keyboard
    if(kbhit()) {
        char c;
        c = getch();
        if (c=='q') Quit = 1;
        if (c=='t')
            for(b=0; b<MAXNB; b++)
                CAEN_DGTZ_SendSWtrigger(handle[b]); /* Send a software trigger to each
                board */

        if (c=='h')
            for(b=0; b<MAXNB; b++)
                for(ch=0; ch<MaxNChannels; ch++)
                    if( ECnt[b][ch] != 0)
                        SaveHistogram("Histo", b, ch, EHisto[b][ch]); /* Save
                        Histograms to file for each board */

        if (c=='w')
            for(b=0; b<MAXNB; b++)
                for(ch=0; ch<MaxNChannels; ch++)
                    DoSaveWave[b][ch] = 1; /* save waveforms to file for each channel
                    for each board (at next trigger) */

        if (c=='r') {
            for(b=0; b<MAXNB; b++) {
                CAEN DGTZ SWStopAcquisition(handle[b]);
                printf("Restarted\n");
                CAEN_DGTZ_ClearData(handle[b]);
                CAEN_DGTZ_SWStartAcquisition(handle[b]);
            }
        }

        if (c=='s') {
            for(b=0; b<MAXNB; b++) {
                // Start Acquisition

```

```

        // NB: the acquisition for each board starts when the following line is
        //       executed
        // so in general the acquisition does NOT starts synchronously for
        //       different boards
        CAEN_DGTZ_SWStartAcquisition(handle[b]);
        printf("Acquisition Started for Board %d\n", b);
    }
    AcqRun = 1;
}
if (c=='S') {
    for(b=0; b<MAXNB; b++) {
        // Stop Acquisition
        CAEN_DGTZ_SWStopAcquisition(handle[b]);
        printf("Acquisition Stopped for Board %d\n", b);
    }
    AcqRun = 0;
}
}
if (!AcqRun) {
    Sleep(10);
    continue;
}

/* Calculate throughput and trigger rate (every second) */
CurrentTime = get time();
ElapsedTime = CurrentTime - PrevRateTime; /* milliseconds */
if (ElapsedTime > 1000) {
    system(CLEARSCR);
    PrintInterface();
    printf("Readout Rate=%.2f MB\n", (float)Nb/((float)ElapsedTime*1048.576f));
    for(b=0; b<MAXNB; b++) {
        printf("\nBoard %d:\n",b);
        for(i=0; i<MaxNChannels; i++) {
            if (TrgCnt[b][i]>0)
                printf("\tCh %d:\tTrgRate=%.2f KHz\tPileUpRate=%.2f%%\n", i,
                    (float)TrgCnt[b][i]/(float)ElapsedTime,
                    (float)PurCnt[b][i]*100/(float)TrgCnt[b][i]);
            else
                printf("\tCh %d:\tNo Data\n", i);
            TrgCnt[b][i]=0;
            PurCnt[b][i]=0;
        }
        Nb = 0;
        PrevRateTime = CurrentTime;
        printf("\n\n");
    }
}

/* Read data from the boards */
for(b=0; b<MAXNB; b++) {
    /* Read data from the board */
    ret = CAEN_DGTZ_ReadData(handle[b], CAEN_DGTZ_SLAVE_TERMINATED_READOUT_MBLT,
        buffer, &BufferSize);
    if (ret) {
        printf("Readout Error\n");
        goto QuitProgram;
    }
    if (BufferSize == 0)
        continue;

    Nb += BufferSize;
    //ret = DataConsistencyCheck((uint32_t *)buffer, BufferSize/4);
    ret |= CAEN_DGTZ_GetDPPEvents(handle[b], buffer, BufferSize, Events,
        NumEvents);
    if (ret) {
        printf("Data Error: %d\n", ret);
        goto QuitProgram;
    }
}

/* Analyze data */
//for(b=0; b<MAXNB; b++) printf("%d now: %d\n", b, Params[b].ChannelMask);
for(ch=0; ch<MaxNChannels; ch++) {
    if (!(Params[b].ChannelMask & (1<<ch)))
        continue;

    /* Update Histograms */
    for(ev=0; ev<NumEvents[ch]; ev++) {
        TrgCnt[b][ch]++;
    }
}

```

```

        /* Time Tag */
        if (Events[ch][ev].TimeTag < PrevTime[b][ch])
            ExtendedTT[b][ch]++;
        PrevTime[b][ch] = Events[ch][ev].TimeTag;
        /* Energy */
        if (Events[ch][ev].Energy > 0) {
            // Fill the histograms
            EHisto[b][ch][(Events[ch][ev].Energy)&BitMask]++;
            ECnt[b][ch]++;
        } else { /* PileUp */
            PurCnt[b][ch]++;
        }
        /* Get Waveforms (only from 1st event in the buffer) */
        if ((Params[b].AcqMode != CAEN_DGTZ_DPP_ACQ_MODE_List) &&
            DoSaveWave[b][ch] && (ev == 0)) {
            int size;
            int16_t *WaveLine;
            uint8_t *DigitalWaveLine;
            CAEN_DGTZ_DecodeDPPWaveforms(handle[b], &Events[ch][ev], Waveform);

            // Use waveform data here...
            size = (int)(Waveform->Ns); // Number of samples
            WaveLine = Waveform->Trace1; // First trace (VIRTUALPROBE1 set with
                CAEN_DGTZ_SetDPP_PSD_VirtualProbe)
            SaveWaveform(b, ch, 1, size, WaveLine);

            WaveLine = Waveform->Trace2; // Second Trace (if single trace mode,
                it is a sequence of zeroes)
            SaveWaveform(b, ch, 2, size, WaveLine);

            DigitalWaveLine = Waveform->DTrace1; // First Digital Trace
                (DIGITALPROBE1 set with CAEN_DGTZ_SetDPP_PSD_VirtualProbe)
            SaveDigitalProbe(b, ch, 1, size, DigitalWaveLine);

            DigitalWaveLine = Waveform->DTrace2; // Second Digital Trace (for
                DPP-PHA it is ALWAYS Trigger)
            SaveDigitalProbe(b, ch, 2, size, DigitalWaveLine);
            DoSaveWave[b][ch] = 0;
            printf("Waveforms saved to
                'Waveform_<board>_<channel>_<trace>.txt'\n");
        } // loop to save waves
    } // loop on events
} // loop on channels
} // loop on boards
} // End of readout loop

QuitProgram:
/* stop the acquisition, close the device and free the buffers */
for (b=0; b<MAXNB; b++) {
    CAEN_DGTZ_SWStopAcquisition(handle[b]);
    CAEN_DGTZ_CloseDigitizer(handle[b]);
    for (ch=0; ch<MaxNChannels; ch++)
        if (EHisto[b][ch] != NULL)
            free(EHisto[b][ch]);
}
CAEN_DGTZ_FreeReadoutBuffer(&buffer);
CAEN_DGTZ_FreeDPPEvents(handle[0], Events);
CAEN_DGTZ_FreeDPPWaveforms(handle[0], Waveform);
return ret;
}

```

DPP-CI EXAMPLE CODE

```

#include <CAENDigitizer.h>
#include <stdio.h>
#include <stdlib.h>

// #define MANUAL_BUFFER_SETTING 0
// The following define must be set to the actual number of connected boards
#define MAXNB 1

// NB: the following define MUST specify the ACTUAL max allowed number of board's channels
// it is needed for consistency inside the CAENDigitizer's functions used to
// allocate the memory
#define MaxNChannels 8

#define MAXNBITS 12

/* include some useful functions from file Functions.c
you can find this file in the src directory */
#include "Functions.h"

/* #####
* Functions
* #####
*/

/* ----- */
/*! \fn int ProgramDigitizer(int handle, DigitizerParams_t Params,
CAEN_DGTZ_DPPParamsPHA_t DPPParams)
* \brief Program the registers of the digitizer with the relevant parameters
* \return 0=success; -1=error */
/* ----- */
int ProgramDigitizer(int handle, DigitizerParams_t Params,
CAEN_DGTZ_DPP_CI_Params_t DPPParams)
{
    /* This function uses the CAENDigitizer API functions to perform the digitizer's
initial configuration */

    int i, ret = 0;

    /* Reset the digitizer */
    ret |= CAEN_DGTZ_Reset(handle);

    if (ret) {
        printf("ERROR: can't reset the digitizer.\n");
        return -1;
    }

    /* Set the DPP acquisition mode
This setting affects the modes Mixed and List (see CAEN DGTZ DPP AcqMode t definition
for details)
CAEN_DGTZ_DPP_SAVE_PARAM_EnergyOnly Only energy (DPP-PHA) or charge (DPP-
PSD/DPP-CI v2) is returned
CAEN_DGTZ_DPP_SAVE_PARAM_TimeOnly Only time is returned
CAEN_DGTZ_DPP_SAVE_PARAM_EnergyAndTime Both energy/charge and time are returned
CAEN_DGTZ_DPP_SAVE_PARAM_None No histogram data is returned */
    ret |= CAEN_DGTZ_SetDPPAcquisitionMode(handle, Params.AcqMode,
CAEN_DGTZ_DPP_SAVE_PARAM_EnergyAndTime);

    /* Set the digitizer acquisition mode (CAEN DGTZ SW CONTROLLED or
CAEN DGTZ S IN CONTROLLED)
ret |= CAEN_DGTZ_SetAcquisitionMode(handle, CAEN_DGTZ_SW_CONTROLLED);

    /* Set the number of samples for each waveform
ret |= CAEN_DGTZ_SetRecordLength(handle, Params.RecordLength);

    /* Set the I/O level (CAEN DGTZ IOLevel NIM or CAEN DGTZ IOLevel TTL)
ret |= CAEN_DGTZ_SetIOLevel(handle, Params.IOlev);

    /* Set the digitizer's behaviour when an external trigger arrives:

CAEN_DGTZ_TRGMODE_DISABLED: do nothing
CAEN_DGTZ_TRGMODE_EXTOUT_ONLY: generate the Trigger Output signal
CAEN_DGTZ_TRGMODE_ACQ_ONLY = generate acquisition trigger
CAEN_DGTZ_TRGMODE_ACQ_AND_EXTOUT = generate both Trigger Output and acquisition trigger

see CAENDigitizer user manual, chapter "Trigger configuration" for details */
    ret |= CAEN_DGTZ_SetExtTriggerInputMode(handle, CAEN_DGTZ_TRGMODE_ACQ_ONLY);

```

```

// Set the enabled channels
ret |= CAEN DGTZ SetChannelEnableMask(handle, Params.ChannelMask);

// Set how many events to accumulate in the board memory before being available for
// readout
ret |= CAEN_DGTZ_SetDPPEventAggregation(handle, Params.EventAggr, 0);

/* Set the mode used to synchronize the acquisition between different boards.
In this example the sync is disabled */
ret |= CAEN DGTZ SetRunSynchronizationMode(handle, CAEN DGTZ RUN SYNC Disabled);

// Set the DPP specific parameters for the channels in the given channelMask
ret |= CAEN DGTZ SetDPPParameters(handle, Params.ChannelMask, &DPPParams);

for(i=0; i<MaxNChannels; i++) {
    if (Params.ChannelMask & (1<<i)) {
        // Set a DC offset to the input signal to adapt it to digitizer's dynamic range
        ret |= CAEN_DGTZ_SetChannelDCOffset(handle, i, 0x8000);

        // Set the Pre-Trigger size (in samples)
        ret |= CAEN DGTZ SetDPPPreTriggerSize(handle, i, 80);

        // Set the polarity for the given channel (CAEN_DGTZ_PulsePolarityPositive or
        // CAEN_DGTZ_PulsePolarityNegative)
        ret |= CAEN DGTZ SetChannelPulsePolarity(handle, i, Params.PulsePolarity);
    }
}

/* Set the virtual probes settings
DPP-CI can save:
2 analog waveforms:
    Analog Trace 1: it is always the input signal;
    Analog Trace 2: it can be specified with the VIRTUALPROBE parameter
4 digital waveforms:
    Digital Trace 1: it is always the trigger
    Digital Trace 2: it is always the gate
    Digital Trace 2/3: they can be specified with the DIGITALPROBE 1 and 2 parameters

CAEN_DGTZ_DPP_VIRTUALPROBE_SINGLE    -> Save only the Input Signal waveform
CAEN_DGTZ_DPP_VIRTUALPROBE_DUAL      -> Save also the Trace specified in VIRTUALPROBE
                                        (interleaved)

Probes types for Analog Trace 2:
    CAEN_DGTZ_DPP_CI_VIRTUALPROBE_Baseline

Probes types for Digital Trace 3:
    ### Virtual Probes only for FW >= 130.21 ###
    CAEN DGTZ DPP CI DIGITALPROBE1 R21 ExtTrg
    CAEN DGTZ DPP CI DIGITALPROBE1 R21 OverThr
    CAEN_DGTZ_DPP_CI_DIGITALPROBE1_R21_TrgOut
    CAEN_DGTZ_DPP_CI_DIGITALPROBE1_R21_CoincWin
    CAEN_DGTZ_DPP_CI_DIGITALPROBE1_R21_Coincidence
    ### Virtual Probes only for FW <= 130.20 ###
    CAEN DGTZ DPP CI DIGITALPROBE1 B1OutSafeBand
    CAEN_DGTZ_DPP_CI_DIGITALPROBE1_B1Timeout
    CAEN_DGTZ_DPP_CI_DIGITALPROBE1_CoincidenceMet
    CAEN DGTZ DPP CI DIGITALPROBE1 Tvaw

Probes types for Digital Trace 4:
    ### Virtual Probes only for FW >= 130.21 ###
    CAEN_DGTZ_DPP_CI_DIGITALPROBE2_R22_OverThr
    CAEN_DGTZ_DPP_CI_DIGITALPROBE2_R22_TrgVal
    CAEN DGTZ DPP CI DIGITALPROBE2 R22 TrgHO
    CAEN DGTZ DPP CI DIGITALPROBE2 R22 Coincidence
    ### Virtual Probes only for FW <= 130.20 ###
    CAEN DGTZ DPP CI DIGITALPROBE2 R22 OverThr
    CAEN_DGTZ_DPP_CI_DIGITALPROBE2_R22_TrgVal
    CAEN_DGTZ_DPP_CI_DIGITALPROBE2_R22_TrgHO
    CAEN DGTZ DPP CI DIGITALPROBE2 R22 Coincidence */
ret |= CAEN DGTZ SetDPP CI VirtualProbe(handle, CAEN DGTZ DPP VIRTUALPROBE SINGLE,
    CAEN DGTZ DPP CI VIRTUALPROBE Baseline,
    CAEN_DGTZ_DPP_CI_DIGITALPROBE1_B1OutSafeBand,
    CAEN_DGTZ_DPP_CI_DIGITALPROBE2_Tvaw);

if (ret) {
    printf("Warning: errors found during the programming of the digitizer.\nSome
    settings may not be executed\n");
    return ret;
} else {

```

```

        return 0;
    }
}

/* ##### */
/* MAIN */
/* ##### */
int main(int argc, char *argv[])
{
    /* The following variable is the type returned from most of CAENDigitizer
    library functions and is used to check if there was an error in function
    execution. For example:
    ret = CAEN DGTZ some function(some args);
    if(ret) printf("Some error"); */
    CAEN DGTZ ErrorCode ret;

    /* Buffers to store the data. The memory must be allocated using the appropriate
    CAENDigitizer API functions (see below), so they must not be initialized here
    NB: you must use the right type for different DPP analysis (in this case CI) */
    char *buffer = NULL; // readout buffer
    CAEN DGTZ DPP CI Event t *Events[MaxNChannels]; // events buffer
    CAEN_DGTZ_DPP_CI_Waveforms_t *Waveform=NULL; // waveforms buffer

    /* The following variables will store the digitizer configuration parameters */
    CAEN DGTZ DPP CI Params t DPPParams[MAXNB];
    DigitizerParams t Params[MAXNB];

    /* Arrays for data analysis */
    uint64_t PrevTime[MAXNB][MaxNChannels];
    uint64_t ExtendedTT[MAXNB][MaxNChannels];
    uint32_t *EHisto[MAXNB][MaxNChannels]; // Energy Histograms
    int ECnt[MAXNB][MaxNChannels];
    int TrgCnt[MAXNB][MaxNChannels];

    /* The following variable will be used to get an handler for the digitizer. The
    handler will be used for most of CAENDigitizer functions to identify the board */
    int handle[MAXNB];

    /* Other variables */
    int i, b, ch, ev;
    int Quit=0;
    int AcqRun = 0;
    uint32_t AllocatedSize, BufferSize;
    int Nb=0;
    int DoSaveWave[MAXNB][MaxNChannels];
    int MajorNumber;
    int BitMask = 0;
    uint64_t CurrentTime, PrevRateTime, ElapsedTime;
    uint32_t NumEvents[MaxNChannels];
    CAEN_DGTZ_BoardInfo_t BoardInfo;

    memset(DoSaveWave, 0, MAXNB*MaxNChannels*sizeof(int));
    for (i=0; i<MAXNBITS; i++)
        BitMask |= 1<<i; /* Create a bit mask based on number of bits of the board */

    /* ##### */
    /* Set Parameters */
    /* ##### */
    memset(&Params, 0, MAXNB*sizeof(DigitizerParams t));
    memset(&DPPParams, 0, MAXNB*sizeof(CAEN DGTZ DPP CI Params t));
    for(b=0; b<MAXNB; b++) {
        for(ch=0; ch<MaxNChannels; ch++)
            EHisto[b][ch] = NULL; //set all histograms pointers to NULL (we will allocate
            them later)

        /*#####\
        * Communication Parameters *
        \#####/
        // Direct USB connection
        Params[b].LinkType = CAEN DGTZ USB; // Link Type
        Params[b].VMEBaseAddress = 0; // For direct USB connection, VMEBaseAddress must be
        0

        // Direct optical connection
        //Params[b].LinkType = CAEN DGTZ PCI OpticalLink; // Link Type
        //Params[b].VMEBaseAddress = 0; // For direct CONET connection, VMEBaseAddress
        must be 0
    }
}

```

```

// Optical connection to A2818 (or A3818) and access to the board with VME bus
//Params[b].LinkType = CAEN DGTZ PCI OpticalLink; // Link Type
(CAEN DGTZ PCIE OpticalLink for A3818)
//Params[b].VMEBaseAddress = 0x32100000; // VME Base Address (only for VME bus
access; must be 0 for direct connection (CONET or USB)

// USB connection to V1718 bridge and access to the board with VME bus
//Params[b].LinkType = CAEN DGTZ USB; // Link Type (CAEN DGTZ PCIE OpticalLink for
A3818)
//Params[b].VMEBaseAddress = 0x11110000; // VME Base Address (only for VME bus
access; must be 0 for direct connection (CONET or USB)

Params[b].IOlev = CAEN DGTZ IOLevel TTL;
/*****\
* Acquisition parameters *
\*****/
Params[b].AcqMode = CAEN_DGTZ_DPP_ACQ_MODE_Mixed; //
CAEN_DGTZ_DPP_ACQ_MODE_List or
CAEN DGTZ DPP ACQ MODE Oscilloscope

Params[b].RecordLength = 300; // Num of samples of the
waveforms (only for Oscilloscope mode)
Params[b].ChannelMask = 0xF; // Channel enable mask
Params[b].EventAggr = 0; // number of events in
one aggregate (0=automatic)
Params[b].PulsePolarity = CAEN DGTZ PulsePolarityNegative; // Pulse Polarity (this
parameter can be individual)

/*****\
* DPP parameters *
\*****/
for(ch=0; ch<MaxNChannels; ch++) {
DPPParams[b].thr[ch] = 100; // Trigger Threshold
/* The following parameter is used to specifiy the number of samples for the
baseline averaging:

0 -> 8samp
1 -> 16samp
2 -> 32samp
3 -> 64samp */
DPPParams[b].nsbl[ch] = 2;
DPPParams[b].gate[ch] = 200; // Gate Width (N*4ns)
DPPParams[b].pgate[ch] = 25; // Pre Gate Width (N*4ns)
/* Self Trigger Mode:
0 -> Disabled
1 -> Enabled */
DPPParams[b].selft[ch] = 1;
/* Trigger configuration:
CAEN_DGTZ_DPP_TriggerConfig_Peak -> trigger on peak
CAEN DGTZ DPP TriggerConfig Threshold -> trigger on threshold */
DPPParams[b].trgc[ch] = CAEN DGTZ DPP TriggerConfig Peak;
/* Trigger Validation Acquisition Window */
DPPParams[b].tvaw[ch] = 50;
}
DPPParams[b].blthr = 3; // Baseline Threshold
DPPParams[b].bltmo = 100; // Baseline Timeout
DPPParams[b].trgho = 0; // Trigger Holdoff
}

/* *****/
/* Open the digitizer and read board information */
/* *****/
/* The following function is used to open the digitizer with the given connection
parameters and get the handler to it */
for(b=0; b<MAXNB; b++) {
/* IMPORTANT: The following function identifies the different boards with a system
which may change
for different connection methods (USB, Conet, ecc). Refer to CAENDigitizer user
manual for more info.
Some examples below */

/* The following is for b boards connected via b USB direct links
in this case you must set Params[b].LinkType = CAEN DGTZ USB and
Params[b].VMEBaseAddress = 0 */
ret = CAEN_DGTZ_OpenDigitizer(Params[b].LinkType, b, 0, Params[b].VMEBaseAddress,
&handle[b]);

/* The following is for b boards connected via 1 opticalLink in dasy chain
in this case you must set Params[b].LinkType = CAEN_DGTZ_PCI_OpticalLink and
Params[b].VMEBaseAddress = 0 */

```

```

//ret = CAEN_DGTZ_OpenDigitizer(Params[b].LinkType, 0, b, Params[b].VMEBaseAddress,
                                &handle[b]);

/* The following is for b boards connected to A2818 (or A3818) via opticalLink (or
   USB with A1718)
   in this case the boards are accessed through VME bus, and you must specify the VME
   address of each board:
Params[b].LinkType = CAEN DGTZ PCI OpticalLink (CAEN DGTZ PCIE OpticalLink for
A3818 or CAEN DGTZ USB for A1718)
Params[0].VMEBaseAddress = <0XXXXXXXX> (address of first board)
Params[1].VMEBaseAddress = <0YYYYYYYY> (address of second board)
etc */
//ret = CAEN_DGTZ_OpenDigitizer(Params[b].LinkType, 0, 0, Params[b].VMEBaseAddress,
                                &handle[b]);

if (ret) {
    printf("Can't open digitizer\n");
    goto QuitProgram;
}

/* Once we have the handler to the digitizer, we use it to call the other functions
   */
ret = CAEN_DGTZ_GetInfo(handle[b], &BoardInfo);
if (ret) {
    printf("Can't read board info\n");
    goto QuitProgram;
}
printf("\nConnected to CAEN Digitizer Model %s, recognized as board %d\n",
        BoardInfo.ModelName, b);
printf("ROC FPGA Release is %s\n", BoardInfo.ROC FirmwareRel);
printf("AMC FPGA Release is %s\n", BoardInfo.AMC FirmwareRel);

/* Check firmware revision (only DPP firmware can be used with this Demo) */
sscanf(BoardInfo.AMC FirmwareRel, "%d", &MajorNumber);
if (MajorNumber != 130) {
    printf("This digitizer has not a DPP-CI firmware\n");
    goto QuitProgram;
}
}

/* ***** */
/* Program the digitizer (see function ProgramDigitizer) */
/* ***** */
for(b=0; b<MAXNB; b++) {
    ret = ProgramDigitizer(handle[b], Params[b], DPPParams[b]);
    if (ret) {
        printf("Failed to program the digitizer\n");
        goto QuitProgram;
    }
}

/* WARNING: The mallocs MUST be done after the digitizer programming,
because the following functions needs to know the digitizer configuration
to allocate the right memory amount */
/* Allocate memory for the readout buffer */
ret = CAEN_DGTZ_MallocReadoutBuffer(handle[0], &buffer, &AllocatedSize);
/* Allocate memory for the events */
ret |= CAEN_DGTZ_MallocDPPEvents(handle[0], Events, &AllocatedSize);
/* Allocate memory for the waveforms */
ret |= CAEN_DGTZ_MallocDPPWaveforms(handle[0], &Waveform, &AllocatedSize);
if (ret) {
    printf("Can't allocate memory buffers\n");
    goto QuitProgram;
}

/* ***** */
/* Readout Loop */
/* ***** */
// Clear Histograms and counters
for(b=0; b<MAXNB; b++) {
    for(ch=0; ch<MaxNChannels; ch++) {
        EHisto[b][ch] = (uint32_t *)malloc( (1<<MAXNBITS)*sizeof(uint32_t) );
        memset(EHisto[b][ch], 0, (1<<MAXNBITS)*sizeof(uint32_t));
        TrgCnt[b][ch] = 0;
        ECnt[b][ch] = 0;
        PrevTime[b][ch] = 0;
        ExtendedTT[b][ch] = 0;
    }
}

```



```

    }
    Nb = 0;
    PrevRateTime = CurrentTime;
    printf("\n\n");
}

/* Read data from the boards */
for(b=0; b<MAXNB; b++) {
    /* Read data from the board */
    ret = CAEN_DGTZ_ReadData(handle[b], CAEN_DGTZ_SLAVE_TERMINATED_READOUT_MBLT,
        buffer, &BufferSize);

    if (ret) {
        printf("Readout Error\n");
        goto QuitProgram;
    }
    if (BufferSize == 0)
        continue;

    Nb += BufferSize;
    //ret = DataConsistencyCheck((uint32 t *)buffer, BufferSize/4);
    ret |= CAEN_DGTZ_GetDPPEvents(handle[b], buffer, BufferSize, Events,
        NumEvents);

    if (ret) {
        printf("Data Error: %d\n", ret);
        goto QuitProgram;
    }

    /* Analyze data */
    for(ch=0; ch<MaxNChannels; ch++) {
        if (!(Params[b].ChannelMask & (1<<ch)))
            continue;

        /* Update Histograms */
        for(ev=0; ev<NumEvents[ch]; ev++) {
            TrgCnt[b][ch]++;
            /* Time Tag */
            if (Events[ch][ev].TimeTag < PrevTime[b][ch])
                ExtendedTT[b][ch]++;
            PrevTime[b][ch] = Events[ch][ev].TimeTag;
            /* Energy */
            if (Events[ch][ev].Charge > 0) {
                // Fill the histograms
                EHisto[b][ch][(Events[ch][ev].Charge) & BitMask]++;
                ECnt[b][ch]++;
            }

            /* Get Waveforms (only from 1st event in the buffer) */
            if ((Params[b].AcqMode != CAEN_DGTZ_DPP_ACQ_MODE_List) &&
                DoSaveWave[b][ch] && (ev == 0)) {
                int size;
                int16_t *WaveLine;
                uint8_t *DigitalWaveLine;
                CAEN_DGTZ_DecodeDPPWaveforms(handle[b], &Events[ch][ev], Waveform);

                // Use waveform data here...
                size = (int)(Waveform->Ns); // Number of samples
                WaveLine = Waveform->Trace1; // First trace (for DPP-CI it is
                    ALWAYS the Input Signal)
                SaveWaveform(b, ch, 1, size, WaveLine);

                WaveLine = Waveform->Trace2; // Second Trace (if single trace mode,
                    it is a sequence of zeroes)
                SaveWaveform(b, ch, 2, size, WaveLine);
                DoSaveWave[b][ch] = 0;

                DigitalWaveLine = Waveform->DTrace1; // First Digital Trace
                    (Trigger)
                SaveDigitalProbe(b, ch, 1, size, DigitalWaveLine);
                DoSaveWave[b][ch] = 0;

                DigitalWaveLine = Waveform->DTrace2; // Second Digital Trace (Gate)
                SaveDigitalProbe(b, ch, 2, size, DigitalWaveLine);
                DoSaveWave[b][ch] = 0;

                DigitalWaveLine = Waveform->DTrace3; // Third Digital Trace
                    (DIGITALPROBE1 set with CAEN_DGTZ_SetDPP_PSD_VirtualProbe)
                SaveDigitalProbe(b, ch, 3, size, DigitalWaveLine);
            }
        }
    }
}

```

```

        DoSaveWave[b][ch] = 0;

        DigitalWaveLine = Waveform->DTrace4; // Fourth Digital Trace
        (DIGITALPROBE2 set with CAEN DGTZ SetDPP PSD VirtualProbe)
        SaveDigitalProbe(b, ch, 4, size, DigitalWaveLine);
        DoSaveWave[b][ch] = 0;
        printf("Waveforms                                saved                to
               'Waveform <board> <channel> <trace>.txt'\n");
        } // loop to save waves
    } // loop on events
} // loop on channels
} // loop on boards
} // End of readout loop

QuitProgram:
/* stop the acquisition, close the device and free the buffers */
for(b=0; b<MAXNB; b++) {
    CAEN DGTZ SWStopAcquisition(handle[b]);
    CAEN DGTZ CloseDigitizer(handle[b]);
    for (ch=0; ch<MaxNChannels; ch++)
        if (EHisto[b][ch] != NULL)
            free(EHisto[b][ch]);
}
CAEN DGTZ FreeReadoutBuffer(&buffer);
CAEN DGTZ FreeDPPEvents(handle[0], Events);
CAEN DGTZ FreeDPPWaveforms(handle[0], Waveform);
return ret;
}

```

DPP-PSD EXAMPLE CODE

```

#include <CAENDigitizer.h>
#include <stdio.h>
#include <stdlib.h>

// #define MANUAL BUFFER SETTING 0
// The following define must be set to the actual number of connected boards
#define MAXNB 1
// NB: the following define MUST specify the ACTUAL max allowed number of board's channels
// it is needed for consistency inside the CAENDigitizer's functions used to allocate the
// memory
#define MaxNChannels 8

#define MAXNBITS 12

/* include some useful functions from file Functions.h
you can find this file in the src directory */
#include "Functions.h"

/* #####
* Functions
* ##### */

/* ----- */
/*! \fn      int ProgramDigitizer(int handle, DigitizerParams_t Params,
                               CAEN_DGTZ_DPPParamsPHA_t DPPParams)
* \brief    Program the registers of the digitizer with the relevant parameters
* \return   0=success; -1=error */
/* ----- */
int ProgramDigitizer(int handle, DigitizerParams_t Params,
                    CAEN_DGTZ_DPP_PSD_Params_t DPPParams)
{
    /* This function uses the CAENDigitizer API functions to perform the digitizer's
    initial configuration */

    int i, ret = 0;

    /* Reset the digitizer */
    ret |= CAEN_DGTZ_Reset(handle);

    if (ret) {
        printf("ERROR: can't reset the digitizer.\n");
    }
}

```

```

    return -1;
}

/* Set the DPP acquisition mode
This setting affects the modes Mixed and List (see CAEN_DGTZ_DPP_AcqMode_t definition
for details)
CAEN DGTZ DPP SAVE PARAM EnergyOnly          Only energy (DPP-PHA) or charge (DPP-
PSD/DPP-CI v2) is returned
CAEN DGTZ DPP SAVE PARAM TimeOnly           Only time is returned
CAEN DGTZ DPP SAVE PARAM EnergyAndTime     Both energy/charge and time are returned
CAEN DGTZ DPP_SAVE_PARAM_None             No histogram data is returned */
ret |= CAEN_DGTZ_SetDPPAcquisitionMode(handle, Params.AcqMode,
CAEN DGTZ DPP SAVE PARAM EnergyAndTime);

// Set the digitizer acquisition mode (CAEN DGTZ SW CONTROLLED or
CAEN DGTZ S IN CONTROLLED)
ret |= CAEN_DGTZ_SetAcquisitionMode(handle, CAEN_DGTZ_SW_CONTROLLED);

// Set the I/O level (CAEN DGTZ IOLevel NIM or CAEN DGTZ IOLevel TTL)
ret |= CAEN_DGTZ_SetIOLevel(handle, Params.IOlev);

/* Set the digitizer's behaviour when an external trigger arrives:

CAEN_DGTZ_TRGMODE_DISABLED: do nothing
CAEN DGTZ TRGMODE EXTOUT ONLY: generate the Trigger Output signal
CAEN DGTZ TRGMODE ACQ ONLY = generate acquisition trigger
CAEN DGTZ TRGMODE ACQ AND EXTOUT = generate both Trigger Output and acquisition trigger

see CAENDigitizer user manual, chapter "Trigger configuration" for details */
ret |= CAEN_DGTZ_SetExtTriggerInputMode(handle, CAEN_DGTZ_TRGMODE_ACQ_ONLY);

// Set the enabled channels
ret |= CAEN_DGTZ_SetChannelEnableMask(handle, Params.ChannelMask);

// Set how many events to accumulate in the board memory before being available for
readout
ret |= CAEN_DGTZ_SetDPPEventAggregation(handle, Params.EventAggr, 0);

/* Set the mode used to synchronize the acquisition between different boards.
In this example the sync is disabled */
ret |= CAEN_DGTZ_SetRunSynchronizationMode(handle, CAEN_DGTZ_RUN_SYNC_Disabled);

// Set the DPP specific parameters for the channels in the given channelMask
ret |= CAEN_DGTZ_SetDPPParameters(handle, Params.ChannelMask, &DPPParams);

for(i=0; i<MaxNChannels; i++) {
    if (Params.ChannelMask & (1<<i)) {
        // Set the number of samples for each waveform (you can set different RL for
different channels)
        ret |= CAEN_DGTZ_SetRecordLength(handle, Params.RecordLength, i);

        // Set a DC offset to the input signal to adapt it to digitizer's dynamic range
        ret |= CAEN_DGTZ_SetChannelDCOffset(handle, i, 0x8000);

        // Set the Pre-Trigger size (in samples)
        ret |= CAEN_DGTZ_SetDPPPreTriggerSize(handle, i, 80);

        // Set the polarity for the given channel (CAEN DGTZ PulsePolarityPositive or
CAEN DGTZ PulsePolarityNegative)
        ret |= CAEN_DGTZ_SetChannelPulsePolarity(handle, i, Params.PulsePolarity);
    }
}

/* Set the virtual probes

DPP-PSD for x720 boards can save:
2 analog waveforms:
    Analog Trace 1: it is always the input signal;
    Analog Trace 2: it can be specified with the VIRTUALPROBE parameter
4 digital waveforms:
    Digital Trace 1: it is always the trigger
    Digital Trace 2: it is always the long gate
    Digital Trace 3/4: they can be specified with the DIGITALPROBE 1 and 2 parameters

DPP-PSD for x751 boards can save:
2 analog waveforms:
    Analog Trace 1: it is always the input signal;
    Analog Trace 2: it can be specified with the VIRTUALPROBE parameter

```

```

3 digital waveforms:
  Digital Trace 1:  it is always the trigger
  Digital Trace 2/3: they can be specified with the DIGITALPROBE 1 and 2 parameters

CAEN_DGTZ_DPP_VIRTUALPROBE_SINGLE    -> Save only the Input Signal waveform
CAEN_DGTZ_DPP_VIRTUALPROBE_DUAL      -> Save also the waveform specified in
      VIRTUALPROBE

Virtual Probes types for Trace 2:
  CAEN_DGTZ_DPP_PSD_VIRTUALPROBE_Baseline    -> Save the Baseline waveform (mean
      on nsbl parameter)
  ### Virtual Probes only for FW <= 13X.5 ###
  CAEN_DGTZ_DPP_PSD_VIRTUALPROBE_Threshold    -> Save the (Baseline - Threshold)
      waveform. NOTE: x720 only

Digital Probes types for Digital Trace 3(x720)/2(x751):
  ### Virtual Probes only for FW >= 13X.6 ###
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_R6_ExtTrg    NOTE: x720 only
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_R6_OverThr
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_R6_TrigOut
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_R6_CoincWin
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_R6_PileUp
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_R6_Coincidence
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_R6_GateLong    NOTE: x751 only
  ### Virtual Probes only for FW <= 13X.5 ###
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_Armed    NOTE: x720 only
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_Trigger    NOTE: x720 only
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_ChargeReady    NOTE: x720 only
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_PileUp    NOTE: x720 only
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_B1OutSafeBand    NOTE: x720 only
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_B1Timeout    NOTE: x720 only
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_CoincidenceMet    NOTE: x720 only
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_Tvaw    NOTE: x720 only
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_OverThr    NOTE: x751 only
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_GateShort    NOTE: x751 only
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_None    NOTE: x751 only

Digital Probes types for Digital Trace 4(x720)/3(x751):
  ### Virtual Probes only for FW >= 13X.6 ###
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_R6_GateShort
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_R6_OverThr
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_R6_TrigVal
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_R6_TrigHO
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_R6_PileUp
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_R6_Coincidence
  ### Virtual Probes only for FW <= 13X.5 ###
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_Armed    NOTE: x720 only
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_Trigger    NOTE:
  x720 only
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_ChargeReady    NOTE:
  x720 only
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_PileUp    NOTE:
  x720 only
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_B1OutSafeBand    NOTE:
  x720 only
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_B1Timeout    NOTE:
  x720 only
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_CoincidenceMet    NOTE:
  x720 only
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_Tvaw    NOTE:
  x720 only
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_GateShort    NOTE: x751 only
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_GateLong    NOTE: x751 only
  CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_None    NOTE: x751 only */
ret |= CAEN_DGTZ_SetDPP_PSD_VirtualProbe(handle, CAEN_DGTZ_DPP_VIRTUALPROBE_SINGLE,
      CAEN_DGTZ_DPP_PSD_VIRTUALPROBE_Baseline,
      CAEN_DGTZ_DPP_PSD_DIGITALPROBE1_R6_GateLong,
      CAEN_DGTZ_DPP_PSD_DIGITALPROBE2_R6_OverThr);

if (ret) {
  printf("Warning: errors found during the programming of the digitizer.\nSome
      settings may not be executed\n");
  return ret;
} else {
  return 0;
}
}

```

```

/* ##### */
/* MAIN */
/* ##### */
int main(int argc, char *argv[])
{
    /* The following variable is the type returned from most of CAENDigitizer
    library functions and is used to check if there was an error in function
    execution. For example:
    ret = CAEN DGTZ some function(some args);
    if(ret) printf("Some error"); */
    CAEN_DGTZ_ErrorCode ret;

    /* Buffers to store the data. The memory must be allocated using the appropriate
    CAENDigitizer API functions (see below), so they must not be initialized here
    NB: you must use the right type for different DPP analysis (in this case PSD) */
    char *buffer = NULL; // readout buffer
    CAEN_DGTZ_DPP_PSD_Event_t *Events[MaxNChannels]; // events buffer
    CAEN_DGTZ_DPP_PSD_Waveforms_t *Waveform=NULL; // waveforms buffer

    /* The following variables will store the digitizer configuration parameters */
    CAEN_DGTZ_DPP_PSD_Params_t DPPParams[MAXNB];
    DigitizerParams_t Params[MAXNB];

    /* Arrays for data analysis */
    uint64_t PrevTime[MAXNB][MaxNChannels];
    uint64_t ExtendedTT[MAXNB][MaxNChannels];
    uint32_t *EHistoShort[MAXNB][MaxNChannels]; // Energy Histograms for short gate
    // charge integration
    uint32_t *EHistoLong[MAXNB][MaxNChannels]; // Energy Histograms for long gate charge
    // integration
    float *EHistoRatio[MAXNB][MaxNChannels]; // Energy Histograms for ratio Long/Short
    int ECnt[MAXNB][MaxNChannels]; // Number-of-Entries Counter for Energy
    // Histograms short and long gate
    int TrgCnt[MAXNB][MaxNChannels];

    /* The following variable will be used to get an handler for the digitizer. The
    handler will be used for most of CAENDigitizer functions to identify the board */
    int handle[MAXNB];

    /* Other variables */
    int i, b, ch, ev;
    int Quit=0;
    int AcqRun = 0;
    uint32_t AllocatedSize, BufferSize;
    int Nb=0;
    int DoSaveWave[MAXNB][MaxNChannels];
    int MajorNumber;
    int BitMask = 0;
    uint64_t CurrentTime, PrevRateTime, ElapsedTime;
    uint32_t NumEvents[MaxNChannels];
    CAEN_DGTZ_BoardInfo_t BoardInfo;

    memset(DoSaveWave, 0, MAXNB*MaxNChannels*sizeof(int));
    for (i=0; i<MAXNBITS; i++)
        BitMask |= 1<<i; /* Create a bit mask based on number of bits of the board */

    /* ##### */
    /* Set Parameters */
    /* ##### */
    memset(&Params, 0, MAXNB*sizeof(DigitizerParams_t));
    memset(&DPPParams, 0, MAXNB*sizeof(CAEN_DGTZ_DPP_PSD_Params_t));
    for(b=0; b<MAXNB; b++) {
        for(ch=0; ch<MaxNChannels; ch++) {
            EHistoShort[b][ch] = NULL; // Set all histograms pointers to NULL (we will
            // allocate them later)
            EHistoLong[b][ch] = NULL;
            EHistoRatio[b][ch] = NULL;
        }

        /*****\
        * Communication Parameters *
        \*****/
        // Direct USB connection
        //Params[b].LinkType = CAEN_DGTZ_USB; // Link Type
        //Params[b].VMEBaseAddress = 0; // For direct USB connection, VMEBaseAddress must
        // be 0

        // Direct optical connection
    }
}

```

```

Params[b].LinkType = CAEN_DGTZ_PCI_OpticalLink; // Link Type
Params[b].VMEBaseAddress = 0; // For direct CONET connection, VMEBaseAddress must
    be 0

// Optical connection to A2818 (or A3818) and access to the board with VME bus
//Params[b].LinkType = CAEN_DGTZ_PCI_OpticalLink; // Link Type
    (CAEN DGTZ PCIE OpticalLink for A3818)
//Params[b].VMEBaseAddress = 0x32100000; // VME Base Address (only for VME bus
    access; must be 0 for direct connection (CONET or USB)

// USB connection to V1718 bridge and access to the board with VME bus
//Params[b].LinkType = CAEN_DGTZ_USB; // Link Type (CAEN_DGTZ_PCI_OpticalLink for
    A3818)
//Params[b].VMEBaseAddress = 0x32110000; // VME Base Address (only for VME bus
    access; must be 0 for direct connection (CONET or USB)

Params[b].IOlev = CAEN_DGTZ_IOLevel_TTL;
/*****\
* Acquisition parameters *
\*****/
Params[b].AcqMode = CAEN_DGTZ_DPP_ACQ_MODE_Mixed; //
    CAEN_DGTZ_DPP_ACQ_MODE_List or
    CAEN_DGTZ_DPP_ACQ_MODE_Oscilloscope
Params[b].RecordLength = 12; // Num of samples of the
    waveforms (only for Oscilloscope mode)
Params[b].ChannelMask = 0xF; // Channel enable mask
Params[b].EventAggr = 17554; // number of events
    in one aggregate (0=automatic)
Params[b].PulsePolarity = CAEN_DGTZ_PulsePolarityNegative; // Pulse Polarity (this
    parameter can be individual)

/*****\
* DPP parameters *
\*****/
for(ch=0; ch<MaxNChannels; ch++) {
    DPPParams[b].thr[ch] = 50; // Trigger Threshold
    /* The following parameter is used to specify the number of samples for the
        baseline averaging:

        0 -> absolute Bl
        1 -> 4samp
        2 -> 8samp
        3 -> 16samp
        4 -> 32samp
        5 -> 64samp
        6 -> 128samp */
    DPPParams[b].nsbl[ch] = 2;
    DPPParams[b].lgate[ch] = 32; // Long Gate Width (N*4ns)
    DPPParams[b].sgate[ch] = 24; // Short Gate Width (N*4ns)
    DPPParams[b].pgate[ch] = 8; // Pre Gate Width (N*4ns)
    /* Self Trigger Mode:
        0 -> Disabled
        1 -> Enabled */
    DPPParams[b].selft[ch] = 1;
    // Trigger configuration:
    // CAEN_DGTZ_DPP_TriggerConfig_Peak -> trigger on peak. NOTE: Only for FW
        <= 13X.5
    // CAEN DGTZ DPP TriggerConfig Threshold -> trigger on threshold */
    DPPParams[b].trgc[ch] = CAEN_DGTZ_DPP_TriggerConfig_Threshold;
    /* Trigger Validation Acquisition Window */
    DPPParams[b].tvaw[ch] = 50;
    /* Charge sensibility: 0->40fc/LSB; 1->160fc/LSB; 2->640fc/LSB; 3->2,5pc/LSB */
    DPPParams[b].csens[ch] = 0;
}
/* Pile-Up rejection Mode
CAEN DGTZ DPP PSD PUR DetectOnly -> Only Detect Pile-Up
CAEN DGTZ DPP PSD PUR Enabled -> Reject Pile-Up */
DPPParams[b].purh = CAEN_DGTZ_DPP_PSD_PUR_DetectOnly;
DPPParams[b].purgap = 100; // Purity Gap
DPPParams[b].blthr = 3; // Baseline Threshold
DPPParams[b].bltmo = 100; // Baseline Timeout
DPPParams[b].trgho = 8; // Trigger HoldOff
}

/* *****/
/* Open the digitizer and read board information */
/* *****/
/* The following function is used to open the digitizer with the given connection
    parameters

```

```

and get the handler to it */
for(b=0; b<MAXNB; b++) {
    /* IMPORTANT: The following function identifies the different boards with a system
       which may change
       for different connection methods (USB, Conet, ecc). Refer to CAENDigitizer user
       manual for more info.
       Some examples below */

    /* The following is for b boards connected via b USB direct links
       in this case you must set Params[b].LinkType = CAEN DGTZ USB and
       Params[b].VMEBaseAddress = 0 */
    //ret = CAEN_DGTZ_OpenDigitizer(Params[b].LinkType, b, 0, Params[b].VMEBaseAddress,
        &handle[b]);

    /* The following is for b boards connected via 1 opticalLink in dasy chain
       in this case you must set Params[b].LinkType = CAEN DGTZ PCI OpticalLink and
       Params[b].VMEBaseAddress = 0 */
    ret = CAEN_DGTZ_OpenDigitizer(Params[b].LinkType, 0, b, Params[b].VMEBaseAddress,
        &handle[b]);

    /* The following is for b boards connected to A2818 (or A3818) via opticalLink (or
       USB with A1718)
       in this case the boards are accessed through VME bus, and you must specify the VME
       address of each board:
       Params[b].LinkType = CAEN DGTZ PCI OpticalLink (CAEN DGTZ PCIE OpticalLink for
       A3818 or CAEN DGTZ USB for A1718)
       Params[0].VMEBaseAddress = <0XXXXXXXX> (address of first board)
       Params[1].VMEBaseAddress = <0YYYYYYYY> (address of second board)
       etc */
    //ret = CAEN_DGTZ_OpenDigitizer(Params[b].LinkType, 0, 0, Params[b].VMEBaseAddress,
        &handle[b]);

    if (ret) {
        printf("Can't open digitizer\n");
        goto QuitProgram;
    }

    /* Once we have the handler to the digitizer, we use it to call the other functions
       */
    ret = CAEN_DGTZ_GetInfo(handle[b], &BoardInfo);
    if (ret) {
        printf("Can't read board info\n");
        goto QuitProgram;
    }
    printf("\nConnected to CAEN Digitizer Model %s, recognized as board %d\n",
        BoardInfo.ModelName, b);
    printf("ROC FPGA Release is %s\n", BoardInfo.ROC_FirmwareRel);
    printf("AMC FPGA Release is %s\n", BoardInfo.AMC_FirmwareRel);

    // Check firmware revision (only DPP firmware can be used with this Demo) */
    sscanf(BoardInfo.AMC_FirmwareRel, "%d", &MajorNumber);
    if (MajorNumber != 131 && MajorNumber != 132 ) {
        printf("This digitizer has not a DPP-PSD firmware\n");
        goto QuitProgram;
    }
}

/* ***** */
/* Program the digitizer (see function ProgramDigitizer) */
/* ***** */
for(b=0; b<MAXNB; b++) {
    ret = ProgramDigitizer(handle[b], Params[b], DPPParams[b]);
    if (ret) {
        printf("Failed to program the digitizer\n");
        goto QuitProgram;
    }
}

/* WARNING: The mallocs MUST be done after the digitizer programming,
because the following functions needs to know the digitizer configuration
to allocate the right memory amount */
/* Allocate memory for the readout buffer */
ret = CAEN_DGTZ_MallocReadoutBuffer(handle[0], &buffer, &AllocatedSize);
/* Allocate memory for the events */
ret |= CAEN_DGTZ_MallocDPPEvents(handle[0], Events, &AllocatedSize);
/* Allocate memory for the waveforms */
ret |= CAEN_DGTZ_MallocDPPWaveforms(handle[0], &Waveform, &AllocatedSize);
if (ret) {
    printf("Can't allocate memory buffers\n");
}

```



```

goto QuitProgram;
}

/* ***** */
/* Readout Loop */
/* ***** */
// Clear Histograms and counters
for(b=0; b<MAXNB; b++) {
    for(ch=0; ch<MaxNChannels; ch++) {
        // Allocate Memory for Histos and set them to 0
        EHistoShort[b][ch] = (uint32_t *)malloc( (1<<MAXNBITS)*sizeof(uint32_t) );
        memset(EHistoShort[b][ch], 0, (1<<MAXNBITS)*sizeof(uint32_t));
        EHistoLong[b][ch] = (uint32_t *)malloc( (1<<MAXNBITS)*sizeof(uint32_t) );
        memset(EHistoLong[b][ch], 0, (1<<MAXNBITS)*sizeof(uint32_t));
        EHistoRatio[b][ch] = (float *)malloc( (1<<MAXNBITS)*sizeof(float) );
        memset(EHistoRatio[b][ch], 0, (1<<MAXNBITS)*sizeof(float));
        TrgCnt[b][ch] = 0;
        ECnt[b][ch] = 0;
        PrevTime[b][ch] = 0;
        ExtendedTT[b][ch] = 0;
    }
}
PrevRateTime = get_time();
AcqRun = 0;
PrintInterface();
printf("Type a command: ");
while(!Quit) {

    // Check keyboard
    if(kbhit()) {
        char c;
        c = getch();
        if (c=='q') Quit = 1;
        if (c=='t')
            for(b=0; b<MAXNB; b++)
                CAEN_DGTZ_SendSWtrigger(handle[b]); /* Send a software trigger to each
                board */

        if (c=='h')
            for(b=0; b<MAXNB; b++)
                for(ch=0; ch<MaxNChannels; ch++)
                    if( ECnt[b][ch] != 0) {
                        /* Save Histograms to file for each board and channel */
                        SaveHistogram("HistoShort", b, ch, EHistoShort[b][ch]);
                        SaveHistogram("HistoLong", b, ch, EHistoLong[b][ch]);
                    }

        if (c=='w')
            for(b=0; b<MAXNB; b++)
                for(ch=0; ch<MaxNChannels; ch++)
                    DoSaveWave[b][ch] = 1; /* save waveforms to file for each channel
                    for each board (at next trigger) */

        if (c=='r') {
            for(b=0; b<MAXNB; b++) {
                CAEN_DGTZ_SWStopAcquisition(handle[b]);
                printf("Restarted\n");
                CAEN_DGTZ_ClearData(handle[b]);
                CAEN_DGTZ_SWStartAcquisition(handle[b]);
            }
        }

        if (c=='s') {
            for(b=0; b<MAXNB; b++) {
                // Start Acquisition
                // NB: the acquisition for each board starts when the following line is
                // executed
                // so in general the acquisition does NOT starts synchronously for
                // different boards
                CAEN_DGTZ_SWStartAcquisition(handle[b]);
                printf("Acquisition Started for Board %d\n", b);
            }
            AcqRun = 1;
        }

        if (c=='S') {
            for(b=0; b<MAXNB; b++) {
                // Stop Acquisition
                CAEN_DGTZ_SWStopAcquisition(handle[b]);
                printf("Acquisition Stopped for Board %d\n", b);
            }
            AcqRun = 0;
        }
    }
}

```

```

    }
}
if (!AcqRun) {
    Sleep(10);
    continue;
}

/* Calculate throughput and trigger rate (every second) */
CurrentTime = get time();
ElapsedTime = CurrentTime - PrevRateTime; /* milliseconds */
if (ElapsedTime > 1000) {
    system(CLEARSCR);
    PrintInterface();
    printf("Readout Rate=%.2f MB\n", (float)Nb/((float)ElapsedTime*1048.576f));
    for(b=0; b<MAXNB; b++) {
        printf("\nBoard %d:\n",b);
        for(i=0; i<MaxNChannels; i++) {
            if (TrgCnt[b][i]>0)
                printf("\tCh %d:\tTrgRate=%.2f KHz\t%\n", b*8+i,
                    (float)TrgCnt[b][i]/(float)ElapsedTime);
            else
                printf("\tCh %d:\tNo Data\n", i);
            TrgCnt[b][i]=0;
        }
    }
    Nb = 0;
    PrevRateTime = CurrentTime;
    printf("\n\n");
}

/* Read data from the boards */
for(b=0; b<MAXNB; b++) {
    /* Read data from the board */
    ret = CAEN_DGTZ_ReadData(handle[b], CAEN_DGTZ_SLAVE_TERMINATED_READOUT_MBLT,
        buffer, &BufferSize);

    if (ret) {
        printf("Readout Error\n");
        goto QuitProgram;
    }
    if (BufferSize == 0)
        continue;

    Nb += BufferSize;
    //ret = DataConsistencyCheck((uint32_t *)buffer, BufferSize/4);
    ret |= CAEN_DGTZ_GetDPPEvents(handle[b], buffer, BufferSize, Events,
        NumEvents);

    if (ret) {
        printf("Data Error: %d\n", ret);
        goto QuitProgram;
    }

    /* Analyze data */
    for(ch=0; ch<MaxNChannels; ch++) {
        if (!(Params[b].ChannelMask & (1<<ch)))
            continue;

        /* Update Histograms */
        for(ev=0; ev<NumEvents[ch]; ev++) {
            TrgCnt[b][ch]++;
            /* Time Tag */
            if (Events[ch][ev].TimeTag < PrevTime[b][ch])
                ExtendedTT[b][ch]++;
            PrevTime[b][ch] = Events[ch][ev].TimeTag;
            /* Energy */
            if ( (Events[ch][ev].ChargeLong > 0) && (Events[ch][ev].ChargeShort >
                0)) {
                // Fill the histograms
                EHistoShort[b][ch][(Events[ch][ev].ChargeShort) & BitMask]++;
                EHistoLong[b][ch][(Events[ch][ev].ChargeLong) & BitMask]++;
                ECnt[b][ch]++;
            }

            /* Get Waveforms (only from 1st event in the buffer) */
            if ((Params[b].AcqMode != CAEN_DGTZ_DPP_ACQ_MODE_List) &&
                DoSaveWave[b][ch] && (ev == 0)) {
                int size;
                int16_t *WaveLine;
                uint8_t *DigitalWaveLine;

```

```

CAEN_DGTZ_DecodeDPPWaveforms(handle[b], &Events[ch][ev], Waveform);

// Use waveform data here...
size = (int)(Waveform->Ns); // Number of samples
WaveLine = Waveform->Trace1; // First trace (for DPP-PSD it is
    ALWAYS the Input Signal)
SaveWaveform(b, ch, 1, size, WaveLine);

WaveLine = Waveform->Trace2; // Second Trace (if single trace mode,
    it is a sequence of zeroes)
SaveWaveform(b, ch, 2, size, WaveLine);
DoSaveWave[b][ch] = 0;

DigitalWaveLine = Waveform->DTrace1; // First Digital Trace (Gate
    Short)
SaveDigitalProbe(b, ch, 1, size, DigitalWaveLine);
DoSaveWave[b][ch] = 0;

DigitalWaveLine = Waveform->DTrace2; // Second Digital Trace (Gate
    Long)
SaveDigitalProbe(b, ch, 2, size, DigitalWaveLine);
DoSaveWave[b][ch] = 0;

DigitalWaveLine = Waveform->DTrace3; // Third Digital Trace
    (DIGITALPROBE1 set with CAEN DGTZ SetDPP PSD VirtualProbe)
SaveDigitalProbe(b, ch, 3, size, DigitalWaveLine);
DoSaveWave[b][ch] = 0;

DigitalWaveLine = Waveform->DTrace4; // Fourth Digital Trace
    (DIGITALPROBE2 set with CAEN DGTZ SetDPP PSD VirtualProbe)
SaveDigitalProbe(b, ch, 4, size, DigitalWaveLine);
DoSaveWave[b][ch] = 0;
printf("Waveforms                saved                to
    'Waveform_<board>_<channel>_<trace>.txt'\n");
    } // loop to save waves
    } // loop on events
    } // loop on channels
    } // loop on boards
} // End of readout loop

QuitProgram:
/* stop the acquisition, close the device and free the buffers */
for(b=0; b<MAXNB; b++) {
    CAEN_DGTZ_SWStopAcquisition(handle[b]);
    CAEN_DGTZ_CloseDigitizer(handle[b]);
    for (ch=0; ch<MaxNChannels; ch++) {
        free(EHistoShort[b][ch]);
        free(EHistoLong[b][ch]);
        free(EHistoRatio[b][ch]);
    }
}
CAEN_DGTZ_FreeReadoutBuffer(&buffer);
CAEN_DGTZ_FreeDPPEvents(handle[0], Events);
CAEN_DGTZ_FreeDPPWaveforms(handle[0], Waveform);
return ret;
}

```

6 ZLE x751 specific functions

Under development.

7 Examples of communication settings

Example No.1

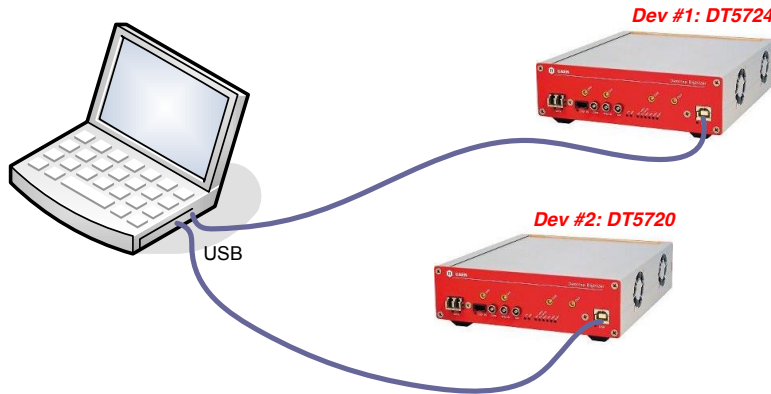


Fig. 7.1: Connection example no.1.

The host PC is connected via 2 USB ports to two desktop digitizer:

- Dev#1: DT5724 - 4 Channel 14 bit 100 MS/s Digitizer
- Dev#2: DT5720 - 4 Channel 12 bit 250 MS/s Digitizer

The computer is first connected to DT5724 then to the DT5720.

Open Dev#1: DT5724 connected via USB cable

```
CAEN_DGTZ_OpenDigitizer (
    CAEN_DGTZ_USB,      LinkType: physical communication channel = USB
    0,                  LinkNum: Link number = 0 first device
    0,                  ConetNode: if USB = 0
    0,                  VMEBaseAddress: must be = 0
    &handleDT5724 1     Pointer to the handler returned by function
);
```

Open Dev#2: DT5720 connected via USB cable

```
CAEN_DGTZ_OpenDigitizer (
    CAEN_DGTZ_USB,      LinkType: physical communication channel = USB
    1,                  LinkNum: Link number = 1 second device
    0,                  ConetNode: if USB = 0
    0,                  VMEBaseAddress: must be = 0
    &handleDT5720_2     Pointer to the handler returned by function
);
```

Arguments description

Name	Description
LinkType	= CAEN_DGTZ_USB. Indicates USB as the physical communication channel.
LinkNum	Link number: in case of USB, the link numbers are assigned by the PC when you connect the cable to the device; it is 0 for the first device (DT5724), 1 for the second (DT5720) There is not a fixed correspondence between the USB port and the link number.
ConetNode	In case of USB, ConetNode must be 0.
VMEBaseAddress	Not used = 0 (used only for model accessed via VME).

Example No.2

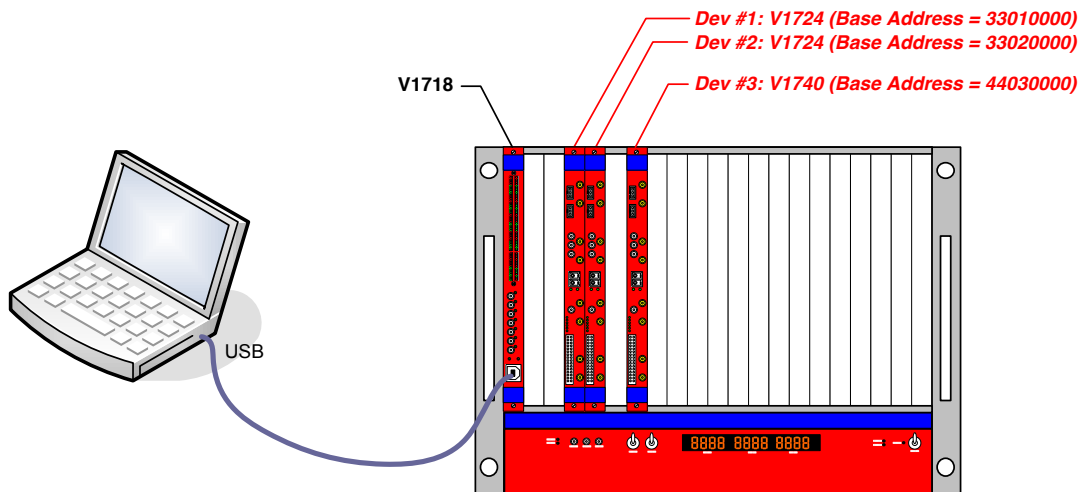


Fig. 7.2: Connection example no.2.

The host PC is connected via USB ports to one V1718 VME-USB2.0 Bridge housed in a VME crate. The crate contains also the following boards

- Dev#1: V1724 - 8 Channel 14 bit 100 MS/s Digitizer (Base address = 0x33010000)
- Dev#2: V1724 - 8 Channel 14 bit 100 MS/s Digitizer (Base address = 0x33020000)
- Dev#3: V1740 - 64 Channel 12 bit 62.5 MS/s Digitizer (Base address = 0x44030000)

Open Dev#1: V1724 (VME base address 0x33010000) accessed via VMEbus through the V1718:

```
CAEN_DGTZ_OpenDigitizer (
    CAEN_DGTZ_USB,      LinkType: physical communication channel = USB
    0,                  LinkNum: Link number = 0 first device
    0,                  ConetNode: if USB = 0
    0x33010000,         VMEBaseAddress
    &handleV1724_1     Pointer to the handler returned by function
);
```

Open Dev#2: V1724 (VME base address 0x33020000) accessed via VMEbus through the V1718:

```
CAEN_DGTZ_OpenDigitizer (
    CAEN_DGTZ_USB,      LinkType: physical communication channel = USB
    0,                  LinkNum: Link number = 0 first device
    0,                  ConetNode: if USB = 0
    0x33020000,         VMEBaseAddress
    &handleV1724_2     Pointer to the handler returned by function
);
```

Open Dev#3: V1740 (VME base address 0x44030000) accessed via VMEbus through the V1718:

```
CAEN_DGTZ_OpenDigitizer (
    CAEN_DGTZ_USB,      LinkType: physical communication channel = USB
    0,                  LinkNum: Link number = 0 first device
    0,                  ConetNode: if USB = 0
    0x44030000,         VMEBaseAddress
    &handleV1740_3     Pointer to the handler returned by function
);
```

Arguments description

Name	Description
LinkType	= CAEN_DGTZ_USB. Indicates USB as the physical communication channel .
LinkNum	Link number: in case of USB, the link numbers are assigned by the PC when you connect the cable to the device; it is 0 for the first device, 1 for the second. There is not a fixed correspondence between the USB port and the link number.
ConetNode	In case of USB, ConetNode must be 0.
VMEBaseAddress	VME Base Address of the board (rotary switches setting) expressed as a 32 bit number. This argument is used only for the VME models accessed through the VME bus and MUST BE 0 in all other cases.

Example No.3

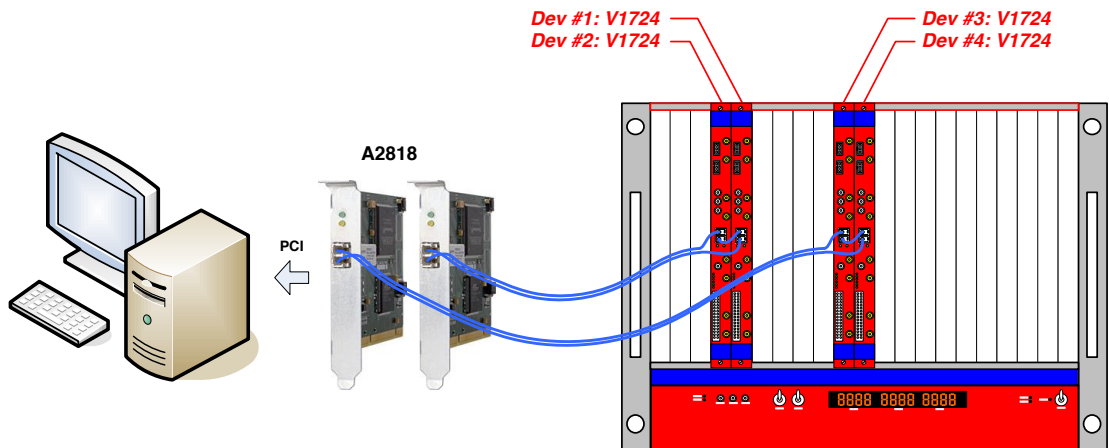


Fig. 7.3: Connection example no.3.

The host PC houses two CAEN A2818 PCI CONET Controllers; the VME crate houses the following boards:

- Two V1724 Digitizer connected in a Daisy chain between them end to the A2818 #0: Dev#1 (first in Daisy chain) and Dev#2 (second in Daisy chain)
- Two V1724 Digitizer connected in a Daisy chain between them end to the A2818 #1: Dev#3 (first in Daisy chain) and Dev#4 (second in Daisy chain)



Note: □ The A2818 number refers to the PCI slot and depends on the motherboard of the PC used. **It is not known a priori which PCI card is assigned to which number.** In this example we assume that the A2818 connected to Dev#1 and Dev#2, is inserted into the first PCI slot and get Link Number = 0.

Open Dev#1: V1724 first device in Daisy chain of A2818#0:

```
CAEN_DGTZ_OpenDigitizer (
    CAEN_DGTZ_PCI_OpticalLink, LinkType: physical communication channel = Optical Link via A2818 (PCI Controller)
    0, LinkNum: Link number = 0 first device
    0, ConetNode: first device in the chain = 0
    0, VMEBaseAddress: must be = 0
    &handleV1724_1 Pointer to the handler returned by function
);
```

Open Dev#2: V1724 second device in Daisy chain of A2818#0:

```
CAEN_DGTZ_OpenDigitizer (
    CAEN_DGTZ_PCI_OpticalLink, LinkType: physical communication channel = Optical Link via A2818 (PCI Controller)
    0, LinkNum: Link number = 0 first device
    1, ConetNode: second device in the chain = 1
    0, VMEBaseAddress: must be = 0
    &handleV1724_2 Pointer to the handler returned by function
);
```

Open Dev#3: V1724 first device in Daisy chain of A2818#1:

```
CAEN_DGTZ_OpenDigitizer (
    CAEN_DGTZ_PCI_OpticalLink, LinkType: physical communication channel = Optical Link via A2818 (PCI Controller)
    1, LinkNum: Link number = 1 second device
    0, ConetNode: first device in the chain = 0
    0, VMEBaseAddress: must be = 0
    &handleV1724_3 Pointer to the handler returned by function
);
```

Open Dev#4: V1724 second device in Daisy chain of A2818#1:

```
CAEN_DGTZ_OpenDigitizer (
    CAEN_DGTZ_PCI_OpticalLink, LinkType: physical communication channel = Optical Link via A2818 (PCI Controller)
    1, LinkNum: Link number = 1 second device
    1, ConetNode: second device in the chain = 1
    0, VMEBaseAddress: must be = 0
    &handleV1724_4 Pointer to the handler returned by function
);
```

Arguments description

Name	Description
LinkType	= CAEN_DGTZ_PCI_OpticalLink. Indicates A2818 -> Optical Link, either direct connection or VME through V2718 as the physical communication channel .
LinkNum	Link number: For the CONET, the link number indicates which link of A2818 or A3818 is used. For A2818 refers to the PCI slot and depends on the motherboard of the PC used. Link index start from 0 (1 st link in the 1 st slot used). It is not known a priori which is the first slot used.
ConetNode	The CONET node identifies which device in the Daisy chain is being addressed. The node is 0 for the first device in the chain, 1 for the second and so on. See Fig. 7.4.
VMEBaseAddress	Not used = 0 (used only for model accessed via VME).

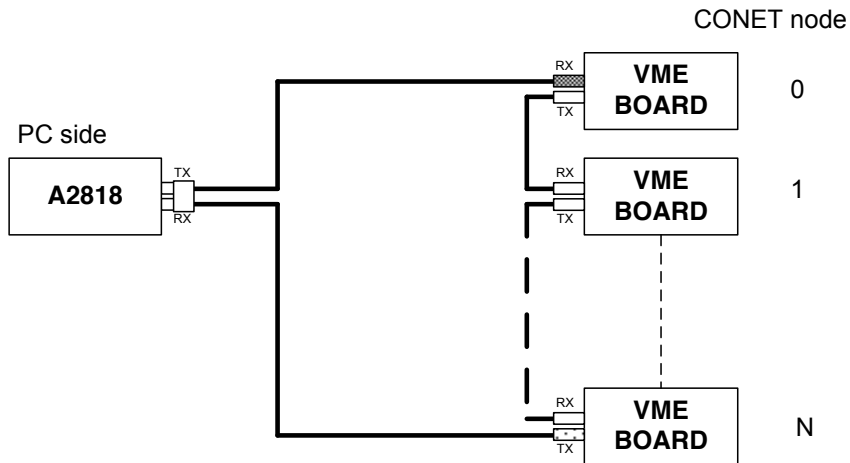


Fig. 7.4: A2818 network scheme.

Example No.4

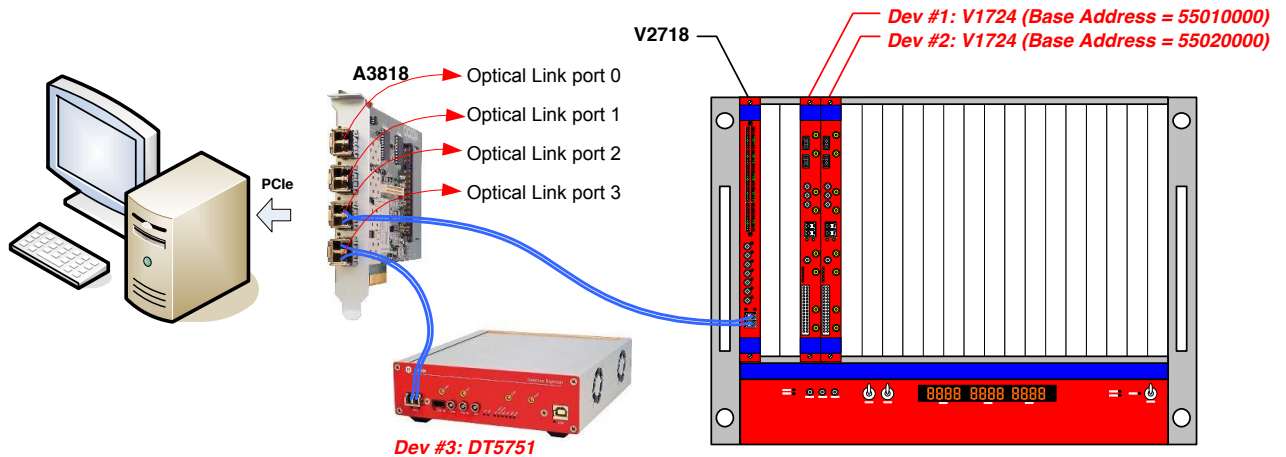


Fig. 7.5: Connection example no.4

The host PC houses one CAEN A3818C PCIe CONET Controller with 4 Optical Link;

- port#3 is connected to Dev#3 (DT5751 - 2/4 Channel 10 bit 2/1 GS/s Digitizer)
- port#2 is connected to a V2718 VME-PCI Optical Link Bridge housed in a VME crate that contains the following boards:
 - Dev#1: V1724 - 8 Channel 14 bit 100 MS/s Digitizer (Base address = 0x55010000)
 - Dev#2: V1724 - 8 Channel 14 bit 100 MS/s Digitizer (Base address = 0x55020000)

Open Dev#1: V1724 (VME base address 0x55010000) accessed via VMEbus through the V2718 connected to A3818 port#2:

```
CAEN_DGTZ_OpenDigitizer (
    CAEN_DGTZ_PCIE_OpticalLink, LinkType: physical communication channel = Optical Link via A3818 (PCIe Controller)
    2, LinkNum: unique device, Link number =A3818 port number: 2
    0, ConetNode: unique device in the chain =0
    0x55010000, VMEBaseAddress
    &handleV1724_1 Pointer to the handler returned by function
);
```

Open Dev#2: V1724 (VME base address 0x55020000) accessed via VMEbus through the V2718 connected to A3818 port#2:

```
CAEN DGTZ_OpenDigitizer (
    CAEN_DGTZ_PCIE_OpticalLink, LinkType: physical communication channel = Optical Link via A3818 (PCIe Controller)
    2, LinkNum: unique device, Link number =A3818 port number: 2
    0, ConetNode: unique device in the chain =0
    0x55020000, VMEBaseAddress
    &handleV1724_2 Pointer to the handler returned by function
);
```

Open Dev#3: DT5751 first device in Daisy chain of A3818 port#2

```
CAEN_DGTZ_OpenDigitizer (
    CAEN DGTZ_PCIE OpticalLink, LinkType: physical communication channel = Optical Link via A3818 (PCIe Controller)
    3, LinkNum: unique device, Link number =A3818 port number: 3
    0, ConetNode: unique device in the chain =0
    0, VMEBaseAddress: must be = 0
    &handleDT5751_3 Pointer to the handler returned by function
);
```

Arguments description

Name	Description
LinkType	= CAEN_DGTZ_PCIE_OpticalLink. Indicates A3818 -> Optical Link, either direct connection or VME through V2718 as the physical communication channel .
LinkNum	Link number: For the CONET, the link number indicates which link of A2818 or A3818 is used. For A3818 refers to the PCI slot and depends on the motherboard of the PC used. Link index start from 0 (1 st Optical link port in the 1 st slot used). It is not known a priori which is the first slot used. Important note: if also A2818s are installed, these ones have lower index assigned.
ConetNode	The CONET node identifies which device in the Daisy chain is being addressed. The node is 0 for the first device in the chain, 1 for the second and so on .
VMEBaseAddress	used only for model accessed via VME. Must be 0 in other cases

Example No.5

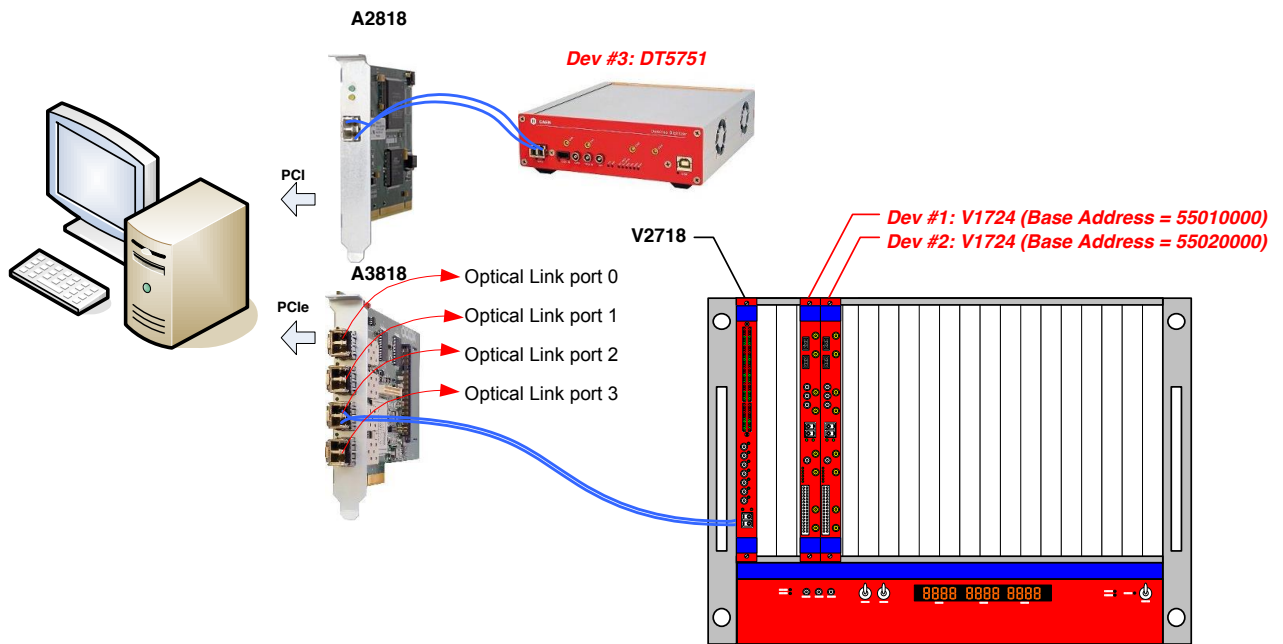


Fig. 7.6: Connection example no.5

The host PC houses

- one A2818 PCI CONET Controller connected to Dev#3 (DT5751 - 2/4 Channel 10 bit 2/1 GS/s Digitizer)
- one CAEN A3818C PCIe CONET Controller with 4 Optical Link; with port#2 connected to a V2718 VME-PCI Optical Link Bridge housed in a VME crate that contains the following boards:
 - Dev#1: V1724 - 8 Channel 14 bit 100 MS/s Digitizer (Base address = 0x55010000)
 - Dev#2: V1724 - 8 Channel 14 bit 100 MS/s Digitizer (Base address = 0x55020000)

Open Dev#1: V1724 (VME base address 0x55010000) accessed via VMEbus through the V2718 connected to A3818 port#2:

```
CAEN_DGTZ_OpenDigitizer (
    CAEN_DGTZ_PCIE_OpticalLink, LinkType: physical communication channel = Optical Link via A3818 (PCIe Controller)
    3, LinkNum: 3 = A3818 port number+1 (to A2818 is assigned the first link =0)
    0, ConetNode: unique device in the chain =0
    0x55010000, VMEBaseAddress
    &handleV1724_1 Pointer to the handler returned by function
);
```

Open Dev#2: V1724 (VME base address 0x55020000) accessed via VMEbus through the V2718 connected to A3818 port#2:

```
CAEN_DGTZ_OpenDigitizer (
    CAEN_DGTZ_PCIE_OpticalLink, LinkType: physical communication channel = Optical Link via A3818 (PCIe Controller)
    3, LinkNum: 3 = A3818 port number+1 (to A2818 is assigned the first link =0)
    0, ConetNode: unique device in the chain =0
    0x55020000, VMEBaseAddress
    &handleV1724_1 Pointer to the handler returned by function
);
```

Open Dev#3: DT5751 first device in Daisy chain of A2818

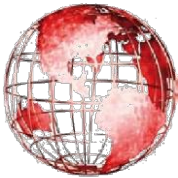
```
CAEN_DGTZ_OpenDigitizer (
    CAEN_DGTZ_PCI_OpticalLink, LinkType: physical communication channel = Optical Link via A2818 (PCI Controller)
    0, LinkNum: A2818 has lower index assigned = 0
    0, ConetNode: unique device in the chain =0
    0, VMEBaseAddress: must be = 0
    &handleDT5751_3 Pointer to the handler returned by function
);
```

Arguments description

Name	Description
LinkType	= CAEN_DGTZ_PCIE_OpticalLink. Indicates A3818 -> Optical Link, either direct connection or VME through V2718 as the physical communication channel . = CAEN_DGTZ_PCI_OpticalLink. Indicates A2818 -> Optical Link, either direct connection or VME through V2718 as the physical communication channel .
LinkNum	Link number: For the CONET, the link number indicates which link of A2818 or A3818 is used. For A3818/A2818 refers to the PCI slot and depends on the motherboard of the PC used. Link index start from 0 (1 st Optical link port in the 1 st slot used). It is not known a priori which is the first slot used. Important note: if also A2818s are installed, these ones have lower index assigned.
ConetNode	The CONET node identifies which device in the Daisy chain is being addressed. The node is 0 for the first device in the chain, 1 for the second and so on .
VMEBaseAddress	Used only for model accessed via VME. Must be 0 in other cases



CAEN SpA is acknowledged as the only company in the world providing a complete range of High/Low Voltage Power Supply systems and Front-End/Data Acquisition modules which meet IEEE Standards for Nuclear and Particle Physics. Extensive Research and Development capabilities have allowed CAEN SpA to play an important, long term role in this field. Our activities have always been at the forefront of technology, thanks to years of intensive collaborations with the most important Research Centres of the world. Our products appeal to a wide range of customers including engineers, scientists and technical professionals who all trust them to help achieve their goals faster and more effectively.

**CAEN S.p.A.**

Via Vetraia, 11
55049 Viareggio
Italy
Tel. +39.0584.388.398
Fax +39.0584.388.959
info@caen.it
www.caen.it

CAEN GmbH

Klingenstraße 108
D-42651 Solingen
Germany
Tel. +49 (0)212 254 4077
Mobile +49 (0)151 16 548 484
Fax +49 (0)212 25 44079
info@caen-de.com
www.caen-de.com
CAEN GmbH

CAEN Technologies, Inc.

1140 Bay Street - Suite 2 C
Staten Island, NY 10305
USA
Tel. +1.718.981.0401
Fax +1.718.556.9185
info@caentechnologies.com
www.caentechnologies.com