libcaer

Release master

iniVation AG

Jul 08, 2023

CONTENTS

CHAPTER

ONE

API

Full API documentation, automatically generated from doxygen comments.

class **AEDAT3NetworkHeader** : private aedat3_network_header

Public Functions

inline **AEDAT3NetworkHeader**()

inline **AEDAT3NetworkHeader**(const uint8_t *h)

inline int64_t **getMagicNumber**() const noexcept

inline bool **checkMagicNumber**() const noexcept

inline int64_t **getSequenceNumber**() const noexcept

inline void **incrementSequenceNumber**() noexcept

inline int8_t **getVersionNumber**() const noexcept

inline bool **checkVersionNumber**() const noexcept

inline int8_t **getFormatNumber**() const noexcept

inline void **setFormatNumber**(int8_t format) noexcept

inline int16_t **getSourceID**() const noexcept

inline void **setSourceID**(int16_t source) noexcept

struct **caer_bias_coarsefine**

#include </builds/inivation/dv/libcaer/include/libcaer/devices/davis.h> On-chip coarse-fine bias current configuration. See 'https://inivation.com/support/hardware/biasing/'[1] for more details.

Public Members

uint8_t **coarseValue**

Coarse current, from 0 to 7, creates big variations in output current.

uint8_t **fineValue**

Fine current, from 0 to 255, creates small variations in output current.

bool **enabled**

Whether this bias is enabled or not.

bool **sexN**

Bias sex: true for 'N' type, false for 'P' type.

bool **typeNormal**

Bias type: true for 'Normal', false for 'Cascode'.

bool **currentLevelNormal**

Bias current level: true for 'Normal, false for 'Low'.

struct **caer_bias_coarsefine1024**

#include </builds/inivation/dv/libcaer/include/libcaer/devices/dvs132s.h> On-chip simplified coarse-fine bias current configuration. See 'https://inivation.com/support/hardware/biasing/'[2] for more details.

Public Members

uint16_t **coarseValue**

Coarse current, from 0 to 1023, creates big variations in output current.

uint16_t **fineValue**

Fine current, from 0 to 1023, creates small variations in output current.

struct **caer_bias_dynapse**

#include </builds/inivation/dv/libcaer/include/libcaer/devices/dynapse.h> On-chip coarse-fine bias current configuration for Dynap-se. See 'https://ai-ctx.com/support/'[3] section 'Neuron's behaviors and parameters tuning'.

[1] https://inivation.com/support/hardware/biasing/'
[2] https://inivation.com/support/hardware/biasing/'

Public Members

uint8_t **biasAddress**

Address of bias to configure, see DYNAPSE_CONFIG_BIAS_* defines.

uint8_t **coarseValue**

Coarse current, from 0 to 7, creates big variations in output current.

uint8_t **fineValue**

Fine current, from 0 to 255, creates small variations in output current.

bool **enabled**

Whether this bias is enabled or not.

bool **sexN**

Bias sex: true for 'N' type, false for 'P' type.

bool **typeNormal**

Bias type: true for 'Normal', false for 'Cascode'.

bool **biasHigh**

Bias current level: true for 'HighBias', false for 'LowBias'.

struct **caer_bias_shiftedsource**

#include </builds/inivation/dv/libcaer/include/libcaer/devices/davis.h> On-chip shifted-source bias current configuration. See 'https://inivation.com/support/hardware/biasing/'[4] for more details.

Public Members

uint8_t **refValue**

Shifted-source bias level, from 0 to 63.

uint8_t **regValue**

Shifted-source bias current for buffer amplifier, from 0 to 63.

enum *caer_bias_shiftedsource_operating_mode* **operatingMode**

Shifted-source operating mode (see 'enum caer_bias_shiftedsource_operating_mode').

enum *caer_bias_shiftedsource_voltage_level* **voltageLevel**

Shifted-source voltage level (see 'enum caer_bias_shiftedsource_voltage_level').

struct **caer_bias_vdac**

#include </builds/inivation/dv/libcaer/include/libcaer/devices/davis.h> On-chip voltage digital-to-analog converter configuration. See 'https://inivation.com/support/hardware/biasing/'[5] for more details.

[3] https://ai-ctx.com/support/'
[4] https://inivation.com/support/hardware/biasing/'

Public Members

uint8_t **voltageValue**

Voltage, between 0 and 63, as a fraction of 1/64th of VDD=3.3V.

uint8_t **currentValue**

Current, between 0 and 7, that drives the voltage.

struct **caer_davis_info**

#include </builds/inivation/dv/libcaer/include/libcaer/devices/davis.h> DAVIS device-related information.

Public Members

int16_t **deviceID**

Unique device identifier. Also 'source' for events.

char **deviceSerialNumber**[8 + 1]

Device serial number.

uint8_t **deviceUSBBusNumber**

Device USB bus number.

uint8_t **deviceUSBDeviceAddress**

Device USB device address.

char ***deviceString**

Device information string, for logging purposes. If not NULL, pointed-to memory is *only* valid while the corresponding device is open! After calling deviceClose() this is invalid memory!

int16_t **firmwareVersion**

USB firmware version.

int16_t **logicVersion**

Logic (FPGA/CPLD) version.

int16_t **chipID**

Chip identifier/type.

bool **deviceIsMaster**

Whether the device is a time-stamp master or slave.

bool **muxHasStatistics**

Feature test: Multiplexer statistics support (event drops).

[5] https://inivation.com/support/hardware/biasing/'

int16_t **dvsSizeX**

DVS X axis resolution.

int16_t **dvsSizeY**

DVS Y axis resolution.

bool **dvsHasPixelFilter**

Feature test: DVS pixel-level filtering.

bool **dvsHasBackgroundActivityFilter**

Feature test: DVS Background Activity filter (and Refractory Period filter).

bool **dvsHasROIFilter**

Feature test: DVS ROI filter.

bool **dvsHasSkipFilter**

Feature test: DVS event skip filter.

bool **dvsHasPolarityFilter**

Feature test: DVS polarity suppression filter.

bool **dvsHasStatistics**

Feature test: DVS statistics support.

int16_t **apsSizeX**

APS X axis resolution.

int16_t **apsSizeY**

APS Y axis resolution.

enum *caer_frame_event_color_filter* **apsColorFilter**

APS color filter type.

bool **apsHasGlobalShutter**

Feature test: APS supports Global Shutter.

enum *caer_imu_types* **imuType**

IMU chip type on device.

bool **extInputHasGenerator**

Feature test: External Input module supports Signal-Generation.

struct **caer_device_discovery_result**

#include </builds/inivation/dv/libcaer/include/libcaer/devices/device_discover.h> Result of a device discovery operation. Contains the type of the device and its informational structure; use the device type to properly select the right info structure! In the info structures, 'deviceID' will always be set to -1 and 'deviceString' will always be NULL, as those are not present during the generic discovery phase.

Public Members

uint16_t **deviceType**

bool **deviceErrorOpen**

bool **deviceErrorVersion**

struct *caer_dvs128_info* **dvs128Info**

struct *caer_edvs_info* **edvsInfo**

struct *caer_davis_info* **davisInfo**

struct *caer_dynapse_info* **dynapseInfo**

struct *caer_dvs132s_info* **dvs132sInfo**

struct *caer_dvx_info* **dvXplorerInfo**

struct *caer_samsung_evk_info* **samsungEVKInfo**

union *caer_device_discovery_result* **deviceInfo**

struct **caer_dvs128_info**

#include </builds/inivation/dv/libcaer/include/libcaer/devices/dvs128.h> DVS128 device-related information.

Public Members

int16_t **deviceID**

Unique device identifier. Also 'source' for events.

char **deviceSerialNumber**[8 + 1]

Device serial number.

uint8_t **deviceUSBBusNumber**

Device USB bus number.

uint8_t **deviceUSBDeviceAddress**

Device USB device address.

char *deviceString

Device information string, for logging purposes. If not NULL, pointed-to memory is *only* valid while the corresponding device is open! After calling deviceClose() this is invalid memory!

int16_t firmwareVersion

USB firmware version.

bool deviceIsMaster

Whether the device is a time-stamp master or slave.

int16_t dvsSizeX

DVS X axis resolution.

int16_t dvsSizeY

DVS Y axis resolution.

struct caer_dvs132s_info

#include </builds/inivation/dv/libcaer/include/libcaer/devices/dvs132s.h> DVS132S device-related information.

Public Members

int16_t deviceID

Unique device identifier. Also 'source' for events.

char deviceSerialNumber[8 + 1]

Device serial number.

uint8_t deviceUSBBusNumber

Device USB bus number.

uint8_t deviceUSBDeviceAddress

Device USB device address.

char *deviceString

Device information string, for logging purposes. If not NULL, pointed-to memory is *only* valid while the corresponding device is open! After calling deviceClose() this is invalid memory!

int16_t firmwareVersion

USB firmware version.

int16_t logicVersion

Logic (FPGA/CPLD) version.

int16_t **chipID**

Chip identifier/type.

bool **deviceIsMaster**

Whether the device is a time-stamp master or slave.

bool **muxHasStatistics**

Feature test: Multiplexer statistics support (event drops).

int16_t **dvsSizeX**

DVS X axis resolution.

int16_t **dvsSizeY**

DVS Y axis resolution.

bool **dvsHasStatistics**

Feature test: DVS statistics support.

enum *caer_imu_types* **imuType**

IMU chip type on device.

bool **extInputHasGenerator**

Feature test: External Input module supports Signal-Generation.

struct **caer_dvx_info**

#include </builds/inivation/dv/libcaer/include/libcaer/devices/dvxplorer.h> DVXPLORER device-related information.

Public Members

int16_t **deviceID**

Unique device identifier. Also 'source' for events.

char **deviceSerialNumber**[8 + 1]

Device serial number.

uint8_t **deviceUSBBusNumber**

Device USB bus number.

uint8_t **deviceUSBDeviceAddress**

Device USB device address.

char ***deviceString**

Device information string, for logging purposes. If not NULL, pointed-to memory is *only* valid while the corresponding device is open! After calling deviceClose() this is invalid memory!

int16_t **firmwareVersion**

USB firmware version.

int16_t **logicVersion**

Logic (FPGA/CPLD) version.

int16_t **chipID**

Chip identifier/type.

bool **deviceIsMaster**

Whether the device is a time-stamp master or slave.

bool **muxHasStatistics**

Feature test: Multiplexer statistics support (event drops).

int16_t **dvsSizeX**

DVS X axis resolution.

int16_t **dvsSizeY**

DVS Y axis resolution.

bool **dvsHasStatistics**

Feature test: DVS statistics support.

enum *caer_imu_types* **imuType**

IMU chip type on device.

bool **extInputHasGenerator**

Feature test: External Input module supports Signal-Generation.

struct **caer_dynapse_info**

#include </builds/inivation/dv/libcaer/include/libcaer/devices/dynapse.h> Dynap-se device-related information.

Public Members

int16_t **deviceID**

Unique device identifier. Also 'source' for events.

char **deviceSerialNumber**[8 + 1]

Device serial number.

uint8_t **deviceUSBBusNumber**

Device USB bus number.

uint8_t **deviceUSBDeviceAddress**

Device USB device address.

char ***deviceString**

Device information string, for logging purposes. If not NULL, pointed-to memory is *only* valid while the corresponding device is open! After calling deviceClose() this is invalid memory!

int16_t **logicVersion**

Logic (FPGA/CPLD) version.

bool **deviceIsMaster**

Whether the device is a time-stamp master or slave.

int16_t **logicClock**

Clock in MHz for main logic (FPGA/CPLD).

int16_t **chipID**

Chip identifier/type.

bool **aerHasStatistics**

Feature test: AER (spikes) statistics support.

bool **muxHasStatistics**

Feature test: Multiplexer statistics support (event drops).

struct **caer_edvs_info**

#include </builds/inivation/dv/libcaer/include/libcaer/devices/edvs.h> EDVS device-related information.

Public Members

int16_t **deviceID**

Unique device identifier. Also 'source' for events.

char ***deviceString**

Device information string, for logging purposes. If not NULL, pointed-to memory is *only* valid while the corresponding device is open! After calling deviceClose() this is invalid memory!

bool **deviceIsMaster**

Whether the device is a time-stamp master or slave.

int16_t **dvsSizeX**

DVS X axis resolution.

int16_t **dvsSizeY**

DVS Y axis resolution.

char **serialPortName**[64]

Connected serial port name (OS-specific).

uint32_t **serialBaudRate**

Serial connection baud-rate.

struct **caer_filter_dvs_pixel**

#include </builds/inivation/dv/libcaer/include/libcaer/filters/dvs_noise.h> Structure representing a single DVS pixel address, with X and Y components. Used in DVS filtering support.

Public Members

uint16_t **x**

uint16_t **y**

struct **caer_samsung_evk_info**

#include </builds/inivation/dv/libcaer/include/libcaer/devices/samsung_evk.h> SAMSUNG_EVK device-related information.

Public Members

int16_t **deviceID**

Unique device identifier. Also 'source' for events.

char **deviceSerialNumber**[8 + 1]

Device serial number.

uint8_t **deviceUSBBusNumber**

Device USB bus number.

uint8_t **deviceUSBDeviceAddress**

Device USB device address.

char ***deviceString**

Device information string, for logging purposes. If not NULL, pointed-to memory is *only* valid while the corresponding device is open! After calling deviceClose() this is invalid memory!

int16_t **firmwareVersion**

USB firmware version.

int16_t **chipID**

Chip identifier/type.

int16_t **dvsSizeX**

DVS X axis resolution.

int16_t **dvsSizeY**

DVS Y axis resolution.

class **davis** : public *libcaer*::*devices*::*usb*

Subclassed by *libcaer::devices::davisfx2*, *libcaer::devices::davisfx3*

Public Functions

inline **davis**(uint16_t deviceID)

inline **davis**(uint16_t deviceID, uint8_t busNumberRestrict, uint8_t devAddressRestrict, const std::string &serialNumberRestrict)

inline struct *caer_davis_info* **infoGet**() const noexcept

inline virtual std::string **toString**() const noexcept override

inline void **roiConfigure**(uint16_t startX, uint16_t startY, uint16_t endX, uint16_t endY) const

Public Static Functions

static inline uint16_t **biasVDACGenerate**(const struct *caer_bias_vdac* vdacBias) noexcept

static inline struct *caer_bias_vdac* **biasVDACParse**(const uint16_t vdacBias) noexcept

static inline uint16_t **biasCoarseFineGenerate**(const struct *caer_bias_coarsefine* coarseFineBias) noexcept

static inline struct *caer_bias_coarsefine* **biasCoarseFineParse**(const uint16_t coarseFineBias) noexcept

static inline struct *caer_bias_coarsefine* **biasCoarseFineFromCurrent**(const uint32_t picoAmps) noexcept

static inline uint32_t **biasCoarseFineToCurrent**(const struct *caer_bias_coarsefine* coarseFineBias) noexcept

static inline uint16_t **biasShiftedSourceGenerate**(const struct *caer_bias_shiftedsource* shiftedSourceBias) noexcept

static inline struct *caer_bias_shiftedsource* **biasShiftedSourceParse**(const uint16_t shiftedSourceBias) noexcept

Protected Functions

inline **davis**(uint16_t deviceID, uint16_t deviceType)

inline **davis**(uint16_t deviceID, uint16_t deviceType, uint8_t busNumberRestrict, uint8_t devAddressRestrict, const std::string &serialNumberRestrict)

class **davisfx2** : public *libcaer*::*devices*::*davis*

Public Functions

inline **davisfx2**(uint16_t deviceID)

inline **davisfx2**(uint16_t deviceID, uint8_t busNumberRestrict, uint8_t devAddressRestrict, const std::string &serialNumberRestrict)

class **davisfx3** : public *libcaer*::*devices*::*davis*

Public Functions

inline **davisfx3**(uint16_t deviceID)

inline **davisfx3**(uint16_t deviceID, uint8_t busNumberRestrict, uint8_t devAddressRestrict, const std::string &serialNumberRestrict)

class **device**

Subclassed by *libcaer::devices::serial*, *libcaer::devices::usb*

Public Functions

virtual **~device**() = default

virtual std::string **toString**() const noexcept = 0

inline void **sendDefaultConfig**() const

inline void **configSet**(int8_t modAddr, uint8_t paramAddr, uint32_t param) const

inline void **configGet**(int8_t modAddr, uint8_t paramAddr, uint32_t *param) const

inline uint32_t **configGet**(int8_t modAddr, uint8_t paramAddr) const

inline void **configGet64**(int8_t modAddr, uint8_t paramAddr, uint64_t *param) const

inline uint64_t **configGet64**(int8_t modAddr, uint8_t paramAddr) const

inline void **dataStart**(void (*dataNotifyIncrease)(void *ptr), void (*dataNotifyDecrease)(void *ptr), void *dataNotifyUserPtr, void (*dataShutdownNotify)(void *ptr), void *dataShutdownUserPtr) const

inline void **dataStop**() const

inline std::unique_ptr<*libcaer*::*events*::*EventPacketContainer*> **dataGet**() const

Protected Functions

device() = default

Protected Attributes

std::shared_ptr<struct caer_device_handle> **handle**

class **discover**

Public Static Functions

static inline std::vector<struct *caer_device_discovery_result*> **device**(int16_t deviceType)

static inline std::vector<struct *caer_device_discovery_result*> **all**()

static inline std::unique_ptr<*libcaer*::*devices*::*device*> **open**(uint16_t deviceID, const struct *caer_device_discovery_result* &discoveredDevice)

class **dvs128** : public *libcaer*::*devices*::*usb*

Public Functions

inline **dvs128**(uint16_t deviceID)

inline **dvs128**(uint16_t deviceID, uint8_t busNumberRestrict, uint8_t devAddressRestrict, const std::string &serialNumberRestrict)

inline struct *caer_dvs128_info* **infoGet**() const noexcept

inline virtual std::string **toString**() const noexcept override

class **dvs132s** : public *libcaer*::*devices*::*usb*

Public Functions

inline **dvs132s**(uint16_t deviceID)

inline **dvs132s**(uint16_t deviceID, uint8_t busNumberRestrict, uint8_t devAddressRestrict, const std::string &serialNumberRestrict)

inline struct *caer_dvs132s_info* **infoGet**() const noexcept

inline virtual std::string **toString**() const noexcept override

class **DVSNoise**

Public Functions

inline **DVSNoise**(uint16_t sizeX, uint16_t sizeY)

~DVSNoise() = default

inline std::string **toString**() const noexcept

inline void **configSet**(uint8_t paramAddr, uint64_t param) const

inline void **configGet**(uint8_t paramAddr, uint64_t *param) const

inline uint64_t **configGet**(uint8_t paramAddr) const

inline std::vector<struct *caer_filter_dvs_pixel*> **getHotPixels**() const

inline void **apply**(*caerPolarityEventPacket* polarity) const noexcept

inline void **apply**(*libcaer*::*events*::*PolarityEventPacket* &polarity) const noexcept

inline void **apply**(*libcaer*::*events*::*PolarityEventPacket* *polarity) const noexcept

inline void **apply**(*caerPolarityEventPacketConst* polarity) const noexcept

inline void **apply**(const *libcaer*::*events*::*PolarityEventPacket* &polarity) const noexcept

inline void **apply**(const *libcaer*::*events*::*PolarityEventPacket* *polarity) const noexcept

Private Members

std::shared_ptr<struct caer_filter_dvs_noise> **handle**

class **dvXplorer** : public *libcaer*::*devices*::*usb*

Public Functions

inline **dvXplorer**(uint16_t deviceID)

inline **dvXplorer**(uint16_t deviceID, uint8_t busNumberRestrict, uint8_t devAddressRestrict, const std::string &serialNumberRestrict)

inline struct *caer_dvx_info* **infoGet**() const noexcept

inline virtual std::string **toString**() const noexcept override

class **dynapse** : public *libcaer*::*devices*::*usb*

Public Functions

inline **dynapse**(uint16_t deviceID)

inline **dynapse**(uint16_t deviceID, uint8_t busNumberRestrict, uint8_t devAddressRestrict, const std::string &serialNumberRestrict)

inline struct *caer_dynapse_info* **infoGet**() const noexcept

inline virtual std::string **toString**() const noexcept override

inline void **sendDataToUSB**(const uint32_t *data, size_t numConfig) const

inline void **writeSramWords**(const uint16_t *data, uint32_t baseAddr, size_t numWords) const

inline void **writePoissonSpikeRate**(uint16_t neuronAddr, float rateHz) const

inline void **writeSram**(uint8_t coreId, uint8_t neuronAddrCore, uint8_t virtualCoreId, bool sx, uint8_t dx, bool sy, uint8_t dy, uint8_t sramId, uint8_t destinationCore) const

inline void **writeSramN**(uint16_t neuronAddr, uint8_t sramId, uint8_t virtualCoreId, bool sx, uint8_t dx, bool sy, uint8_t dy, uint8_t destinationCore) const

inline void **writeCam**(uint16_t inputNeuronAddr, uint16_t neuronAddr, uint8_t camId, uint8_t synapseType) const

Public Static Functions

static inline uint32_t **biasDynapseGenerate**(const struct *caer_bias_dynapse* dynapseBias) noexcept

static inline struct *caer_bias_dynapse* **biasDynapseParse**(const uint32_t dynapseBias) noexcept

static inline uint32_t **generateCamBits**(uint16_t inputNeuronAddr, uint16_t neuronAddr, uint8_t camId, uint8_t synapseType) noexcept

static inline uint32_t **generateSramBits**(uint16_t neuronAddr, uint8_t sramId, uint8_t virtualCoreId, bool sx, uint8_t dx, bool sy, uint8_t dy, uint8_t destinationCore) noexcept

static inline uint16_t **coreXYToNeuronId**(uint8_t coreId, uint8_t columnX, uint8_t rowY) noexcept

static inline uint16_t **coreAddrToNeuronId**(uint8_t coreId, uint8_t neuronAddrCore) noexcept

static inline uint16_t **spikeEventGetX**(const *libcaer*::*events*::*SpikeEvent* &event) noexcept

static inline uint16_t **spikeEventGetY**(const *libcaer*::*events*::*SpikeEvent* &event) noexcept

static inline uint16_t **spikeEventGetX**(const *libcaer*::*events*::*SpikeEvent* *event) noexcept

static inline uint16_t **spikeEventGetY**(const *libcaer*::*events*::*SpikeEvent* *event) noexcept

static inline *libcaer*::*events*::*SpikeEvent* **spikeEventFromXY**(uint16_t x, uint16_t y) noexcept

class **edvs** : public *libcaer*::*devices*::*serial*

Public Functions

inline **edvs**(uint16_t deviceID, const std::string &serialPortName, uint32_t serialBaudRate)

inline struct *caer_edvs_info* **infoGet**() const noexcept

inline virtual std::string **toString**() const noexcept override

class **EventPacket**

Subclassed by *libcaer::events::EventPacketCommon< FrameEventPacket, FrameEvent >*, *libcaer::events::EventPacketCommon< IMU6EventPacket, IMU6Event >*, *libcaer::events::EventPacketCommon< IMU9EventPacket, IMU9Event >*, *libcaer::events::EventPacketCommon< PolarityEventPacket, PolarityEvent >*, *libcaer::events::EventPacketCommon< SpecialEventPacket, SpecialEvent >*, *libcaer::events::EventPacketCommon< SpikeEventPacket, SpikeEvent >*, *libcaer::events::EventPacketCommon< PKT, EVT >*

Public Types

enum class **copyTypes**

Values:

enumerator **FULL**

enumerator **EVENTS_ONLY**

enumerator **VALID_EVENTS_ONLY**

using **value_type** = *GenericEvent*

using **const_value_type** = const *GenericEvent*

using **pointer** = *GenericEvent**

using **const_pointer** = const *GenericEvent**

using **reference** = *GenericEvent*&

using **const_reference** = const *GenericEvent*&

using **size_type** = int32_t

using **difference_type** = ptrdiff_t

Public Functions

inline **EventPacket**(*caerEventPacketHeader* packetHeader, bool takeMemoryOwnership = true)

inline virtual **~EventPacket**()

inline **EventPacket**(const *EventPacket* &rhs)

inline *EventPacket* &**operator=**(const *EventPacket* &rhs)

inline **EventPacket**(*EventPacket* &&rhs) noexcept

inline *EventPacket* &**operator=**(*EventPacket* &&rhs)

inline bool **operator==**(const *EventPacket* &rhs) const noexcept

inline bool **operator!=**(const *EventPacket* &rhs) const noexcept

inline int16_t **getEventType**() const noexcept

inline void **setEventType**(int16_t eventType)

inline int16_t **getEventSource**() const noexcept

inline void **setEventSource**(int16_t eventSource)

inline int32_t **getEventSize**() const noexcept

inline void **setEventSize**(int32_t eventSize)

inline int32_t **getEventTSOffset**() const noexcept

inline void **setEventTSOffset**(int32_t eventTSOffset)

inline int32_t **getEventTSOverflow**() const noexcept

inline void **setEventTSOverflow**(int32_t eventTSOverflow)

inline int32_t **getEventCapacity**() const noexcept

inline void **setEventCapacity**(int32_t eventCapacity)

inline int32_t **getEventNumber**() const noexcept

inline void **setEventNumber**(int32_t eventNumber)

inline int32_t **getEventValid**() const noexcept

inline void **setEventValid**(int32_t eventValid)

inline *const_value_type* **genericGetEvent**(*size_type* index) const

inline int64_t **getDataSize**() const noexcept

inline int64_t **getSize**() const noexcept

inline int64_t **getDataSizeEvents**() const noexcept

inline int64_t **getSizeEvents**() const noexcept

inline void **clear**() noexcept

inline void **clean**() noexcept

inline void **resize**(*size_type* newEventCapacity)

inline void **shrink_to_fit**()

inline void **grow**(*size_type* newEventCapacity)

inline void **append**(const *EventPacket* &appendPacket)

inline std::unique_ptr<*EventPacket*> **copy**(*copyTypes* ct) const

inline void **swap**(*EventPacket* &rhs)

inline *caerEventPacketHeader* **getHeaderPointer**() noexcept

inline *caerEventPacketHeaderConst* **getHeaderPointer**() const noexcept

inline bool **isPacketMemoryOwner**() const noexcept

inline *caerEventPacketHeader* **getHeaderPointerForCOutput**() noexcept

inline *size_type* **capacity**() const noexcept

inline *size_type* **size**() const noexcept

inline bool **empty**() const noexcept

Protected Functions

inline **EventPacket**()

inline virtual std::unique_ptr<*EventPacket*> **virtualCopy**(*copyTypes* ct) const

inline *size_type* **getEventIndex**(*size_type* index, bool limitIsCapacity) const

Protected Attributes

caerEventPacketHeader **header**

bool **isMemoryOwner**

Protected Static Functions

static inline *caerEventPacketHeader* **internalCopy**(*caerEventPacketHeaderConst* header, *copyTypes* ct)

static inline void **constructorCheckCapacitySourceTSOverflow**(*size_type* eventCapacity, int16_t eventSource, int32_t tsOverflow)

static inline void **constructorCheckNullptr**(const void *packet)

static inline void **constructorCheckEventType**(*caerEventPacketHeaderConst* packet, int16_t type)

template<class **PKT**, class **EVT**>

class **EventPacketCommon** : public *libcaer*::*events*::*EventPacket*

Public Types

using **value_type** = *EVT*

using **const_value_type** = const *EVT*

using **pointer** = *EVT**

using **const_pointer** = const *EVT**

using **reference** = *EVT*&

using **const_reference** = const *EVT*&

using **size_type** = int32_t

using **difference_type** = ptrdiff_t

using **iterator** = *EventPacketIterator*<*value_type*>

using **const_iterator** = *EventPacketIterator*<*const_value_type*>

using **reverse_iterator** = std::reverse_iterator<*iterator*>

using **const_reverse_iterator** = std::reverse_iterator<*const_iterator*>

Public Functions

inline *reference* **getEvent**(*size_type* index)

inline *const_reference* **getEvent**(*size_type* index) const

inline *reference* **operator[]**(*size_type* index)

inline *const_reference* **operator[]**(*size_type* index) const

inline *reference* **front**()

inline *const_reference* **front**() const

inline *reference* **back**()

inline *const_reference* **back**() const

inline std::unique_ptr<*PKT*> **copy**(copyTypes ct) const

inline *iterator* **begin**() noexcept

inline *iterator* **end**() noexcept

inline *const_iterator* **begin**() const noexcept

inline *const_iterator* **end**() const noexcept

inline *const_iterator* **cbegin**() const noexcept

inline *const_iterator* **cend**() const noexcept

inline *reverse_iterator* **rbegin**() noexcept

inline *reverse_iterator* **rend**() noexcept

inline *const_reverse_iterator* **rbegin**() const noexcept

inline *const_reverse_iterator* **rend**() const noexcept

inline *const_reverse_iterator* **crbegin**() const noexcept

inline *const_reverse_iterator* **crend**() const noexcept

Protected Functions

inline virtual std::unique_ptr<*EventPacket*> **virtualCopy**(copyTypes ct) const override

virtual *reference* **virtualGetEvent**(*size_type* index) noexcept = 0

virtual *const_reference* **virtualGetEvent**(*size_type* index) const noexcept = 0

class **EventPacketContainer**

Public Types

using **value_type** = std::shared_ptr<*EventPacket*>

using **const_value_type** = std::shared_ptr<const *EventPacket*>

using **size_type** = int32_t

using **difference_type** = ptrdiff_t

using **iterator** = *EventPacketContainerCopyIterator*<std::vector<std::shared_ptr<*EventPacket*>>::iterator, std::shared_ptr<*EventPacket*>>

using **const_iterator** = *EventPacketContainerCopyIterator*<std::vector<std::shared_ptr<*EventPacket*>>::const_iterator, std::shared_ptr<const *EventPacket*>>

using **reverse_iterator** = std::reverse_iterator<*iterator*>

using **const_reverse_iterator** = std::reverse_iterator<*const_iterator*>

Public Functions

inline **EventPacketContainer**()

Construct a new *EventPacketContainer*.

inline **EventPacketContainer**(*size_type* eventPacketsNumber)

Construct a new *EventPacketContainer* with enough space to store up to the given number of event packet pointers. The pointers are present and initialized to nullptr.

Parameters

eventPacketsNumber – the initial number of event packet pointers that can be stored in this container. Must be equal to one or higher.

inline **EventPacketContainer**(*caerEventPacketContainer* packetContainer, bool takeMemoryOwnership = true)

Construct a new *EventPacketContainer* from a C-style caerEventPacketContainer. The contained packets can take over memory ownership if so requested.

Parameters

- **packetContainer** – C-style caerEventPacketContainer from which to initialize the new packet container.
- **takeMemoryOwnership** – true if the container packets shall take over the ownership of the memory containing the events from the C-style packets.

inline *size_type* **capacity**() const noexcept

inline *size_type* **size**() const noexcept

inline bool **empty**() const noexcept

inline void **clear**() noexcept

inline *value_type* **getEventPacket**(*size_type* index)

Get the pointer to the event packet stored in this container at the given index.

Parameters

index – the index of the event packet to get.

Throws

std:out_of_range – no packet exists at given index.

Returns

a pointer to an event packet.

inline *value_type* **operator[]**(*size_type* index)

inline *const_value_type* **getEventPacket**(*size_type* index) const

Get the pointer to the event packet stored in this container at the given index. This is a read-only event packet, do not change its contents in any way!

Parameters
index – the index of the event packet to get.

Throws
std:out_of_range – no packet exists at given index.

Returns
a pointer to a read-only event packet.

inline *const_value_type* **operator[]**(*size_type* index) const

inline void **setEventPacket**(*size_type* index, *value_type* packetHeader)

Set the pointer to the event packet stored in this container at the given index. The index must be valid already, this does not change the container size.

Parameters

- **index** – the index of the event packet to set.
- **packetHeader** – a pointer to an event packet. Can be a nullptr.

Throws
std:out_of_range – no packet exists at given index.

inline void **addEventPacket**(*value_type* packetHeader)

Add an event packet pointer at the end of this container. Increases container size by one.

Parameters
packetHeader – a pointer to an event packet. Can be a nullptr.

inline int64_t **getLowestEventTimestamp**() const noexcept

Get the lowest timestamp contained in this event packet container.

Returns
the lowest timestamp (in µs) or -1 if not initialized.

inline int64_t **getHighestEventTimestamp**() const noexcept

Get the highest timestamp contained in this event packet container.

Returns
the highest timestamp (in µs) or -1 if not initialized.

inline int32_t **getEventsNumber**() const noexcept

Get the number of events contained in this event packet container.

Returns
the number of events in this container.

inline int32_t **getEventsValidNumber**() const noexcept

Get the number of valid events contained in this event packet container.

Returns
the number of valid events in this container.

inline void **updateStatistics**() noexcept

Recalculates and updates all the packet-container level statistics (event counts and timestamps).

inline *value_type* **findEventPacketByType**(int16_t typeID)

Get the pointer to an event packet stored in this container with the given event type. This returns the first found event packet with that type ID, or nullptr if we get to the end without finding any such event packet.

Parameters
typeID – the event type to search for.

Returns
a pointer to an event packet with a certain type or nullptr if none found.

inline std::unique_ptr<std::vector<*value_type*>> **findEventPacketsByType**(int16_t typeID)

inline *const_value_type* **findEventPacketByType**(int16_t typeID) const

Get the pointer to a read-only event packet stored in this container with the given event type. This returns the first found event packet with that type ID, or nullptr if we get to the end without finding any such event packet.

Parameters
typeID – the event type to search for.

Returns
a pointer to a read-only event packet with a certain type or nullptr if none found.

inline std::unique_ptr<std::vector<*const_value_type*>> **findEventPacketsByType**(int16_t typeID) const

inline *value_type* **findEventPacketBySource**(int16_t sourceID)

Get the pointer to an event packet stored in this container from the given event source. This returns the first found event packet with that source ID, or nullptr if we get to the end without finding any such event packet.

Parameters
sourceID – the event source to search for.

Returns
a pointer to an event packet with a certain source or nullptr if none found.

inline std::unique_ptr<std::vector<*value_type*>> **findEventPacketsBySource**(int16_t sourceID)

inline *const_value_type* **findEventPacketBySource**(int16_t sourceID) const

Get the pointer to a read-only event packet stored in this container from the given event source. This returns the first found event packet with that source ID, or nullptr if we get to the end without finding any such event packet.

Parameters
sourceID – the event source to search for.

Returns
a pointer to a read-only event packet with a certain source or nullptr if none found.

inline std::unique_ptr<std::vector<*const_value_type*>> **findEventPacketsBySource**(int16_t sourceID) const

inline std::unique_ptr<*EventPacketContainer*> **copyAllEvents**() const

Make a deep copy of this event packet container and all of its event packets and their current events. A normal copy (using copy constructor or assignment) copies all the container's internals, and correctly handles the pointers to the event packets held in it, but those will still point to the old event packets. To also copy the individual event packets and point to the new copies, use this function.

Returns
a deep copy of this event packet container, containing all events.

inline std::unique_ptr<*EventPacketContainer*> **copyValidEvents**() const

Make a deep copy of this event packet container, with its event packets sized down to only include the currently valid events (eventValid), and discarding everything else. A normal copy (using copy constructor or assignment) copies all the container's internals, and correctly handles the pointers to the event packets held in it, but those will still point to the old event packets. To also copy the individual event packets and point to the new copies, use this function.

Returns
a deep copy of this event packet container, containing only valid events.

inline *iterator* **begin**() noexcept

inline *iterator* **end**() noexcept

inline *const_iterator* **begin**() const noexcept

inline *const_iterator* **end**() const noexcept

inline *const_iterator* **cbegin**() const noexcept

inline *const_iterator* **cend**() const noexcept

inline *reverse_iterator* **rbegin**() noexcept

inline *reverse_iterator* **rend**() noexcept

inline *const_reverse_iterator* **rbegin**() const noexcept

inline *const_reverse_iterator* **rend**() const noexcept

inline *const_reverse_iterator* **crbegin**() const noexcept

inline *const_reverse_iterator* **crend**() const noexcept

Private Members

int64_t **lowestEventTimestamp**

Smallest event timestamp contained in this packet container.

int64_t **highestEventTimestamp**

Largest event timestamp contained in this packet container.

int32_t **eventsNumber**

Number of events contained within all the packets in this container.

int32_t **eventsValidNumber**

Number of valid events contained within all the packets in this container.

std::vector<std::shared_ptr<*EventPacket*>> **eventPackets**

Vector of pointers to the actual event packets.

template<class **InteralIterator**, class **SharedPtrType**>

class **EventPacketContainerCopyIterator**

Public Types

using **iterator_category** = typename *InteralIterator*::iterator_category

using **value_type** = *SharedPtrType*

using **pointer** = const *SharedPtrType**

using **reference** = const *SharedPtrType*&

using **difference_type** = typename *InteralIterator*::difference_type

using **size_type** = typename *InteralIterator*::difference_type

Public Functions

inline **EventPacketContainerCopyIterator**()

inline **EventPacketContainerCopyIterator**(*InteralIterator* _eventPacketsIterator)

inline *reference* **operator***() const noexcept

inline *pointer* **operator->**() const noexcept

inline *reference* **operator[]**(*size_type* idx) const noexcept

inline bool **operator==**(const *EventPacketContainerCopyIterator* &rhs) const noexcept

inline bool **operator!=**(const *EventPacketContainerCopyIterator* &rhs) const noexcept

inline bool **operator<**(const *EventPacketContainerCopyIterator* &rhs) const noexcept

inline bool **operator>**(const *EventPacketContainerCopyIterator* &rhs) const noexcept

inline bool **operator<=**(const *EventPacketContainerCopyIterator* &rhs) const noexcept

inline bool **operator>=**(const *EventPacketContainerCopyIterator* &rhs) const noexcept

inline *EventPacketContainerCopyIterator* &**operator++**() noexcept

inline *EventPacketContainerCopyIterator* **operator++**(int) noexcept

inline *EventPacketContainerCopyIterator* &**operator--**() noexcept

inline *EventPacketContainerCopyIterator* **operator--**(int) noexcept

inline *EventPacketContainerCopyIterator* &**operator+=**(*size_type* add) noexcept

inline *EventPacketContainerCopyIterator* **operator+**(*size_type* add) const noexcept

inline *EventPacketContainerCopyIterator* &**operator-=**(*size_type* sub) noexcept

inline *EventPacketContainerCopyIterator* **operator-**(*size_type* sub) const noexcept

inline *difference_type* **operator-**(const *EventPacketContainerCopyIterator* &rhs) const noexcept

inline void **swap**(*EventPacketContainerCopyIterator* &rhs) noexcept

Private Members

InteralIterator **eventPacketsIterator**

mutable *SharedPtrType* **currElement**

Friends

inline friend *EventPacketContainerCopyIterator* **operator+**(*size_type* lhs, const *EventPacketContainerCopyIterator* &rhs) noexcept

template<class **T**>

class **EventPacketIterator**

Public Types

using **iterator_category** = std::random_access_iterator_tag

using **value_type** = typename std::remove_cv<*T*>::type

using **pointer** = *T**

using **reference** = *T*&

using **difference_type** = ptrdiff_t

using **size_type** = int32_t

Public Functions

inline **EventPacketIterator**()

inline **EventPacketIterator**(*eventPtrType* _eventPtr, size_t _eventSize)

inline *reference* **operator***() const noexcept

inline *pointer* **operator->**() const noexcept

inline *reference* **operator[]**(*size_type* index) const noexcept

inline bool **operator==**(const *EventPacketIterator* &rhs) const noexcept

inline bool **operator!=**(const *EventPacketIterator* &rhs) const noexcept

inline bool **operator<**(const *EventPacketIterator* &rhs) const noexcept

inline bool **operator>**(const *EventPacketIterator* &rhs) const noexcept

inline bool **operator<=**(const *EventPacketIterator* &rhs) const noexcept

inline bool **operator>=**(const *EventPacketIterator* &rhs) const noexcept

inline *EventPacketIterator* &**operator++**() noexcept

inline *EventPacketIterator* **operator++**(int) noexcept

inline *EventPacketIterator* &**operator--**() noexcept

inline *EventPacketIterator* **operator--**(int) noexcept

inline *EventPacketIterator* &**operator+=**(*size_type* add) noexcept

inline *EventPacketIterator* **operator+**(*size_type* add) const noexcept

inline *EventPacketIterator* &**operator-=**(*size_type* sub) noexcept

inline *EventPacketIterator* **operator-**(*size_type* sub) const noexcept

inline *difference_type* **operator-**(const *EventPacketIterator* &rhs) const noexcept

inline void **swap**(*EventPacketIterator* &rhs) noexcept

Private Types

using **eventPtrType** = typename std::conditional<std::is_const<*T*>::value, const uint8_t*, uint8_t*>::type

Private Members

eventPtrType **eventPtr**

size_t **eventSize**

Friends

inline friend *EventPacketIterator* **operator+**(*size_type* lhs, const *EventPacketIterator* &rhs) noexcept

struct **FrameEvent** : public caer_frame_event

#include </builds/inivation/dv/libcaer/include/libcaercpp/events/frame.hpp> Assignment/Move Constructors/Operators cannot be used with Frame Events due to their particular memory layout that is not entirely known to the compiler (dynamic pixel array). As such those constructors and operators are disabled. Please use caerGenericEventCopy() to copy frame events!

Public Types

enum class **colorChannels**

Values:

enumerator **GRAYSCALE**

Grayscale, one channel only.

enumerator **RGB**

Red Green Blue, 3 color channels.

enumerator **RGBA**

Red Green Blue Alpha, 3 color channels plus transparency.

enum class **colorFilter**

Values:

enumerator **MONO**

No color filter present, all light passes.

enumerator **RGBG**

Standard Bayer color filter, 1 red 2 green 1 blue. Variation 1.

enumerator **GRGB**

Standard Bayer color filter, 1 red 2 green 1 blue. Variation 2.

enumerator **GBGR**

Standard Bayer color filter, 1 red 2 green 1 blue. Variation 3.

enumerator **BGRG**

Standard Bayer color filter, 1 red 2 green 1 blue. Variation 4.

enumerator **RGBW**

Modified Bayer color filter, with white (pass all light) instead of extra green. Variation 1.

enumerator **GRWB**

Modified Bayer color filter, with white (pass all light) instead of extra green. Variation 2.

enumerator **WBGR**

Modified Bayer color filter, with white (pass all light) instead of extra green. Variation 3.

enumerator **BWRG**

Modified Bayer color filter, with white (pass all light) instead of extra green. Variation 4.

Public Functions

FrameEvent() = default

FrameEvent(const *FrameEvent* &rhs) = delete

FrameEvent &**operator=**(const *FrameEvent* &rhs) = delete

FrameEvent(*FrameEvent* &&rhs) = delete

FrameEvent &**operator=**(*FrameEvent* &&rhs) = delete

inline int32_t **getTSStartOfFrame**() const noexcept

inline int64_t **getTSStartOfFrame64**(const *EventPacket* &packet) const noexcept

inline void **setTSStartOfFrame**(int32_t ts)

inline int32_t **getTSEndOfFrame**() const noexcept

inline int64_t **getTSEndOfFrame64**(const *EventPacket* &packet) const noexcept

inline void **setTSEndOfFrame**(int32_t ts)

inline int32_t **getTSStartOfExposure**() const noexcept

inline int64_t **getTSStartOfExposure64**(const *EventPacket* &packet) const noexcept

inline void **setTSStartOfExposure**(int32_t ts)

inline int32_t **getTSEndOfExposure**() const noexcept

inline int64_t **getTSEndOfExposure64**(const *EventPacket* &packet) const noexcept

inline void **setTSEndOfExposure**(int32_t ts)

inline int32_t **getTimestamp**() const noexcept

inline int64_t **getTimestamp64**(const *EventPacket* &packet) const noexcept

inline int32_t **getExposureLength**() const noexcept

inline bool **isValid**() const noexcept

inline void **validate**(*EventPacket* &packet) noexcept

inline void **invalidate**(*EventPacket* &packet) noexcept

inline uint8_t **getROIIdentifier**() const noexcept

inline void **setROIIdentifier**(uint8_t roiIdent) noexcept

inline *colorFilter* **getColorFilter**() const noexcept

inline void **setColorFilter**(*colorFilter* cFilter) noexcept

inline int32_t **getLengthX**() const noexcept

inline int32_t **getLengthY**() const noexcept

inline *colorChannels* **getChannelNumber**() const noexcept

inline void **setLengthXLengthYChannelNumber**(int32_t lenX, int32_t lenY, *colorChannels* cNumber, const *EventPacket* &packet)

inline size_t **getPixelsMaxIndex**() const noexcept

inline size_t **getPixelsSize**() const noexcept

inline int32_t **getPositionX**() const noexcept

inline void **setPositionX**(int32_t posX) noexcept

inline int32_t **getPositionY**() const noexcept

inline void **setPositionY**(int32_t posY) noexcept

inline uint16_t **getPixel**(int32_t xAddress, int32_t yAddress) const

inline void **setPixel**(int32_t xAddress, int32_t yAddress, uint16_t pixelValue)

inline uint16_t **getPixel**(int32_t xAddress, int32_t yAddress, uint8_t channel) const

inline void **setPixel**(int32_t xAddress, int32_t yAddress, uint8_t channel, uint16_t pixelValue)

inline uint16_t **getPixelUnsafe**(int32_t xAddress, int32_t yAddress) const noexcept

inline void **setPixelUnsafe**(int32_t xAddress, int32_t yAddress, uint16_t pixelValue) noexcept

inline uint16_t **getPixelUnsafe**(int32_t xAddress, int32_t yAddress, uint8_t channel) const noexcept

inline void **setPixelUnsafe**(int32_t xAddress, int32_t yAddress, uint8_t channel, uint16_t pixelValue) noexcept

inline uint16_t ***getPixelArrayUnsafe**() noexcept

inline const uint16_t ***getPixelArrayUnsafe**() const noexcept

class **FrameEventPacket** : public *libcaer*::*events*::*EventPacketCommon*<*FrameEventPacket*, *FrameEvent*>

Public Types

enum class **demosaicTypes**

Values:

enumerator **STANDARD**

enumerator **TO_GRAY**

enum class **contrastTypes**

Values:

enumerator **STANDARD**

Public Functions

inline **FrameEventPacket**(size_type eventCapacity, int16_t eventSource, int32_t tsOverflow, int32_t maxLengthX, int32_t maxLengthY, int16_t maxChannelNumber)

inline **FrameEventPacket**(size_type eventCapacity, int16_t eventSource, int32_t tsOverflow, int32_t maxNumPixels, int16_t maxChannelNumber)

inline **FrameEventPacket**(*caerFrameEventPacket* packet, bool takeMemoryOwnership = true)

inline **FrameEventPacket**(*caerEventPacketHeader* packetHeader, bool takeMemoryOwnership = true)

inline size_t **getPixelsSize**() const noexcept

inline size_t **getPixelsMaxIndex**() const noexcept

inline std::unique_ptr<*FrameEventPacket*> **demosaic**(*demosaicTypes* demosaicType) const

inline void **contrast**(*contrastTypes* contrastType) noexcept

Protected Functions

inline virtual reference **virtualGetEvent**(size_type index) noexcept override

inline virtual const_reference **virtualGetEvent**(size_type index) const noexcept override

struct **GenericEvent**

Public Functions

inline int32_t **getTimestamp**() const noexcept

inline int64_t **getTimestamp64**() const noexcept

inline bool **isValid**() const noexcept

inline void **copy**(void *eventPtrDestination, *caerEventPacketHeaderConst* headerPtrDestination) const

inline void **copy**(struct *GenericEvent* &destinationEvent) const

inline void **copy**(struct *GenericEvent* *destinationEvent) const

Public Members

const void ***event**

caerEventPacketHeaderConst **header**

struct **IMU6Event** : public caer_imu6_event

Public Functions

inline int32_t **getTimestamp**() const noexcept

inline int64_t **getTimestamp64**(const *EventPacket* &packet) const noexcept

inline void **setTimestamp**(int32_t ts)

inline bool **isValid**() const noexcept

inline void **validate**(*EventPacket* &packet) noexcept

inline void **invalidate**(*EventPacket* &packet) noexcept

inline float **getAccelX**() const noexcept

inline void **setAccelX**(float accelX) noexcept

inline float **getAccelY**() const noexcept

inline void **setAccelY**(float accelY) noexcept

inline float **getAccelZ**() const noexcept

inline void **setAccelZ**(float accelZ) noexcept

inline float **getGyroX**() const noexcept

inline void **setGyroX**(float gyroX) noexcept

inline float **getGyroY**() const noexcept

inline void **setGyroY**(float gyroY) noexcept

inline float **getGyroZ**() const noexcept

inline void **setGyroZ**(float gyroZ) noexcept

inline float **getTemp**() const noexcept

inline void **setTemp**(float t) noexcept

class **IMU6EventPacket** : public *libcaer*::*events*::*EventPacketCommon*<*IMU6EventPacket*, *IMU6Event*>

Public Functions

inline **IMU6EventPacket**(size_type eventCapacity, int16_t eventSource, int32_t tsOverflow)

inline **IMU6EventPacket**(*caerIMU6EventPacket* packet, bool takeMemoryOwnership = true)

inline **IMU6EventPacket**(*caerEventPacketHeader* packetHeader, bool takeMemoryOwnership = true)

Protected Functions

inline virtual reference **virtualGetEvent**(size_type index) noexcept override

inline virtual const_reference **virtualGetEvent**(size_type index) const noexcept override

struct **IMU9Event** : public caer_imu9_event

Public Functions

inline int32_t **getTimestamp**() const noexcept

inline int64_t **getTimestamp64**(const *EventPacket* &packet) const noexcept

inline void **setTimestamp**(int32_t ts)

inline bool **isValid**() const noexcept

inline void **validate**(*EventPacket* &packet) noexcept

inline void **invalidate**(*EventPacket* &packet) noexcept

inline float **getAccelX**() const noexcept

inline void **setAccelX**(float accelX) noexcept

inline float **getAccelY**() const noexcept

inline void **setAccelY**(float accelY) noexcept

inline float **getAccelZ**() const noexcept

inline void **setAccelZ**(float accelZ) noexcept

inline float **getGyroX**() const noexcept

inline void **setGyroX**(float gyroX) noexcept

inline float **getGyroY**() const noexcept

inline void **setGyroY**(float gyroY) noexcept

inline float **getGyroZ**() const noexcept

inline void **setGyroZ**(float gyroZ) noexcept

inline float **getTemp**() const noexcept

inline void **setTemp**(float t) noexcept

inline float **getCompX**() const noexcept

inline void **setCompX**(float compX) noexcept

inline float **getCompY**() const noexcept

inline void **setCompY**(float compY) noexcept

inline float **getCompZ**() const noexcept

inline void **setCompZ**(float compZ) noexcept

class **IMU9EventPacket** : public *libcaer*::*events*::*EventPacketCommon*<*IMU9EventPacket*, *IMU9Event*>

Public Functions

inline **IMU9EventPacket**(size_type eventCapacity, int16_t eventSource, int32_t tsOverflow)

inline **IMU9EventPacket**(*caerIMU9EventPacket* packet, bool takeMemoryOwnership = true)

inline **IMU9EventPacket**(*caerEventPacketHeader* packetHeader, bool takeMemoryOwnership = true)

Protected Functions

inline virtual reference **virtualGetEvent**(size_type index) noexcept override

inline virtual const_reference **virtualGetEvent**(size_type index) const noexcept override

struct **PolarityEvent** : public caer_polarity_event

Public Functions

inline int32_t **getTimestamp**() const noexcept

inline int64_t **getTimestamp64**(const *EventPacket* &packet) const noexcept

inline void **setTimestamp**(int32_t ts)

inline bool **isValid**() const noexcept

inline void **validate**(*EventPacket* &packet) noexcept

inline void **invalidate**(*EventPacket* &packet) noexcept

inline bool **getPolarity**() const noexcept

inline void **setPolarity**(bool pol) noexcept

inline uint16_t **getY**() const noexcept

inline void **setY**(uint16_t y) noexcept

inline uint16_t **getX**() const noexcept

inline void **setX**(uint16_t x) noexcept

class **PolarityEventPacket** : public *libcaer*::*events*::*EventPacketCommon*<*PolarityEventPacket*, *PolarityEvent*>

Public Functions

inline **PolarityEventPacket**(size_type eventCapacity, int16_t eventSource, int32_t tsOverflow)

inline **PolarityEventPacket**(*caerPolarityEventPacket* packet, bool takeMemoryOwnership = true)

inline **PolarityEventPacket**(*caerEventPacketHeader* packetHeader, bool takeMemoryOwnership = true)

Protected Functions

inline virtual reference **virtualGetEvent**(size_type index) noexcept override

inline virtual const_reference **virtualGetEvent**(size_type index) const noexcept override

template<typename **T**>

class **RingBuffer**

Public Functions

inline **RingBuffer**(size_t sz)

inline bool **operator==**(const *RingBuffer* &rhs) const noexcept

inline bool **operator!=**(const *RingBuffer* &rhs) const noexcept

inline void **put**(const *T* &elem)

inline bool **full**() const noexcept

inline *T* **get**()

inline *T* **look**() const

inline bool **empty**() const noexcept

Private Members

size_t **putPos**

size_t **getPos**

std::vector<std::atomic<*T*>> **elements**

const size_t **sizeAdj**

const *T* **placeholder**

class **samsungEVK** : public *libcaer*::*devices*::*usb*

Public Functions

inline **samsungEVK**(uint16_t deviceID)

inline **samsungEVK**(uint16_t deviceID, uint8_t busNumberRestrict, uint8_t devAddressRestrict, const std::string &serialNumberRestrict)

inline struct *caer_samsung_evk_info* **infoGet**() const noexcept

inline virtual std::string **toString**() const noexcept override

class **serial** : public *libcaer*::*devices*::*device*

Subclassed by *libcaer::devices::edvs*

Protected Functions

inline **serial**(uint16_t deviceID, uint16_t deviceType, const std::string &serialPortName, uint32_t serialBaudRate)

struct **SpecialEvent** : public caer_special_event

Public Functions

inline int32_t **getTimestamp**() const noexcept

inline int64_t **getTimestamp64**(const *EventPacket* &packet) const noexcept

inline void **setTimestamp**(int32_t ts)

inline bool **isValid**() const noexcept

inline void **validate**(*EventPacket* &packet) noexcept

inline void **invalidate**(*EventPacket* &packet) noexcept

inline uint8_t **getType**() const noexcept

inline void **setType**(uint8_t t) noexcept

inline uint32_t **getData**() const noexcept

inline void **setData**(uint32_t d) noexcept

class **SpecialEventPacket** : public *libcaer*::*events*::*EventPacketCommon*<*SpecialEventPacket*, *SpecialEvent*>

Public Functions

inline **SpecialEventPacket**(size_type eventCapacity, int16_t eventSource, int32_t tsOverflow)

inline **SpecialEventPacket**(*caerSpecialEventPacket* packet, bool takeMemoryOwnership = true)

inline **SpecialEventPacket**(*caerEventPacketHeader* packetHeader, bool takeMemoryOwnership = true)

inline reference **findEventByType**(uint8_t type)

inline const_reference **findEventByType**(uint8_t type) const

inline reference **findValidEventByType**(uint8_t type)

inline const_reference **findValidEventByType**(uint8_t type) const

Protected Functions

inline virtual reference **virtualGetEvent**(size_type index) noexcept override

inline virtual const_reference **virtualGetEvent**(size_type index) const noexcept override

struct **SpikeEvent** : public caer_spike_event

Public Functions

inline int32_t **getTimestamp**() const noexcept

inline int64_t **getTimestamp64**(const *EventPacket* &packet) const noexcept

inline void **setTimestamp**(int32_t ts)

inline bool **isValid**() const noexcept

inline void **validate**(*EventPacket* &packet) noexcept

inline void **invalidate**(*EventPacket* &packet) noexcept

inline uint8_t **getSourceCoreID**() const noexcept

inline void **setSourceCoreID**(uint8_t coreID) noexcept

inline uint8_t **getChipID**() const noexcept

inline void **setChipID**(uint8_t chipID) noexcept

inline uint32_t **getNeuronID**() const noexcept

inline void **setNeuronID**(uint32_t neuronID) noexcept

class **SpikeEventPacket** : public *libcaer*::*events*::*EventPacketCommon*<*SpikeEventPacket*, *SpikeEvent*>

Public Functions

inline **SpikeEventPacket**(size_type eventCapacity, int16_t eventSource, int32_t tsOverflow)

inline **SpikeEventPacket**(*caerSpikeEventPacket* packet, bool takeMemoryOwnership = true)

inline **SpikeEventPacket**(*caerEventPacketHeader* packetHeader, bool takeMemoryOwnership = true)

Protected Functions

inline virtual reference **virtualGetEvent**(size_type index) noexcept override

inline virtual const_reference **virtualGetEvent**(size_type index) const noexcept override

class **usb** : public *libcaer*::*devices*::*device*

Subclassed by *libcaer::devices::davis*, *libcaer::devices::dvXplorer*, *libcaer::devices::dvs128*, *libcaer::devices::dvs132s*, *libcaer::devices::dynapse*, *libcaer::devices::samsungEVK*

Protected Functions

inline **usb**(uint16_t deviceID, uint16_t deviceType)

inline **usb**(uint16_t deviceID, uint16_t deviceType, uint8_t busNumberRestrict, uint8_t devAddressRestrict, const std::string &serialNumberRestrict)

namespace **libcaer**

namespace **devices**

namespace **events**

namespace **utils**

Functions

inline std::unique_ptr<*EventPacket*> **makeUniqueFromCStruct**(*caerEventPacketHeader* packet, bool takeMemoryOwnership)

inline std::shared_ptr<*EventPacket*> **makeSharedFromCStruct**(*caerEventPacketHeader* packet, bool takeMemoryOwnership)

inline enum *caer_frame_utils_pixel_color* **getPixelColor**(*libcaer*::*events*::*FrameEvent*::*colorFilter* cFilter, int32_t x, int32_t y)

inline enum *caer_frame_utils_pixel_color* **getPixelColor**(enum *caer_frame_event_color_filter* cFilter, int32_t x, int32_t y)

namespace **filters**

namespace **log**

Enums

enum class **logLevel**

Values:

enumerator **EMERGENCY**

enumerator **ALERT**

enumerator **CRITICAL**

enumerator **ERROR**

enumerator **WARNING**

enumerator **NOTICE**

enumerator **INFO**

enumerator **DEBUG**

Functions

inline void **logLevelSet**(*logLevel* l) noexcept

inline *logLevel* **logLevelGet**() noexcept

inline void **callbackSet**(*caerLogCallback* callback) noexcept

inline *caerLogCallback* **callbackGet**() noexcept

inline void **fileDescriptorsSet**(int fd1, int fd2) noexcept

inline int **fileDescriptorsGetFirst**() noexcept

inline int **fileDescriptorsGetSecond**() noexcept

inline void **log**(*logLevel* l, const char *subSystem, const char *format, ...) noexcept

inline void **logVA**(*logLevel* l, const char *subSystem, const char *format, va_list args) noexcept

inline void **logVAFull**(uint8_t systemLogLevel, *logLevel* l, const char *subSystem, const char *format, va_list args) noexcept

inline void **disable**(bool disableLogging) noexcept

inline bool **disabled**() noexcept

namespace **network**

namespace **ringbuffer**

file **davis.h**

#include "../events/frame.h" *#include* "../events/imu6.h" *#include* "../events/polarity.h" *#include* "../events/special.h" *#include* "imu_support.h" *#include* "usb.h" DAVIS specific configuration defines and information structures.

Defines

CAER_DEVICE_DAVIS_FX2

Device type definition for iniVation DAVIS FX2-based boards, like DAVIS240a/b/c. Deprecated in favor of CAER_DEVICE_DAVIS.

CAER_DEVICE_DAVIS_FX3

Device type definition for iniVation DAVIS FX3-based boards, like DAVIS346. Deprecated in favor of CAER_DEVICE_DAVIS.

CAER_DEVICE_DAVIS

Device type definition for iniVation DAVIS boards, supporting both FX2 and FX3 generation devices. This is the preferred way to access cameras now.

CAER_DEVICE_DAVIS_RPI

Device type definition for iniVation Raspberry Pi-based DAVIS boards. REMOVED, OBSOLETE!

DAVIS_CHIP_DAVIS240A

DAVIS240A chip identifier. 240x180, no color, no global shutter.

DAVIS_CHIP_DAVIS240B

DAVIS240B chip identifier. 240x180, no color, 50 test columns left-side.

DAVIS_CHIP_DAVIS240C

DAVIS240C chip identifier. 240x180, no color.

DAVIS_CHIP_DAVIS128

DAVIS128 chip identifier. 128x128, color possible, internal ADC.

DAVIS_CHIP_DAVIS346A

DAVIS346A chip identifier. 346x260, color possible, internal ADC.

DAVIS_CHIP_DAVIS346B

DAVIS346B chip identifier. 346x260, color possible, internal ADC.

DAVIS_CHIP_DAVIS640

DAVIS640 chip identifier. 640x480, color possible, internal ADC.

DAVIS_CHIP_DAVIS640H

DAVIS640H chip identifier. 640x480 APS, 320x240 DVS, color possible, internal ADC.

DAVIS_CHIP_DAVIS208

DAVIS208 chip identifier. 208x192, special sensitive test pixels, color possible, internal ADC.

DAVIS_CHIP_DAVIS346C

DAVIS346C chip identifier. 346x260, BSI, color possible, internal ADC.

IS_DAVIS128(chipID)

Macros to check a chip identifier integer against the known chip types. Returns true if a chip identifier matches, false otherwise.

IS_DAVIS208(chipID)

IS_DAVIS240A(chipID)

IS_DAVIS240B(chipID)

IS_DAVIS240C(chipID)

IS_DAVIS240(chipID)

IS_DAVIS346A(chipID)

IS_DAVIS346B(chipID)

IS_DAVIS346C(chipID)

IS_DAVIS346(chipID)

IS_DAVIS640(chipID)

IS_DAVIS640H(chipID)

DAVIS_CONFIG_MUX

Module address: device-side Multiplexer configuration. The Multiplexer is responsible for mixing, timestamping and outputting (via USB) the various event types generated by the device. It is also responsible for timestamp generation and synchronization.

DAVIS_CONFIG_DVS

Module address: device-side DVS configuration. The DVS state machine handshakes with the chip's AER bus and gets the polarity events from it. It supports various configurable delays, as well as advanced filtering capabilities on the polarity events.

DAVIS_CONFIG_APS

Module address: device-side APS (Frame) configuration. The APS (Active-Pixel-Sensor) is responsible for getting the normal, synchronous frame from the camera chip. It supports various options for very precise timing control, as well as Region of Interest imaging.

DAVIS_CONFIG_IMU

Module address: device-side IMU (Inertial Measurement Unit) configuration. The IMU module connects to the external IMU chip and sends data on the device's movement in space. It can configure various options on the external chip, such as accelerometer range or gyroscope refresh rate.

DAVIS_CONFIG_EXTINPUT

Module address: device-side External Input (signal detector/generator) configuration. The External Input module is used to detect external signals on the external input jack and inject an event into the event stream when this happens. It can detect pulses of a specific length or rising and falling edges. On some systems, a signal generator module is also present, which can generate PWM-like pulsed signals with configurable timing.

DAVIS_CONFIG_BIAS

Module address: device-side chip bias configuration. Shared with DAVIS_CONFIG_CHIP. This state machine is responsible for configuring the chip's bias generator.

DAVIS_CONFIG_CHIP

Module address: device-side chip control configuration. Shared with DAVIS_CONFIG_BIAS. This state machine is responsible for configuring the chip's internal control shift registers, to set special options.

DAVIS_CONFIG_SYSINFO

Module address: device-side system information. The system information module provides various details on the device, such as currently installed logic revision or clock speeds. All its parameters are read-only. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation for more details on what information is available.

DAVIS_CONFIG_USB

Module address: device-side USB output configuration. The USB output module forwards the data from the device and the FPGA/CPLD to the USB chip, usually a Cypress FX2 or FX3.

DAVIS_CONFIG_DDRAER

Module address: device-side DDR-AER output configuration. The DDR-AER output module forwards the data from the device and the FPGA/CPLD to some external device using a 4-phase handshake with data on both flanks.

DAVIS_CONFIG_MUX_RUN

Parameter address for module DAVIS_CONFIG_MUX: run the Multiplexer state machine, which is responsible for mixing the various event types at the device level, timestamping them and outputting them via USB or other connectors.

DAVIS_CONFIG_MUX_TIMESTAMP_RUN

Parameter address for module DAVIS_CONFIG_MUX: run the Timestamp Generator inside the Multiplexer state machine, which will provide microsecond accurate timestamps to the events passing through.

DAVIS_CONFIG_MUX_TIMESTAMP_RESET

Parameter address for module DAVIS_CONFIG_MUX: reset the Timestamp Generator to zero. This also sends a reset pulse to all connected slave devices, resetting their timestamp too.

DAVIS_CONFIG_MUX_RUN_CHIP

Parameter address for module DAVIS_CONFIG_MUX: power up the chip's bias generator, enabling the chip to work.

DAVIS_CONFIG_MUX_DROP_EXTINPUT_ON_TRANSFER_STALL

Parameter address for module DAVIS_CONFIG_MUX: drop External Input events if the USB output FIFO is full, instead of having them pile up at the input FIFOs.

DAVIS_CONFIG_MUX_DROP_DVS_ON_TRANSFER_STALL

Parameter address for module DAVIS_CONFIG_MUX: drop DVS events if the USB output FIFO is full, instead of having them pile up at the input FIFOs.

DAVIS_CONFIG_MUX_HAS_STATISTICS

Parameter address for module DAVIS_CONFIG_MUX: read-only parameter, information about the presence of the statistics feature. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get this information.

DAVIS_CONFIG_MUX_STATISTICS_EXTINPUT_DROPPED

Parameter address for module DAVIS_CONFIG_MUX: read-only parameter, representing the number of dropped External Input events on the device due to full USB buffers. This is a 64bit value, and should always be read using the function: *caerDeviceConfigGet64()*.

DAVIS_CONFIG_MUX_STATISTICS_DVS_DROPPED

Parameter address for module DAVIS_CONFIG_MUX: read-only parameter, representing the number of dropped DVS events on the device due to full USB buffers. This is a 64bit value, and should always be read using the function: *caerDeviceConfigGet64()*.

DAVIS_CONFIG_DVS_SIZE_COLUMNS

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, contains the X axis resolution of the DVS events returned by the camera. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get proper size information that already considers the rotation and orientation settings.

DAVIS_CONFIG_DVS_SIZE_ROWS

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, contains the Y axis resolution of the DVS events returned by the camera. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get proper size information that already considers the rotation and orientation settings.

DAVIS_CONFIG_DVS_ORIENTATION_INFO

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, contains information on the orientation of the X/Y axes, whether they should be inverted or not on the host when parsing incoming events. Bit 2: dvsInvertXY Bit 1: reserved Bit 0: reserved This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get proper size information that already considers the rotation and orientation settings.

DAVIS_CONFIG_DVS_RUN

Parameter address for module DAVIS_CONFIG_DVS: run the DVS state machine and get polarity events from the chip by handshaking with its AER bus.

DAVIS_CONFIG_DVS_WAIT_ON_TRANSFER_STALL

Parameter address for module DAVIS_CONFIG_DVS: if the output FIFO for this module is full, stall the

AER handshake with the chip and wait until it's free again, instead of just continuing the handshake and dropping the resulting events.

DAVIS_CONFIG_DVS_EXTERNAL_AER_CONTROL

Parameter address for module DAVIS_CONFIG_DVS: enable external AER control. This ensures the chip and the DVS pixel array are running, but doesn't do the handshake and leaves the ACK pin in high-impedance, to allow for an external system to take over the AER communication with the chip. DAVIS_CONFIG_DVS_RUN has to be turned off for this to work.

DAVIS_CONFIG_DVS_HAS_PIXEL_FILTER

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, information about the presence of the pixel filter feature. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get this information.

DAVIS_CONFIG_DVS_FILTER_PIXEL_0_ROW

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 0, Y axis setting.

DAVIS_CONFIG_DVS_FILTER_PIXEL_0_COLUMN

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 0, X axis setting.

DAVIS_CONFIG_DVS_FILTER_PIXEL_1_ROW

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 1, Y axis setting.

DAVIS_CONFIG_DVS_FILTER_PIXEL_1_COLUMN

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 1, X axis setting.

DAVIS_CONFIG_DVS_FILTER_PIXEL_2_ROW

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 2, Y axis setting.

DAVIS_CONFIG_DVS_FILTER_PIXEL_2_COLUMN

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 2, X axis setting.

DAVIS_CONFIG_DVS_FILTER_PIXEL_3_ROW

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 3, Y axis setting.

DAVIS_CONFIG_DVS_FILTER_PIXEL_3_COLUMN

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 3, X axis setting.

DAVIS_CONFIG_DVS_FILTER_PIXEL_4_ROW

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 4, Y axis setting.

DAVIS_CONFIG_DVS_FILTER_PIXEL_4_COLUMN

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 4, X axis setting.

DAVIS_CONFIG_DVS_FILTER_PIXEL_5_ROW

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 5, Y axis setting.

DAVIS_CONFIG_DVS_FILTER_PIXEL_5_COLUMN

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 5, X axis setting.

DAVIS_CONFIG_DVS_FILTER_PIXEL_6_ROW

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 6, Y axis setting.

DAVIS_CONFIG_DVS_FILTER_PIXEL_6_COLUMN

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 6, X axis setting.

DAVIS_CONFIG_DVS_FILTER_PIXEL_7_ROW

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 7, Y axis setting.

DAVIS_CONFIG_DVS_FILTER_PIXEL_7_COLUMN

Parameter address for module DAVIS_CONFIG_DVS: the pixel filter completely suppresses up to eight pixels in the DVS array, filtering out all events produced by them. This is the pixel 7, X axis setting.

DAVIS_CONFIG_DVS_HAS_BACKGROUND_ACTIVITY_FILTER

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, information about the presence of the background-activity filter feature. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get this information.

DAVIS_CONFIG_DVS_FILTER_BACKGROUND_ACTIVITY

Parameter address for module DAVIS_CONFIG_DVS: enable the background-activity filter, which tries to remove events caused by transistor leakage, by rejecting uncorrelated events.

DAVIS_CONFIG_DVS_FILTER_BACKGROUND_ACTIVITY_TIME

Parameter address for module DAVIS_CONFIG_DVS: specify the time difference constant for the background-activity filter. Range: 0 - 4095, in 250µs units. Events that are correlated within this time-frame are let through, while others are filtered out.

DAVIS_CONFIG_DVS_FILTER_REFRACTORY_PERIOD

Parameter address for module DAVIS_CONFIG_DVS: enable the refractory period filter, which limits the firing rate of pixels. This is supported together with the background-activity filter.

DAVIS_CONFIG_DVS_FILTER_REFRACTORY_PERIOD_TIME

Parameter address for module DAVIS_CONFIG_DVS: specify the time constant for the refractory period filter. Range: 0 - 4095, in 250µs units. Pixels will be inhibited from generating new events during this time after the last even has fired.

DAVIS_CONFIG_DVS_HAS_ROI_FILTER

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, information about the presence of the ROI filter feature. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get this information.

DAVIS_CONFIG_DVS_FILTER_ROI_START_COLUMN

Parameter address for module DAVIS_CONFIG_DVS: start position on the X axis for Region of Interest. Must be between 0 and DVS_SIZE_X-1, and be smaller or equal to DAVIS_CONFIG_DVS_FILTER_ROI_END_COLUMN.

DAVIS_CONFIG_DVS_FILTER_ROI_START_ROW

Parameter address for module DAVIS_CONFIG_DVS: start position on the Y axis for Region of Interest. Must be between 0 and DVS_SIZE_Y-1, and be smaller or equal to DAVIS_CONFIG_DVS_FILTER_ROI_END_ROW.

DAVIS_CONFIG_DVS_FILTER_ROI_END_COLUMN

Parameter address for module DAVIS_CONFIG_DVS: end position on the X axis for Region of Interest. Must be between 0 and DVS_SIZE_X-1, and be greater or equal to DAVIS_CONFIG_DVS_FILTER_ROI_START_COLUMN.

DAVIS_CONFIG_DVS_FILTER_ROI_END_ROW

Parameter address for module DAVIS_CONFIG_DVS: end position on the Y axis for Region of Interest. Must be between 0 and DVS_SIZE_Y-1, and be greater or equal to DAVIS_CONFIG_DVS_FILTER_ROI_START_ROW.

DAVIS_CONFIG_DVS_HAS_SKIP_FILTER

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, information about the presence of the event skip filter feature. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get this information.

DAVIS_CONFIG_DVS_FILTER_SKIP_EVENTS

Parameter address for module DAVIS_CONFIG_DVS: enable the event skip filter, which simply throws away one event every N events (decimation filter).

DAVIS_CONFIG_DVS_FILTER_SKIP_EVENTS_EVERY

Parameter address for module DAVIS_CONFIG_DVS: number of events to let through before skipping one. Range: 0 - 255 events.

DAVIS_CONFIG_DVS_HAS_POLARITY_FILTER

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, information about the presence of the polarity suppression filter feature. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get this information.

DAVIS_CONFIG_DVS_FILTER_POLARITY_FLATTEN

Parameter address for module DAVIS_CONFIG_DVS: flatten all polarities to OFF (0).

DAVIS_CONFIG_DVS_FILTER_POLARITY_SUPPRESS

Parameter address for module DAVIS_CONFIG_DVS: suppress one of the two ON/OFF polarities completely. Use DAVIS_CONFIG_DVS_FILTER_POLARITY_IGNORE to select which.

DAVIS_CONFIG_DVS_FILTER_POLARITY_SUPPRESS_TYPE

Parameter address for module DAVIS_CONFIG_DVS: polarity to suppress (0=OFF, 1=ON). Use DAVIS_CONFIG_DVS_FILTER_POLARITY_IGNORE to enable.

DAVIS_CONFIG_DVS_HAS_STATISTICS

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, information about the presence of the statistics feature. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get this information.

DAVIS_CONFIG_DVS_STATISTICS_EVENTS_ROW

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, representing the number of row event transactions completed on the device. This is a 64bit value, and should always be read using the function: *caerDeviceConfigGet64()*.

DAVIS_CONFIG_DVS_STATISTICS_EVENTS_COLUMN

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, representing the number of column event transactions completed on the device. This is a 64bit value, and should always be read using the function: *caerDeviceConfigGet64()*.

DAVIS_CONFIG_DVS_STATISTICS_EVENTS_DROPPED

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, representing the number of dropped transaction sequences on the device due to full buffers. This is a 64bit value, and should always be read using the function: *caerDeviceConfigGet64()*.

DAVIS_CONFIG_DVS_STATISTICS_FILTERED_PIXELS

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, representing the number of dropped events due to the pixel filter. This is a 64bit value, and should always be read using the function: *caerDeviceConfigGet64()*.

DAVIS_CONFIG_DVS_STATISTICS_FILTERED_BACKGROUND_ACTIVITY

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, representing the number of dropped events due to the background-activity filter. This is a 64bit value, and should always be read using the function: *caerDeviceConfigGet64()*.

DAVIS_CONFIG_DVS_STATISTICS_FILTERED_REFRACTORY_PERIOD

Parameter address for module DAVIS_CONFIG_DVS: read-only parameter, representing the number of dropped events due to the refractory period filter. This is a 64bit value, and should always be read using the function: *caerDeviceConfigGet64()*.

DAVIS_CONFIG_DVS_FILTER_PIXEL_AUTO_TRAIN

Parameter address for module DAVIS_CONFIG_DVS: automatically discover the eight most active pixels (above ~5KHz) and set up the hardware pixel filter to remove them from the output.

DAVIS_CONFIG_APS_SIZE_COLUMNS

Parameter address for module DAVIS_CONFIG_APS: read-only parameter, contains the X axis resolution of the APS frames returned by the camera. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get proper size information that already considers the rotation and orientation settings.

DAVIS_CONFIG_APS_SIZE_ROWS

Parameter address for module DAVIS_CONFIG_APS: read-only parameter, contains the Y axis resolution of the APS frames returned by the camera. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get proper size information that already considers the rotation and orientation settings.

DAVIS_CONFIG_APS_ORIENTATION_INFO

Parameter address for module DAVIS_CONFIG_APS: read-only parameter, contains information on the orientation of the X/Y axes, whether they should be inverted or not on the host when parsing incoming pixels, as well as if the X or Y axes need to be flipped when reading the pixels. Bit 2: apsInvertXY Bit 1: apsFlipX Bit 0: apsFlipY This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get proper size information that already considers the rotation and orientation settings.

DAVIS_CONFIG_APS_COLOR_FILTER

Parameter address for module DAVIS_CONFIG_APS: read-only parameter, contains information on the type of color filter present on the device. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get proper color filter information.

DAVIS_CONFIG_APS_RUN

Parameter address for module DAVIS_CONFIG_APS: enable the APS module and take intensity images of the scene. While this parameter is enabled, frames will be taken continuously. To slow down the frame-rate, see DAVIS_CONFIG_APS_FRAME_DELAY. To only take snapshots, see DAVIS_CONFIG_APS_SNAPSHOT.

DAVIS_CONFIG_APS_WAIT_ON_TRANSFER_STALL

Parameter address for module DAVIS_CONFIG_APS: if the output FIFO for this module is full, stall the APS state machine and wait until it's free again, instead of just dropping the pixels as they are being read out. This guarantees a complete frame readout, at the possible cost of slight timing differences between pixels. If disabled, incomplete frames may be transmitted and will then be dropped on the host, resulting in lower frame-rates, especially during high DVS traffic.

DAVIS_CONFIG_APS_HAS_GLOBAL_SHUTTER

Parameter address for module DAVIS_CONFIG_APS: read-only parameter, information about the presence of the global shutter feature. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get this information.

DAVIS_CONFIG_APS_GLOBAL_SHUTTER

Parameter address for module DAVIS_CONFIG_APS: enable Global Shutter mode instead of Rolling Shutter. The Global Shutter eliminates motion artifacts, but is noisier than the Rolling Shutter (worse quality).

DAVIS_CONFIG_APS_START_COLUMN_0

Parameter address for module DAVIS_CONFIG_APS: start position on the X axis for Region of Interest 0. Must be between 0 and APS_SIZE_X-1, and be smaller or equal to DAVIS_CONFIG_APS_END_COLUMN_0.

DAVIS_CONFIG_APS_START_ROW_0

Parameter address for module DAVIS_CONFIG_APS: start position on the Y axis for Region of Interest 0. Must be between 0 and APS_SIZE_Y-1, and be smaller or equal to DAVIS_CONFIG_APS_END_ROW_0.

DAVIS_CONFIG_APS_END_COLUMN_0

Parameter address for module DAVIS_CONFIG_APS: end position on the X axis for Region of Interest 0. Must be between 0 and APS_SIZE_X-1, and be greater or equal to DAVIS_CONFIG_APS_START_COLUMN_0.

DAVIS_CONFIG_APS_END_ROW_0

Parameter address for module DAVIS_CONFIG_APS: end position on the Y axis for Region of Interest 0. Must be between 0 and APS_SIZE_Y-1, and be greater or equal to DAVIS_CONFIG_APS_START_ROW_0.

DAVIS_CONFIG_APS_EXPOSURE

Parameter address for module DAVIS_CONFIG_APS: frame exposure time. Range: 0-4194303, in microseconds (maximum ~4s). Very precise for Global Shutter, slightly less exact for Rolling Shutter due to column-based timing constraints.

DAVIS_CONFIG_APS_FRAME_INTERVAL

Parameter address for module DAVIS_CONFIG_APS: time between consecutive frames. Range: 0-8388607, in microseconds (maximum ~8s). This can be used to set a frame-rate. Please note the frame-rate is best-effort, and may not be met if readout and exposure times exceed this value.

DAVIS640H_CONFIG_APS_TRANSFER

Parameter address for module DAVIS_CONFIG_APS (only for DAVIS640H chip): charge transfer time in ADCClock cycles.

DAVIS640H_CONFIG_APS_RSFDSETTLE

Parameter address for module DAVIS_CONFIG_APS (only for DAVIS640H chip): Rolling Shutter FD settle time in ADCClock cycles.

DAVIS640H_CONFIG_APS_GSPDRESET

Parameter address for module DAVIS_CONFIG_APS (only for DAVIS640H chip): Global Shutter PD reset time in ADCClock cycles.

DAVIS640H_CONFIG_APS_GSRESETFALL

Parameter address for module DAVIS_CONFIG_APS (only for DAVIS640H chip): Global Shutter Reset Fall time in ADCClock cycles.

DAVIS640H_CONFIG_APS_GSTXFALL

Parameter address for module DAVIS_CONFIG_APS (only for DAVIS640H chip): Global Shutter Transfer Fall time in ADCClock cycles.

DAVIS640H_CONFIG_APS_GSFDRESET

Parameter address for module DAVIS_CONFIG_APS (only for DAVIS640H chip): Global Shutter FD reset time in ADCClock cycles.

DAVIS_CONFIG_APS_SNAPSHOT

Parameter address for module DAVIS_CONFIG_APS: takes a snapshot (one frame), like a photo-camera. More efficient implementation that just toggling the DAVIS_CONFIG_APS_RUN parameter. The APS module should not be running prior to calling this, as it only makes sense if frames are not being generated at the time. Also, DAVIS_CONFIG_APS_FRAME_INTERVAL should be set to zero if only doing snapshots, to ensure a quicker readiness for the next one, since the delay is always observed after taking a frame.

DAVIS_CONFIG_APS_AUTOEXPOSURE

Parameter address for module DAVIS_CONFIG_APS: automatic exposure control, tries to set the exposure value automatically to an appropriate value to maximize information in the scene and minimize under- and over-exposure.

DAVIS_CONFIG_APS_FRAME_MODE

Parameter address for module DAVIS_CONFIG_APS: select desired type of frame output. Available are: 0 - Default, meaning grayscale on MONO cameras and RGB color on cameras with color filters. 1 - Grayscale, always a grayscale intensity frame. 2 - Original, send the frame exactly as it comes in from the device (will show grid pattern on color cameras).

DAVIS_CONFIG_IMU_TYPE

Parameter address for module DAVIS_CONFIG_IMU: read-only parameter, contains information on the type of IMU chip being used in this device: 0 - no IMU present 1 - InvenSense MPU 6050/6150 2 - InvenSense MPU 9250 This is reserved for internal use and should not be used by anything other than libcaer.

DAVIS_CONFIG_IMU_ORIENTATION_INFO

Parameter address for module DAVIS_CONFIG_IMU: read-only parameter, contains information on the orientation of the X/Y/Z axes, whether they should be flipped or not on the host when parsing incoming IMU data samples. Bit 2: imuFlipX Bit 1: imuFlipY Bit 0: imuFlipZ This is reserved for internal use and should not be used by anything other than libcaer. Generated IMU events are already properly flipped when returned to the user.

DAVIS_CONFIG_IMU_RUN_ACCELEROMETER

Parameter address for module DAVIS_CONFIG_IMU: enable the IMU's accelerometer. This takes the IMU chip out of sleep.

DAVIS_CONFIG_IMU_RUN_GYROSCOPE

Parameter address for module DAVIS_CONFIG_IMU: enable the IMU's gyroscope. This takes the IMU chip out of sleep.

DAVIS_CONFIG_IMU_RUN_TEMPERATURE

Parameter address for module DAVIS_CONFIG_IMU: enable the IMU's temperature sensor. This takes the IMU chip out of sleep.

DAVIS_CONFIG_IMU_SAMPLE_RATE_DIVIDER

Parameter address for module DAVIS_CONFIG_IMU: this specifies the divider from the Gyroscope Output Rate used to generate the Sample Rate for the IMU. Valid values are from 0 to 255. The Sample Rate is generated like this: Sample Rate = Gyroscope Output Rate / (1 + DAVIS_CONFIG_IMU_SAMPLE_RATE_DIVIDER) where Gyroscope Output Rate = 8 kHz when DAVIS_CONFIG_IMU_DIGITAL_LOW_PASS_FILTER is disabled (set to 0 or 7), and 1 kHz when enabled. Note: the accelerometer output rate is 1 kHz. This means that for a Sample Rate greater than 1 kHz, the same accelerometer sample may be output multiple times.

DAVIS_CONFIG_IMU_ACCEL_DLPF

Parameter address for module DAVIS_CONFIG_IMU: this configures the digital low-pass filter for both the accelerometer and the gyroscope on InvenSense MPU 6050/6150 IMU devices, or for the accelerometer only on InvenSense MPU 9250. Valid values are from 0 to 7 and have the following meaning:

On InvenSense MPU 6050/6150: 0 - Accel: BW=260Hz, Delay=0ms, FS=1kHz - Gyro: BW=256Hz, Delay=0.98ms, FS=8kHz 1 - Accel: BW=184Hz, Delay=2.0ms, FS=1kHz - Gyro: BW=188Hz, Delay=1.9ms, FS=1kHz 2 - Accel: BW=94Hz, Delay=3.0ms, FS=1kHz - Gyro: BW=98Hz, Delay=2.8ms, FS=1kHz 3 - Accel: BW=44Hz, Delay=4.9ms, FS=1kHz - Gyro: BW=42Hz, Delay=4.8ms, FS=1kHz 4 - Accel: BW=21Hz, Delay=8.5ms, FS=1kHz - Gyro: BW=20Hz, Delay=8.3ms, FS=1kHz 5 - Accel: BW=10Hz, Delay=13.8ms, FS=1kHz - Gyro: BW=10Hz, Delay=13.4ms, FS=1kHz 6 - Accel: BW=5Hz, Delay=19.0ms, FS=1kHz - Gyro: BW=5Hz, Delay=18.6ms, FS=1kHz 7 - Accel: RESERVED, FS=1kHz - Gyro: RESERVED, FS=8kHz

On InvenSense MPU 9250: 0 - Accel: BW=218.1Hz, Delay=1.88ms, FS=1kHz 1 - Accel: BW=218.1Hz, Delay=1.88ms, FS=1kHz 2 - Accel: BW=99Hz, Delay=2.88ms, FS=1kHz 3 - Accel: BW=44.8Hz, Delay=4.88ms, FS=1kHz 4 - Accel: BW=21.2Hz, Delay=8.87ms, FS=1kHz 5 - Accel: BW=10.2Hz, Delay=16.83ms, FS=1kHz 6 - Accel: BW=5.05Hz, Delay=32.48ms, FS=1kHz 7 - Accel: BW=420Hz, Delay=1.38ms, FS=1kHz

DAVIS_CONFIG_IMU_DIGITAL_LOW_PASS_FILTER

DAVIS_CONFIG_IMU_ACCEL_FULL_SCALE

Parameter address for module DAVIS_CONFIG_IMU: select the full scale range of the accelerometer outputs. Valid values are: 0 - +- 2 g 1 - +- 4 g 2 - +- 8 g 3 - +- 16 g

DAVIS_CONFIG_IMU_GYRO_DLPF

Parameter address for module DAVIS_CONFIG_IMU: this configures the digital low-pass filter for the gyroscope on devices using the InvenSense MPU 9250. Valid values are from 0 to 7 and have the following meaning:

0 - Gyro: BW=250Hz, Delay=0.97ms, FS=8kHz 1 - Gyro: BW=184Hz, Delay=2.9ms, FS=1kHz 2 - Gyro: BW=92Hz, Delay=3.9ms, FS=1kHz 3 - Gyro: BW=41Hz, Delay=5.9ms, FS=1kHz 4 - Gyro: BW=20Hz, Delay=9.9ms, FS=1kHz 5 - Gyro: BW=10Hz, Delay=17.85ms, FS=1kHz 6 - Gyro: BW=5Hz, Delay=33.48ms, FS=1kHz 7 - Gyro: BW=3600Hz, Delay=0.17ms, FS=8kHz

DAVIS_CONFIG_IMU_GYRO_FULL_SCALE

Parameter address for module DAVIS_CONFIG_IMU: select the full scale range of the gyroscope outputs. Valid values are: 0 - +- 250 °/s 1 - +- 500 °/s 2 - +- 1000 °/s 3 - +- 2000 °/s

DAVIS_CONFIG_EXTINPUT_RUN_DETECTOR

Parameter address for module DAVIS_CONFIG_EXTINPUT: enable the signal detector module. It generates events when it sees certain types of signals, such as edges or pulses of a defined length, on the IN JACK signal. This can be useful to inject events into the event stream in response to external stimuli or controls, such as turning on a LED lamp.

DAVIS_CONFIG_EXTINPUT_DETECT_RISING_EDGES

Parameter address for module DAVIS_CONFIG_EXTINPUT: send a special EXTERNAL_INPUT_RISING_EDGE event when a rising edge is detected (transition from low voltage to high).

DAVIS_CONFIG_EXTINPUT_DETECT_FALLING_EDGES

Parameter address for module DAVIS_CONFIG_EXTINPUT: send a special EXTERNAL_INPUT_FALLING_EDGE event when a falling edge is detected (transition from high voltage to low).

DAVIS_CONFIG_EXTINPUT_DETECT_PULSES

Parameter address for module DAVIS_CONFIG_EXTINPUT: send a special EXTERNAL_INPUT_PULSE event when a pulse, of a specified, configurable polarity and length, is detected. See DAVIS_CONFIG_EXTINPUT_DETECT_PULSE_POLARITY and DAVIS_CONFIG_EXTINPUT_DETECT_PULSE_LENGTH for more details.

DAVIS_CONFIG_EXTINPUT_DETECT_PULSE_POLARITY

Parameter address for module DAVIS_CONFIG_EXTINPUT: the polarity the pulse must exhibit to be detected as such. '1' means active high; a pulse will start when the signal goes from low to high and will continue to be seen as the same pulse as long as it stays high. '0' means active low; a pulse will start when the signal goes from high to low and will continue to be seen as the same pulse as long as it stays low.

DAVIS_CONFIG_EXTINPUT_DETECT_PULSE_LENGTH

Parameter address for module DAVIS_CONFIG_EXTINPUT: the minimal length that a pulse must have to trigger the sending of a special event. Range: 1-1048575, in microseconds.

DAVIS_CONFIG_EXTINPUT_HAS_GENERATOR

Parameter address for module DAVIS_CONFIG_EXTINPUT: read-only parameter, information about the presence of the signal generator feature. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get this information.

DAVIS_CONFIG_EXTINPUT_RUN_GENERATOR

Parameter address for module DAVIS_CONFIG_EXTINPUT: enable the signal generator module. It generates a PWM-like signal based on configurable parameters and outputs it on the OUT JACK signal.

DAVIS_CONFIG_EXTINPUT_GENERATE_PULSE_POLARITY

Parameter address for module DAVIS_CONFIG_EXTINPUT: polarity of the PWM-like signal to be generated. '1' means active high, '0' means active low.

DAVIS_CONFIG_EXTINPUT_GENERATE_PULSE_INTERVAL

Parameter address for module DAVIS_CONFIG_EXTINPUT: the interval between the start of two consecutive pulses. Range: 1-1048575, in microseconds. This must

be bigger or equal to DAVIS_CONFIG_EXTINPUT_GENERATE_PULSE_LENGTH. To generate a signal with 50% duty cycle, this would have to be exactly double of DAVIS_CONFIG_EXTINPUT_GENERATE_PULSE_LENGTH.

DAVIS_CONFIG_EXTINPUT_GENERATE_PULSE_LENGTH

Parameter address for module DAVIS_CONFIG_EXTINPUT: the length a pulse stays active. Range: 1-1048575, in microseconds. This must be smaller or equal to DAVIS_CONFIG_EXTINPUT_GENERATE_PULSE_INTERVAL. To generate a signal with 50% duty cycle, this would have to be exactly half of DAVIS_CONFIG_EXTINPUT_GENERATE_PULSE_INTERVAL.

DAVIS_CONFIG_EXTINPUT_GENERATE_INJECT_ON_RISING_EDGE

Parameter address for module DAVIS_CONFIG_EXTINPUT: enables event injection when a rising edge occurs in the generated signal; a special event EXTERNAL_GENERATOR_RISING_EDGE is emitted into the event stream.

DAVIS_CONFIG_EXTINPUT_GENERATE_INJECT_ON_FALLING_EDGE

Parameter address for module DAVIS_CONFIG_EXTINPUT: enables event injection when a falling edge occurs in the generated signal; a special event EXTERNAL_GENERATOR_FALLING_EDGE is emitted into the event stream.

DAVIS_CONFIG_SYSINFO_LOGIC_VERSION

Parameter address for module DAVIS_CONFIG_SYSINFO: read-only parameter, the version of the logic currently running on the device's FPGA/CPLD. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get this information.

DAVIS_CONFIG_SYSINFO_CHIP_IDENTIFIER

Parameter address for module DAVIS_CONFIG_SYSINFO: read-only parameter, an integer used to identify the different types of sensor chips used on the device. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get this information.

DAVIS_CONFIG_SYSINFO_DEVICE_IS_MASTER

Parameter address for module DAVIS_CONFIG_SYSINFO: read-only parameter, whether the device is currently a timestamp master or slave when synchronizing multiple devices together. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get this information.

DAVIS_CONFIG_SYSINFO_LOGIC_CLOCK

Parameter address for module DAVIS_CONFIG_SYSINFO: read-only parameter, the frequency in MHz at which the main FPGA/CPLD logic is running. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get this information.

DAVIS_CONFIG_SYSINFO_ADC_CLOCK

Parameter address for module DAVIS_CONFIG_SYSINFO: read-only parameter, the frequency in MHz at which the FPGA/CPLD logic related to APS frame grabbing is running. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get this information.

DAVIS_CONFIG_SYSINFO_USB_CLOCK

Parameter address for module DAVIS_CONFIG_SYSINFO: read-only parameter, the frequency in MHz at which the FPGA/CPLD logic related to USB data transmission is running. This is reserved for internal use and should not be used by anything other than libcaer.

DAVIS_CONFIG_SYSINFO_CLOCK_DEVIATION

Parameter address for module DAVIS_CONFIG_SYSINFO: read-only parameter, the deviation factor for the clocks. Due to how FX3 generates the clocks, which are then used by FPGA/CPLD, they are not integers but have a fractional part. This is reserved for internal use and should not be used by anything other than libcaer.

DAVIS_CONFIG_SYSINFO_LOGIC_PATCH

Parameter address for module DAVIS_CONFIG_SYSINFO: read-only parameter, the patch version of the logic currently running on the device's FPGA/CPLD. This is reserved for internal use and should not be used by anything other than libcaer.

DAVIS_CONFIG_USB_RUN

Parameter address for module DAVIS_CONFIG_USB: enable the USB FIFO module, which transfers the data from the FPGA/CPLD to the USB chip, to be then sent to the host. Turning this off will suppress any USB data communication!

DAVIS_CONFIG_USB_EARLY_PACKET_DELAY

Parameter address for module DAVIS_CONFIG_USB: the time delay after which a packet of data is committed to USB, even if it is not full yet (short USB packet). The value is in 125µs time-slices, corresponding to how USB schedules its operations (a value of 4 for example would mean waiting at most 0.5ms until sending a short USB packet to the host).

DAVIS_CONFIG_DDRAER_RUN

Parameter address for module DAVIS_CONFIG_DDRAER: enable the DDR-AER output module, which transfers the data from the FPGA/CPLD to some external device like a Raspberry Pi.

DAVIS128_CONFIG_BIAS_APSOVERFLOWLEVEL

Parameter address for module DAVIS128_CONFIG_BIAS: DAVIS128 chip biases. Bias configuration values must be generated using the proper functions, which are:

- *caerBiasVDACGenerate()* for VDAC (voltage) biases.
- *caerBiasCoarseFineGenerate()* for coarse-fine (current) biases.
- *caerBiasShiftedSourceGenerate()* for shifted-source biases. See 'https://inivation.com/support/hardware/biasing/'[6] for more details.

DAVIS128_CONFIG_BIAS_APSCAS

DAVIS128_CONFIG_BIAS_ADCREFHIGH

DAVIS128_CONFIG_BIAS_ADCREFLOW

DAVIS128_CONFIG_BIAS_LOCALBUFBN

DAVIS128_CONFIG_BIAS_PADFOLLBN

DAVIS128_CONFIG_BIAS_DIFFBN

DAVIS128_CONFIG_BIAS_ONBN

DAVIS128_CONFIG_BIAS_OFFBN

DAVIS128_CONFIG_BIAS_PIXINVBN

DAVIS128_CONFIG_BIAS_PRBP

DAVIS128_CONFIG_BIAS_PRSFBP

DAVIS128_CONFIG_BIAS_REFRBP

DAVIS128_CONFIG_BIAS_READOUTBUFBP

DAVIS128_CONFIG_BIAS_APSROSFBN

DAVIS128_CONFIG_BIAS_ADCCOMPBP

DAVIS128_CONFIG_BIAS_COLSELLOWBN

DAVIS128_CONFIG_BIAS_DACBUFBP

DAVIS128_CONFIG_BIAS_LCOLTIMEOUTBN

DAVIS128_CONFIG_BIAS_AEPDBN

DAVIS128_CONFIG_BIAS_AEPUXBP

DAVIS128_CONFIG_BIAS_AEPUYBP

DAVIS128_CONFIG_BIAS_IFREFRBN

DAVIS128_CONFIG_BIAS_IFTHRBN

DAVIS128_CONFIG_BIAS_BIASBUFFER

DAVIS128_CONFIG_BIAS_SSP

DAVIS128_CONFIG_BIAS_SSN

DAVIS128_CONFIG_CHIP_DIGITALMUX0

Parameter address for module DAVIS128_CONFIG_CHIP: DAVIS128 chip configuration. These are for expert control and should never be used or changed unless for advanced debugging purposes. To change the Global Shutter configuration, please use DAVIS_CONFIG_APS_GLOBAL_SHUTTER instead.

DAVIS128_CONFIG_CHIP_DIGITALMUX1

DAVIS128_CONFIG_CHIP_DIGITALMUX2

DAVIS128_CONFIG_CHIP_DIGITALMUX3

DAVIS128_CONFIG_CHIP_ANALOGMUX0

DAVIS128_CONFIG_CHIP_ANALOGMUX1

DAVIS128_CONFIG_CHIP_ANALOGMUX2

DAVIS128_CONFIG_CHIP_BIASMUX0

DAVIS128_CONFIG_CHIP_RESETCALIBNEURON

DAVIS128_CONFIG_CHIP_TYPENCALIBNEURON

DAVIS128_CONFIG_CHIP_RESETTESTPIXEL

DAVIS128_CONFIG_CHIP_AERNAROW

DAVIS128_CONFIG_CHIP_USEAOUT

DAVIS128_CONFIG_CHIP_GLOBAL_SHUTTER

DAVIS128_CONFIG_CHIP_SELECTGRAYCOUNTER

DAVIS208_CONFIG_BIAS_APSOVERFLOWLEVEL

Parameter address for module DAVIS208_CONFIG_BIAS: DAVIS208 chip biases. Bias configuration values must be generated using the proper functions, which are:

- *caerBiasVDACGenerate()* for VDAC (voltage) biases.
- *caerBiasCoarseFineGenerate()* for coarse-fine (current) biases.
- *caerBiasShiftedSourceGenerate()* for shifted-source biases. See 'https://inivation.com/support/hardware/biasing/'[7] for more details.

DAVIS208_CONFIG_BIAS_APSCAS

DAVIS208_CONFIG_BIAS_ADCREFHIGH

DAVIS208_CONFIG_BIAS_ADCREFLOW

DAVIS208_CONFIG_BIAS_RESETHIGHPASS

DAVIS208_CONFIG_BIAS_REFSS

DAVIS208_CONFIG_BIAS_LOCALBUFBN

DAVIS208_CONFIG_BIAS_PADFOLLBN

DAVIS208_CONFIG_BIAS_DIFFBN

DAVIS208_CONFIG_BIAS_ONBN

DAVIS208_CONFIG_BIAS_OFFBN

DAVIS208_CONFIG_BIAS_PIXINVBN

DAVIS208_CONFIG_BIAS_PRBP

DAVIS208_CONFIG_BIAS_PRSFBP

DAVIS208_CONFIG_BIAS_REFRBP

DAVIS208_CONFIG_BIAS_READOUTBUFBP

DAVIS208_CONFIG_BIAS_APSROSFBN

DAVIS208_CONFIG_BIAS_ADCCOMPBP

DAVIS208_CONFIG_BIAS_COLSELLOWBN

DAVIS208_CONFIG_BIAS_DACBUFBP

DAVIS208_CONFIG_BIAS_LCOLTIMEOUTBN

DAVIS208_CONFIG_BIAS_AEPDBN

DAVIS208_CONFIG_BIAS_AEPUXBP

DAVIS208_CONFIG_BIAS_AEPUYBP

DAVIS208_CONFIG_BIAS_IFREFRBN

DAVIS208_CONFIG_BIAS_IFTHRBN

DAVIS208_CONFIG_BIAS_REGBIASBP

DAVIS208_CONFIG_BIAS_REFSSBN

DAVIS208_CONFIG_BIAS_BIASBUFFER

DAVIS208_CONFIG_BIAS_SSP

DAVIS208_CONFIG_BIAS_SSN

DAVIS208_CONFIG_CHIP_DIGITALMUX0

Parameter address for module DAVIS208_CONFIG_CHIP: DAVIS208 chip configuration. These are for expert control and should never be used or changed unless for advanced debugging purposes. To change the Global Shutter configuration, please use DAVIS_CONFIG_APS_GLOBAL_SHUTTER instead.

DAVIS208_CONFIG_CHIP_DIGITALMUX1

DAVIS208_CONFIG_CHIP_DIGITALMUX2

DAVIS208_CONFIG_CHIP_DIGITALMUX3

DAVIS208_CONFIG_CHIP_ANALOGMUX0

DAVIS208_CONFIG_CHIP_ANALOGMUX1

DAVIS208_CONFIG_CHIP_ANALOGMUX2

DAVIS208_CONFIG_CHIP_BIASMUX0

DAVIS208_CONFIG_CHIP_RESETCALIBNEURON

DAVIS208_CONFIG_CHIP_TYPENCALIBNEURON

DAVIS208_CONFIG_CHIP_RESETTESTPIXEL

DAVIS208_CONFIG_CHIP_AERNAROW

DAVIS208_CONFIG_CHIP_USEAOUT

DAVIS208_CONFIG_CHIP_GLOBAL_SHUTTER

DAVIS208_CONFIG_CHIP_SELECTGRAYCOUNTER

DAVIS208_CONFIG_CHIP_SELECTPREAMPAVG

DAVIS208_CONFIG_CHIP_SELECTBIASREFSS

DAVIS208_CONFIG_CHIP_SELECTSENSE

DAVIS208_CONFIG_CHIP_SELECTPOSFB

DAVIS208_CONFIG_CHIP_SELECTHIGHPASS

DAVIS240_CONFIG_BIAS_DIFFBN

Parameter address for module DAVIS240_CONFIG_BIAS: DAVIS240chip biases. Bias configuration values must be generated using the proper functions, which are:

- *caerBiasCoarseFineGenerate()* for coarse-fine (current) biases.
- *caerBiasShiftedSourceGenerate()* for shifted-source biases. See 'https://inivation.com/support/hardware/biasing/'[8] for more details.

DAVIS240_CONFIG_BIAS_ONBN

DAVIS240_CONFIG_BIAS_OFFBN

DAVIS240_CONFIG_BIAS_APSCASEPC

DAVIS240_CONFIG_BIAS_DIFFCASBNC

DAVIS240_CONFIG_BIAS_APSROSFBN

DAVIS240_CONFIG_BIAS_LOCALBUFBN

DAVIS240_CONFIG_BIAS_PIXINVBN

DAVIS240_CONFIG_BIAS_PRBP

DAVIS240_CONFIG_BIAS_PRSFBP

DAVIS240_CONFIG_BIAS_REFRBP

DAVIS240_CONFIG_BIAS_AEPDBN

DAVIS240_CONFIG_BIAS_LCOLTIMEOUTBN

DAVIS240_CONFIG_BIAS_AEPUXBP

DAVIS240_CONFIG_BIAS_AEPUYBP

DAVIS240_CONFIG_BIAS_IFTHRBN

DAVIS240_CONFIG_BIAS_IFREFRBN

DAVIS240_CONFIG_BIAS_PADFOLLBN

DAVIS240_CONFIG_BIAS_APSOVERFLOWLEVELBN

DAVIS240_CONFIG_BIAS_BIASBUFFER

DAVIS240_CONFIG_BIAS_SSP

DAVIS240_CONFIG_BIAS_SSN

DAVIS240_CONFIG_CHIP_DIGITALMUX0

Parameter address for module DAVIS240_CONFIG_CHIP: DAVIS240 chip configuration. These are for expert control and should never be used or changed unless for advanced debugging purposes. To change the Global Shutter configuration, please use DAVIS_CONFIG_APS_GLOBAL_SHUTTER instead. On DAVIS240B cameras, DAVIS240_CONFIG_CHIP_SPECIALPIXELCONTROL can be used to enable the test pixel array.

DAVIS240_CONFIG_CHIP_DIGITALMUX1

DAVIS240_CONFIG_CHIP_DIGITALMUX2

DAVIS240_CONFIG_CHIP_DIGITALMUX3

DAVIS240_CONFIG_CHIP_ANALOGMUX0

DAVIS240_CONFIG_CHIP_ANALOGMUX1

DAVIS240_CONFIG_CHIP_ANALOGMUX2

DAVIS240_CONFIG_CHIP_BIASMUX0

DAVIS240_CONFIG_CHIP_RESETCALIBNEURON

DAVIS240_CONFIG_CHIP_TYPENCALIBNEURON

DAVIS240_CONFIG_CHIP_RESETTESTPIXEL

DAVIS240_CONFIG_CHIP_SPECIALPIXELCONTROL

DAVIS240_CONFIG_CHIP_AERNAROW

DAVIS240_CONFIG_CHIP_USEAOUT

DAVIS240_CONFIG_CHIP_GLOBAL_SHUTTER

DAVIS346_CONFIG_BIAS_APSOVERFLOWLEVEL

Parameter address for module DAVIS346_CONFIG_BIAS: DAVIS346 chip biases. Bias configuration values must be generated using the proper functions, which are:

- *caerBiasVDACGenerate()* for VDAC (voltage) biases.
- *caerBiasCoarseFineGenerate()* for coarse-fine (current) biases.
- *caerBiasShiftedSourceGenerate()* for shifted-source biases. See 'https://inivation.com/support/hardware/biasing/'[9] for more details.

DAVIS346_CONFIG_BIAS_APSCAS

DAVIS346_CONFIG_BIAS_ADCREFHIGH

DAVIS346_CONFIG_BIAS_ADCREFLOW

DAVIS346_CONFIG_BIAS_ADCTESTVOLTAGE

DAVIS346_CONFIG_BIAS_LOCALBUFBN

DAVIS346_CONFIG_BIAS_PADFOLLBN

DAVIS346_CONFIG_BIAS_DIFFBN

DAVIS346_CONFIG_BIAS_ONBN

DAVIS346_CONFIG_BIAS_OFFBN

DAVIS346_CONFIG_BIAS_PIXINVBN

DAVIS346_CONFIG_BIAS_PRBP

DAVIS346_CONFIG_BIAS_PRSFBP

DAVIS346_CONFIG_BIAS_REFRBP

DAVIS346_CONFIG_BIAS_READOUTBUFBP

DAVIS346_CONFIG_BIAS_APSROSFBN

DAVIS346_CONFIG_BIAS_ADCCOMPBP

DAVIS346_CONFIG_BIAS_COLSELLOWBN

DAVIS346_CONFIG_BIAS_DACBUFBP

DAVIS346_CONFIG_BIAS_LCOLTIMEOUTBN

DAVIS346_CONFIG_BIAS_AEPDBN

DAVIS346_CONFIG_BIAS_AEPUXBP

DAVIS346_CONFIG_BIAS_AEPUYBP

DAVIS346_CONFIG_BIAS_IFREFRBN

DAVIS346_CONFIG_BIAS_IFTHRBN

DAVIS346_CONFIG_BIAS_BIASBUFFER

DAVIS346_CONFIG_BIAS_SSP

DAVIS346_CONFIG_BIAS_SSN

DAVIS346_CONFIG_CHIP_DIGITALMUX0

Parameter address for module DAVIS346_CONFIG_CHIP: DAVIS346 chip configuration. These are for expert control and should never be used or changed unless for advanced debugging purposes. To change the Global Shutter configuration, please use DAVIS_CONFIG_APS_GLOBAL_SHUTTER instead.

DAVIS346_CONFIG_CHIP_DIGITALMUX1

DAVIS346_CONFIG_CHIP_DIGITALMUX2

DAVIS346_CONFIG_CHIP_DIGITALMUX3

DAVIS346_CONFIG_CHIP_ANALOGMUX0

DAVIS346_CONFIG_CHIP_ANALOGMUX1

DAVIS346_CONFIG_CHIP_ANALOGMUX2

DAVIS346_CONFIG_CHIP_BIASMUX0

DAVIS346_CONFIG_CHIP_RESETCALIBNEURON

DAVIS346_CONFIG_CHIP_TYPENCALIBNEURON

DAVIS346_CONFIG_CHIP_RESETTESTPIXEL

DAVIS346_CONFIG_CHIP_AERNAROW

DAVIS346_CONFIG_CHIP_USEAOUT

DAVIS346_CONFIG_CHIP_GLOBAL_SHUTTER

DAVIS346_CONFIG_CHIP_SELECTGRAYCOUNTER

DAVIS346_CONFIG_CHIP_TESTADC

DAVIS640_CONFIG_BIAS_APSOVERFLOWLEVEL

Parameter address for module DAVIS640_CONFIG_BIAS: DAVIS640 chip biases. Bias configuration values must be generated using the proper functions, which are:

- *caerBiasVDACGenerate()* for VDAC (voltage) biases.
- *caerBiasCoarseFineGenerate()* for coarse-fine (current) biases.
- *caerBiasShiftedSourceGenerate()* for shifted-source biases. See 'https://inivation.com/support/hardware/biasing/'[10] for more details.

DAVIS640_CONFIG_BIAS_APSCAS

DAVIS640_CONFIG_BIAS_ADCREFHIGH

DAVIS640_CONFIG_BIAS_ADCREFLOW

DAVIS640_CONFIG_BIAS_ADCTESTVOLTAGE

DAVIS640_CONFIG_BIAS_LOCALBUFBN

DAVIS640_CONFIG_BIAS_PADFOLLBN

DAVIS640_CONFIG_BIAS_DIFFBN

DAVIS640_CONFIG_BIAS_ONBN

DAVIS640_CONFIG_BIAS_OFFBN

DAVIS640_CONFIG_BIAS_PIXINVBN

DAVIS640_CONFIG_BIAS_PRBP

DAVIS640_CONFIG_BIAS_PRSFBP

DAVIS640_CONFIG_BIAS_REFRBP

DAVIS640_CONFIG_BIAS_READOUTBUFBP

DAVIS640_CONFIG_BIAS_APSROSFBN

DAVIS640_CONFIG_BIAS_ADCCOMPBP

DAVIS640_CONFIG_BIAS_COLSELLOWBN

DAVIS640_CONFIG_BIAS_DACBUFBP

DAVIS640_CONFIG_BIAS_LCOLTIMEOUTBN

DAVIS640_CONFIG_BIAS_AEPDBN

DAVIS640_CONFIG_BIAS_AEPUXBP

DAVIS640_CONFIG_BIAS_AEPUYBP

DAVIS640_CONFIG_BIAS_IFREFRBN

DAVIS640_CONFIG_BIAS_IFTHRBN

DAVIS640_CONFIG_BIAS_BIASBUFFER

DAVIS640_CONFIG_BIAS_SSP

DAVIS640_CONFIG_BIAS_SSN

DAVIS640_CONFIG_CHIP_DIGITALMUX0

Parameter address for module DAVIS640_CONFIG_CHIP: DAVIS640 chip configuration. These are for expert control and should never be used or changed unless for advanced debugging purposes. To change the Global Shutter configuration, please use DAVIS_CONFIG_APS_GLOBAL_SHUTTER instead.

DAVIS640_CONFIG_CHIP_DIGITALMUX1

DAVIS640_CONFIG_CHIP_DIGITALMUX2

DAVIS640_CONFIG_CHIP_DIGITALMUX3

DAVIS640_CONFIG_CHIP_ANALOGMUX0

DAVIS640_CONFIG_CHIP_ANALOGMUX1

DAVIS640_CONFIG_CHIP_ANALOGMUX2

DAVIS640_CONFIG_CHIP_BIASMUX0

DAVIS640_CONFIG_CHIP_RESETCALIBNEURON

DAVIS640_CONFIG_CHIP_TYPENCALIBNEURON

DAVIS640_CONFIG_CHIP_RESETTESTPIXEL

DAVIS640_CONFIG_CHIP_AERNAROW

DAVIS640_CONFIG_CHIP_USEAOUT

DAVIS640_CONFIG_CHIP_GLOBAL_SHUTTER

DAVIS640_CONFIG_CHIP_SELECTGRAYCOUNTER

DAVIS640_CONFIG_CHIP_TESTADC

DAVIS640H_CONFIG_BIAS_APSCAS

Parameter address for module DAVIS640H_CONFIG_BIAS: DAVIS640H chip biases. Bias configuration values must be generated using the proper functions, which are:

- *caerBiasVDACGenerate()* for VDAC (voltage) biases.
- *caerBiasCoarseFineGenerate()* for coarse-fine (current) biases.
- *caerBiasShiftedSourceGenerate()* for shifted-source biases. See 'https://inivation.com/support/hardware/biasing/'[11] for more details.

DAVIS640H_CONFIG_BIAS_OVG1LO

DAVIS640H_CONFIG_BIAS_OVG2LO

DAVIS640H_CONFIG_BIAS_TX2OVG2HI

DAVIS640H_CONFIG_BIAS_GND07

DAVIS640H_CONFIG_BIAS_ADCTESTVOLTAGE

DAVIS640H_CONFIG_BIAS_ADCREFHIGH

DAVIS640H_CONFIG_BIAS_ADCREFLOW

DAVIS640H_CONFIG_BIAS_IFREFRBN

DAVIS640H_CONFIG_BIAS_IFTHRBN

DAVIS640H_CONFIG_BIAS_LOCALBUFBN

DAVIS640H_CONFIG_BIAS_PADFOLLBN

DAVIS640H_CONFIG_BIAS_PIXINVBN

DAVIS640H_CONFIG_BIAS_DIFFBN

DAVIS640H_CONFIG_BIAS_ONBN

DAVIS640H_CONFIG_BIAS_OFFBN

DAVIS640H_CONFIG_BIAS_PRBP

DAVIS640H_CONFIG_BIAS_PRSFBP

DAVIS640H_CONFIG_BIAS_REFRBP

DAVIS640H_CONFIG_BIAS_ARRAYBIASBUFFERBN

DAVIS640H_CONFIG_BIAS_ARRAYLOGICBUFFERBN

DAVIS640H_CONFIG_BIAS_FALLTIMEBN

DAVIS640H_CONFIG_BIAS_RISETIMEBP

DAVIS640H_CONFIG_BIAS_READOUTBUFBP

DAVIS640H_CONFIG_BIAS_APSROSFBN

DAVIS640H_CONFIG_BIAS_ADCCOMPBP

DAVIS640H_CONFIG_BIAS_DACBUFBP

DAVIS640H_CONFIG_BIAS_LCOLTIMEOUTBN

DAVIS640H_CONFIG_BIAS_AEPDBN

DAVIS640H_CONFIG_BIAS_AEPUXBP

DAVIS640H_CONFIG_BIAS_AEPUYBP

DAVIS640H_CONFIG_BIAS_BIASBUFFER

DAVIS640H_CONFIG_BIAS_SSP

DAVIS640H_CONFIG_BIAS_SSN

DAVIS640H_CONFIG_CHIP_DIGITALMUX0

Parameter address for module DAVIS640H_CONFIG_CHIP: DAVIS640H chip configuration. These are for expert control and should never be used or changed unless for advanced debugging purposes. To change the Global Shutter configuration, please use DAVIS_CONFIG_APS_GLOBAL_SHUTTER instead.

DAVIS640H_CONFIG_CHIP_DIGITALMUX1

DAVIS640H_CONFIG_CHIP_DIGITALMUX2

DAVIS640H_CONFIG_CHIP_DIGITALMUX3

DAVIS640H_CONFIG_CHIP_ANALOGMUX0

DAVIS640H_CONFIG_CHIP_ANALOGMUX1

DAVIS640H_CONFIG_CHIP_ANALOGMUX2

DAVIS640H_CONFIG_CHIP_BIASMUX0

DAVIS640H_CONFIG_CHIP_RESETCALIBNEURON

DAVIS640H_CONFIG_CHIP_TYPENCALIBNEURON

DAVIS640H_CONFIG_CHIP_RESETTESTPIXEL

DAVIS640H_CONFIG_CHIP_AERNAROW

DAVIS640H_CONFIG_CHIP_USEAOUT

DAVIS640H_CONFIG_CHIP_SELECTGRAYCOUNTER

DAVIS640H_CONFIG_CHIP_TESTADC

DAVIS640H_CONFIG_CHIP_ADJUSTOVG1LO

DAVIS640H_CONFIG_CHIP_ADJUSTOVG2LO

DAVIS640H_CONFIG_CHIP_ADJUSTTX2OVG2HI

Enums

enum **caer_davis_aps_frame_modes**

List of supported APS frame modes.

Values:

enumerator **APS_FRAME_DEFAULT**

enumerator **APS_FRAME_GRAYSCALE**

enumerator **APS_FRAME_ORIGINAL**

enum **caer_bias_shiftedsource_operating_mode**

Shifted-source bias operating mode.

Values:

enumerator **SHIFTED_SOURCE**

Standard mode.

enumerator **HI_Z**

High impedance (driven from outside).

enumerator **TIED_TO_RAIL**

Tied to ground (SSN) or VDD (SSP).

enum **caer_bias_shiftedsource_voltage_level**

Shifted-source bias voltage level.

Values:

enumerator **SPLIT_GATE**

Standard mode (200-400mV).

enumerator **SINGLE_DIODE**

Higher shifted-source voltage (one cascode).

enumerator **DOUBLE_DIODE**

Even higher shifted-source voltage (two cascodes).

Functions

LIBRARY_PUBLIC_VISIBILITY struct caer_davis_info caerDavisInfoGet (caerDeviceHandle handle)

Return basic information on the device, such as its ID, its resolution, the logic version, and so on. See the 'struct *caer_davis_info*' documentation for more details.

Parameters
handle – a valid device handle.

Returns
a copy of the device information structure if successful, an empty structure (all zeros) on failure.

LIBRARY_PUBLIC_VISIBILITY uint16_t caerBiasVDACGenerate (const struct caer_bias_vdac vdacBias)

Transform VDAC bias structure into internal integer representation, suited for sending directly to the device via *caerDeviceConfigSet()*.

Parameters
vdacBias – VDAC bias structure.

Returns
internal integer representation for device configuration.

`LIBRARY_PUBLIC_VISIBILITY struct caer_bias_vdac caerBiasVDACParse (const uint16_t vdacBias)`

Transform internal integer representation, as received by calls to *caerDeviceConfigGet()*, into a VDAC bias structure, for easier handling and understanding of the various parameters.

Parameters
`vdacBias` – internal integer representation from device.

Returns
VDAC bias structure.

`LIBRARY_PUBLIC_VISIBILITY uint16_t caerBiasCoarseFineGenerate (const struct caer_bias_coarsefine co`

Transform coarse-fine bias structure into internal integer representation, suited for sending directly to the device via *caerDeviceConfigSet()*.

Parameters
`coarseFineBias` – coarse-fine bias structure.

Returns
internal integer representation for device configuration.

`LIBRARY_PUBLIC_VISIBILITY struct caer_bias_coarsefine caerBiasCoarseFineParse (const uint16_t coars`

Transform internal integer representation, as received by calls to *caerDeviceConfigGet()*, into a coarse-fine bias structure, for easier handling and understanding of the various parameters.

Parameters
`coarseFineBias` – internal integer representation from device.

Returns
coarse-fine bias structure.

`LIBRARY_PUBLIC_VISIBILITY struct caer_bias_coarsefine caerBiasCoarseFineFromCurrent (uint32_t picoA`

Transform current value in pico-Ampere to coarse-fine bias structure. Limit is 24.8 micro-Ampere.

Parameters
`picoAmps` – desired current value in pico-Ampere.

Returns
coarse-fine bias structure.

`LIBRARY_PUBLIC_VISIBILITY uint32_t caerBiasCoarseFineToCurrent (struct caer_bias_coarsefine coarseF`

Transform coarse-fine bias structure into corresponding current value in pico-Ampere.

Parameters
`coarseFineBias` – coarse-fine bias structure.

Returns
corresponding current value in pico-Ampere.

`LIBRARY_PUBLIC_VISIBILITY uint16_t caerBiasShiftedSourceGenerate (const struct caer_bias_shiftedsou`

Transform shifted-source bias structure into internal integer representation, suited for sending directly to the device via *caerDeviceConfigSet()*.

Parameters
`shiftedSourceBias` – shifted-source bias structure.

Returns
internal integer representation for device configuration.

LIBRARY_PUBLIC_VISIBILITY struct caer_bias_shiftedsource caerBiasShiftedSourceParse (const uint16_t

Transform internal integer representation, as received by calls to *caerDeviceConfigGet()*, into a shifted-source bias structure, for easier handling and understanding of the various parameters.

Parameters
shiftedSourceBias – internal integer representation from device.

Returns
shifted-source bias structure.

LIBRARY_PUBLIC_VISIBILITY bool caerDavisROIConfigure (caerDeviceHandle handle, uint16_t startX, uint16_t startY, uint16_t endX, uint16_t endY)

Configure the APS ROI region in one step. This function guarantees efficiency and atomicity (no partial-sized results possible).

Parameters

- **handle** – a valid device handle.
- **startX** – start corner X coordinate (0, 0 is upper left of frame).
- **startY** – start corner Y coordinate (0, 0 is upper left of frame).
- **endX** – end corner X coordinate (0, 0 is upper left of frame). Must be bigger than startX.
- **endY** – end corner Y coordinate (0, 0 is upper left of frame). Must be bigger than startY.

Returns
true on success, false otherwise.

file **device.h**

#include "../libcaer.h"#include "../events/packetContainer.h*"* Common functions to access, configure and exchange data with supported devices. Also contains defines for host related configuration options.

Defines

CAER_SUPPORTED_DEVICES_NUMBER

Number of devices supported by this library. 0 - CAER_DEVICE_DVS128 1 - CAER_DEVICE_DAVIS_FX2 2 - CAER_DEVICE_DAVIS_FX3 3 - CAER_DEVICE_DYNAPSE 4 - CAER_DEVICE_DAVIS 5 - CAER_DEVICE_EDVS 6 - CAER_DEVICE_DAVIS_RPI // REMOVED 7 - CAER_DEVICE_DVS132S 8 - CAER_DEVICE_DVXPLORER 9 - CAER_DEVICE_SAMSUNG_EVK

CAER_HOST_CONFIG_DATAEXCHANGE

Module address: host-side data exchange (ring-buffer) configuration.

CAER_HOST_CONFIG_PACKETS

Module address: host-side event packets generation configuration.

[6] https://inivation.com/support/hardware/biasing/'
[7] https://inivation.com/support/hardware/biasing/'
[8] https://inivation.com/support/hardware/biasing/'
[9] https://inivation.com/support/hardware/biasing/'
[10] https://inivation.com/support/hardware/biasing/'
[11] https://inivation.com/support/hardware/biasing/'

CAER_HOST_CONFIG_LOG

Module address: host-side logging configuration.

CAER_HOST_CONFIG_DATAEXCHANGE_BUFFER_SIZE

Parameter address for module CAER_HOST_CONFIG_DATAEXCHANGE: set size of elements that can be held by the thread-safe FIFO buffer between the data transfer thread and the main thread. The default values are usually fine, only change them if you're running into lots of dropped/missing packets; you can turn on the INFO log level to see when this is the case.

CAER_HOST_CONFIG_DATAEXCHANGE_BLOCKING

Parameter address for module CAER_HOST_CONFIG_DATAEXCHANGE: when calling *caerDeviceDataGet()*, the function can either be blocking, meaning it waits until it has a valid EventPacketContainer to return, or not, meaning it returns right away. This behavior can be set with this flag. Please see the *caerDeviceDataGet()* documentation for more information on its return values.

CAER_HOST_CONFIG_DATAEXCHANGE_START_PRODUCERS

Parameter address for module CAER_HOST_CONFIG_DATAEXCHANGE: whether to start all the data producer modules on the device (DVS, APS, Mux, ...) automatically when starting the data transfer thread with *caerDeviceDataStart()* or not. If disabled, be aware you will have to start the right modules manually, which can be useful if you need precise control over which ones are running at any time.

CAER_HOST_CONFIG_DATAEXCHANGE_STOP_PRODUCERS

Parameter address for module CAER_HOST_CONFIG_DATAEXCHANGE: whether to stop all the data producer modules on the device (DVS, APS, Mux, ...) automatically when stopping the data transfer thread with *caerDeviceDataStop()* or not. If disabled, be aware you will have to stop the right modules manually, to halt the data flow, which can be useful if you need precise control over which ones are running at any time.

CAER_HOST_CONFIG_PACKETS_MAX_CONTAINER_PACKET_SIZE

Parameter address for module CAER_HOST_CONFIG_PACKETS: set the maximum number of events any of a packet container's packets may hold before it's made available to the user. Set to zero to disable. This is checked for each number of events held in each typed EventPacket that is a part of the EventPacketContainer.

CAER_HOST_CONFIG_PACKETS_MAX_CONTAINER_INTERVAL

Parameter address for module CAER_HOST_CONFIG_PACKETS: set the time interval between subsequent packet containers. Must be at least 1 microsecond. The value is in microseconds, and is checked across all types of events contained in the EventPacketContainer.

CAER_HOST_CONFIG_LOG_LEVEL

Parameter address for module CAER_HOST_CONFIG_LOG: set the log-level for this device, to be used when logging messages. Defaults to the value of the global log-level when the device was first opened.

Typedefs

typedef struct caer_device_handle *caerDeviceHandle

Pointer to an open device on which to operate.

Functions

LIBRARY_PUBLIC_VISIBILITY bool caerDeviceClose (caerDeviceHandle *handle)

Close a previously opened device and invalidate its handle.

Parameters
handle – pointer to a valid device handle. Will set handle to NULL if closing is successful, to prevent further usage of this handle for other operations.

Returns
true if closing was successful, false on errors.

LIBRARY_PUBLIC_VISIBILITY bool caerDeviceSendDefaultConfig (caerDeviceHandle handle)

Send a set of good default configuration settings to the device. This avoids users having to set every configuration option each time, especially when wanting to get going quickly or just needing to change a few settings to get to the desired operating mode.

Parameters
handle – a valid device handle.

Returns
true if sending the configuration was successful, false on errors.

LIBRARY_PUBLIC_VISIBILITY bool caerDeviceConfigSet (caerDeviceHandle handle, int8_t modAddr, uint8_t paramAddr, uint32_t param)

Set a configuration parameter to a given value.

Parameters

- handle – a valid device handle.
- modAddr – a module address, used to specify which configuration module one wants to update. Negative addresses are used for host-side configuration, while positive addresses (including zero) are used for device-side configuration.
- paramAddr – a parameter address, to select a specific parameter to update from this particular configuration module. Only positive numbers (including zero) are allowed.
- param – a configuration parameter's new value.

Returns
true if sending the configuration was successful, false on errors.

LIBRARY_PUBLIC_VISIBILITY bool caerDeviceConfigGet (caerDeviceHandle handle, int8_t modAddr, uint8_t paramAddr, uint32_t *param)

Get the value of a configuration parameter.

Parameters

- handle – a valid device handle.

- **modAddr** – a module address, used to specify which configuration module one wants to query. Negative addresses are used for host-side configuration, while positive addresses (including zero) are used for device-side configuration.
- **paramAddr** – a parameter address, to select a specific parameter to query from this particular configuration module. Only positive numbers (including zero) are allowed.
- **param** – a pointer to an integer, in which to store the configuration parameter's current value. The integer will always be either set to zero (on failure), or to the current value (on success).

Returns
true if getting the configuration was successful, false on errors.

LIBRARY_PUBLIC_VISIBILITY bool caerDeviceConfigGet64 (caerDeviceHandle handle, int8_t modAddr, uint8_t paramAddr, uint64_t *param)

Get the value of a 64bit configuration parameter. This is for special read-only configuration parameters only! Use only when required by the parameter's documentation!

Parameters

- **handle** – a valid device handle.
- **modAddr** – a module address, used to specify which configuration module one wants to query. Negative addresses are used for host-side configuration, while positive addresses (including zero) are used for device-side configuration.
- **paramAddr** – a parameter address, to select a specific parameter to query from this particular configuration module. Only positive numbers (including zero) are allowed.
- **param** – a pointer to a 64bit integer, in which to store the configuration parameter's current value. The integer will always be either set to zero (on failure), or to the current value (on success).

Returns
true if getting the configuration was successful, false on errors.

LIBRARY_PUBLIC_VISIBILITY bool caerDeviceDataStart (caerDeviceHandle handle, void(*dataNotifyIncrease)(void *ptr), void(*dataNotifyDecrease)(void *ptr), void *dataNotifyUserPtr, void(*dataShutdownNotify)(void *ptr), void *dataShutdownUserPtr)

Start getting data from the device, setting up the data transfers and starting the data producers (see CAER_HOST_CONFIG_DATAEXCHANGE_START_PRODUCERS). Supports notification of new data and exceptional shutdown events via user-defined call-backs.

Parameters

- **handle** – a valid device handle.
- **dataNotifyIncrease** – function pointer, called every time a new piece of data available and has been put in the FIFO buffer for consumption. dataNotifyUserPtr will be passed as parameter to the function.
- **dataNotifyDecrease** – function pointer, called every time a new piece of data has been consumed from the FIFO buffer inside *caerDeviceDataGet()*. dataNotifyUserPtr will be passed as parameter to the function.
- **dataNotifyUserPtr** – pointer that will be passed to the dataNotifyIncrease and dataNotifyDecrease functions. Can be NULL.

- **dataShutdownNotify** – function pointer, called on exceptional shut-down of the data transfers. This is used to detect exceptional shut-downs that do not come from calling *caerDeviceDataStop()*, such as when the device is disconnected or all data transfers fail.
- **dataShutdownUserPtr** – pointer that will be passed to the dataShutdownNotify function. Can be NULL.

Returns
true if starting the data transfer was successful, false on errors.

LIBRARY_PUBLIC_VISIBILITY bool caerDeviceDataStop (caerDeviceHandle handle)

Stop getting data from the device, shutting down the data transfers and stopping the data producers (see CAER_HOST_CONFIG_DATAEXCHANGE_STOP_PRODUCERS). This normal shut-down will not generate a notification (see *caerDeviceDataStart()*).

Parameters
handle – a valid device handle.

Returns
true if stopping the data transfer was successful, false on errors.

LIBRARY_PUBLIC_VISIBILITY caerEventPacketContainer caerDeviceDataGet (caerDeviceHandle handle)

Get an event packet container, which contains events of various types generated by the device, for further processing. The returned data structures are allocated in memory and will need to be freed. The caerEventPacketContainerFree() function can be used to correctly free the full container memory. For single caerEventPackets, just use free(). This function can be made blocking with the CAER_HOST_CONFIG_DATAEXCHANGE_BLOCKING configuration parameter. By default it is non-blocking.

Parameters
handle – a valid device handle.

Returns
a valid event packet container. NULL will be returned on errors, such as exceptional device shutdown, or when there is no container available in non-blocking mode. Always check this return value!

file device_discover.h

*#include "*davis.h*"#include "*dvs128.h*"#include "*dvs132s.h*"#include "*dvxplorer.h*"#include "*dynapse.h*"#include "*edvs.h*"#include "*samsung_evk.h*"* Functions to discover supported devices attached to the current host system, and then open them.

Defines

CAER_DEVICE_DISCOVER_ALL

Define for special value to discover all device types.

Typedefs

typedef struct *caer_device_discovery_result* ***caerDeviceDiscoveryResult**

Pointer to result of a device discovery operation.

Functions

LIBRARY_PUBLIC_VISIBILITY ssize_t caerDeviceDiscover (int16_t deviceType, caerDeviceDiscoveryResult *discoveredDevices)

Discover all supported devices that are accessible on this system. Use -1 as 'deviceType' to search for any device, or an actual device type ID to only search for matches of that specific type.

Parameters

- **deviceType** – type of device to search for, use -1 for any.
- **discoveredDevices** – pointer to array of results, memory will be allocated for it automatically. On error, the pointer is set to NULL. Remember to free() the memory once done!

Returns

number of discovered devices, 0 if no device could be found; or -1 if an error occurred.

LIBRARY_PUBLIC_VISIBILITY caerDeviceHandle caerDeviceDiscoverOpen (uint16_t deviceID, caerDeviceDiscoveryResult discoveredDevice)

Open a specific device based on information returned by *caerDeviceDiscover()*, then assign an ID to it and return a handle for further usage.

Parameters

- **deviceID** – a unique ID to identify the device from others. Will be used as the source for EventPackets being generated from its data.
- **discoveredDevice** – pointer to the result of a device discovery operation. Uniquely identifies a particular device.

Returns

a valid device handle that can be used with the other libcaer functions, or NULL on error. Always check for this!

file **dvs128.h**

#include "../events/polarity.h" #include "../events/special.h" #include "usb.h" DVS128 specific configuration defines and information structures.

Defines

CAER_DEVICE_DVS128

Device type definition for iniVation DVS128.

DVS128_CONFIG_DVS

Module address: device-side DVS configuration.

DVS128_CONFIG_BIAS

Module address: device-side chip bias generator configuration.

DVS128_CONFIG_DVS_RUN

Parameter address for module DVS128_CONFIG_DVS: run the DVS chip and generate polarity event data.

DVS128_CONFIG_DVS_TIMESTAMP_RESET

Parameter address for module DVS128_CONFIG_DVS: reset the time-stamp counter of the device. This is a temporary configuration switch and will reset itself right away.

DVS128_CONFIG_DVS_ARRAY_RESET

Parameter address for module DVS128_CONFIG_DVS: reset the whole DVS pixel array. This is a temporary configuration switch and will reset itself right away.

DVS128_CONFIG_DVS_TS_MASTER

Parameter address for module DVS128_CONFIG_DVS: control if this DVS is a timestamp master device. Default is enabled.

DVS128_CONFIG_BIAS_CAS

Parameter address for module DVS128_CONFIG_BIAS: First stage amplifier cascode bias. See 'https://inivation.com/support/hardware/biasing/'[12] for more details.

DVS128_CONFIG_BIAS_INJGND

Parameter address for module DVS128_CONFIG_BIAS: Injected ground bias. See 'https://inivation.com/support/hardware/biasing/'[13] for more details.

DVS128_CONFIG_BIAS_REQPD

Parameter address for module DVS128_CONFIG_BIAS: Pull down on chip request (AER). See 'https://inivation.com/support/hardware/biasing/'[14] for more details.

DVS128_CONFIG_BIAS_PUX

Parameter address for module DVS128_CONFIG_BIAS: Pull up on request from X arbiter (AER). See 'https://inivation.com/support/hardware/biasing/'[15] for more details.

DVS128_CONFIG_BIAS_DIFFOFF

Parameter address for module DVS128_CONFIG_BIAS: Off events threshold bias. See 'https://inivation.com/support/hardware/biasing/'[16] for more details.

DVS128_CONFIG_BIAS_REQ

Parameter address for module DVS128_CONFIG_BIAS: Pull down for passive load inverters in digital AER pixel circuitry. See 'https://inivation.com/support/hardware/biasing/'[17] for more details.

DVS128_CONFIG_BIAS_REFR

Parameter address for module DVS128_CONFIG_BIAS: Refractory period bias. See 'https://inivation.com/support/hardware/biasing/'[18] for more details.

DVS128_CONFIG_BIAS_PUY

Parameter address for module DVS128_CONFIG_BIAS: Pull up on request from Y arbiter (AER). See 'https://inivation.com/support/hardware/biasing/'[19] for more details.

DVS128_CONFIG_BIAS_DIFFON

Parameter address for module DVS128_CONFIG_BIAS: On events threshold bias. See 'https://inivation.com/support/hardware/biasing/'[20] for more details.

DVS128_CONFIG_BIAS_DIFF

Parameter address for module DVS128_CONFIG_BIAS: Differential (second stage amplifier) bias. See 'https://inivation.com/support/hardware/biasing/'[21] for more details.

DVS128_CONFIG_BIAS_FOLL

Parameter address for module DVS128_CONFIG_BIAS: Source follower bias. See 'https://inivation.com/support/hardware/biasing/'[22] for more details.

DVS128_CONFIG_BIAS_PR

Parameter address for module DVS128_CONFIG_BIAS: Photoreceptor bias. See 'https://inivation.com/support/hardware/biasing/'[23] for more details.

Functions

LIBRARY_PUBLIC_VISIBILITY struct caer_dvs128_info caerDVS128InfoGet (caerDeviceHandle handle)

Return basic information on the device, such as its ID, its resolution, the logic version, and so on. See the 'struct *caer_dvs128_info*' documentation for more details.

Parameters
handle – a valid device handle.

Returns
a copy of the device information structure if successful, an empty structure (all zeros) on failure.

file **dvs132s.h**

#include "../events/polarity.h"#include "../events/special.h"#include "imu_support.h"#include "usb.h" DVS132S specific configuration defines and information structures.

[12] https://inivation.com/support/hardware/biasing/'
[13] https://inivation.com/support/hardware/biasing/'
[14] https://inivation.com/support/hardware/biasing/'
[15] https://inivation.com/support/hardware/biasing/'
[16] https://inivation.com/support/hardware/biasing/'
[17] https://inivation.com/support/hardware/biasing/'
[18] https://inivation.com/support/hardware/biasing/'
[19] https://inivation.com/support/hardware/biasing/'
[20] https://inivation.com/support/hardware/biasing/'
[21] https://inivation.com/support/hardware/biasing/'
[22] https://inivation.com/support/hardware/biasing/'
[23] https://inivation.com/support/hardware/biasing/'

Defines

CAER_DEVICE_DVS132S

Device type definition for iniVation DVS132S.

DVS132S_CHIP_ID

DVS132S chip identifier. 104x132, synchronous readout.

DVS132S_CONFIG_MUX

Module address: device-side Multiplexer configuration. The Multiplexer is responsible for mixing, timestamping and outputting (via USB) the various event types generated by the device. It is also responsible for timestamp generation and synchronization.

DVS132S_CONFIG_DVS

Module address: device-side DVS configuration. The DVS state machine interacts with the DVS chip and gets the polarity events from it. It supports various configurable delays, as well as advanced filtering capabilities on the polarity events.

DVS132S_CONFIG_IMU

Module address: device-side IMU (Inertial Measurement Unit) configuration. The IMU module connects to the external IMU chip and sends data on the device's movement in space. It can configure various options on the external chip, such as accelerometer range or gyroscope refresh rate.

DVS132S_CONFIG_EXTINPUT

Module address: device-side External Input (signal detector/generator) configuration. The External Input module is used to detect external signals on the external input jack and inject an event into the event stream when this happens. It can detect pulses of a specific length or rising and falling edges. On some systems, a signal generator module is also present, which can generate PWM-like pulsed signals with configurable timing.

DVS132S_CONFIG_BIAS

Module address: device-side chip bias configuration. This state machine is responsible for configuring the chip's bias generator.

DVS132S_CONFIG_SYSINFO

Module address: device-side system information. The system information module provides various details on the device, such as currently installed logic revision or clock speeds. All its parameters are read-only. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_dvs132s_info*' documentation for more details on what information is available.

DVS132S_CONFIG_USB

Module address: device-side USB output configuration. The USB output module forwards the data from the device and the FPGA/CPLD to the USB chip, usually a Cypress FX2 or FX3.

DVS132S_CONFIG_MUX_RUN

Parameter address for module DVS132S_CONFIG_MUX: run the Multiplexer state machine, which is responsible for mixing the various event types at the device level, timestamping them and outputting them via USB or other connectors.

DVS132S_CONFIG_MUX_TIMESTAMP_RUN

Parameter address for module DVS132S_CONFIG_MUX: run the Timestamp Generator inside the Multiplexer state machine, which will provide microsecond accurate timestamps to the events passing through.

DVS132S_CONFIG_MUX_TIMESTAMP_RESET

Parameter address for module DVS132S_CONFIG_MUX: reset the Timestamp Generator to zero. This also sends a reset pulse to all connected slave devices, resetting their timestamp too.

DVS132S_CONFIG_MUX_RUN_CHIP

Parameter address for module DVS132S_CONFIG_MUX: power up the chip's bias generator, enabling the chip to work.

DVS132S_CONFIG_MUX_DROP_EXTINPUT_ON_TRANSFER_STALL

Parameter address for module DVS132S_CONFIG_MUX: drop External Input events if the USB output FIFO is full, instead of having them pile up at the input FIFOs.

DVS132S_CONFIG_MUX_DROP_DVS_ON_TRANSFER_STALL

Parameter address for module DVS132S_CONFIG_MUX: drop DVS events if the USB output FIFO is full, instead of having them pile up at the input FIFOs.

DVS132S_CONFIG_MUX_HAS_STATISTICS

Parameter address for module DVS132S_CONFIG_MUX: read-only parameter, information about the presence of the statistics feature. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get this information.

DVS132S_CONFIG_MUX_STATISTICS_EXTINPUT_DROPPED

Parameter address for module DVS132S_CONFIG_MUX: read-only parameter, representing the number of dropped External Input events on the device due to full USB buffers. This is a 64bit value, and should always be read using the function: *caerDeviceConfigGet64()*.

DVS132S_CONFIG_MUX_STATISTICS_DVS_DROPPED

Parameter address for module DVS132S_CONFIG_MUX: read-only parameter, representing the number of dropped DVS events on the device due to full USB buffers. This is a 64bit value, and should always be read using the function: *caerDeviceConfigGet64()*.

DVS132S_CONFIG_DVS_SIZE_COLUMNS

Parameter address for module DVS132S_CONFIG_DVS: read-only parameter, contains the X axis resolution of the DVS events returned by the camera. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get proper size information that already considers the rotation and orientation settings.

DVS132S_CONFIG_DVS_SIZE_ROWS

Parameter address for module DVS132S_CONFIG_DVS: read-only parameter, contains the Y axis resolution of the DVS events returned by the camera. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get proper size information that already considers the rotation and orientation settings.

DVS132S_CONFIG_DVS_ORIENTATION_INFO

Parameter address for module DVS132S_CONFIG_DVS: read-only parameter, contains information on the orientation of the X/Y axes, whether they should be inverted or not on the host when parsing incoming events. Bit 2: dvsInvertXY Bit 1: reserved Bit 0: reserved This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get proper size information that already considers the rotation and orientation settings.

DVS132S_CONFIG_DVS_RUN

Parameter address for module DVS132S_CONFIG_DVS: run the DVS state machine and read out polarity events from the chip.

DVS132S_CONFIG_DVS_WAIT_ON_TRANSFER_STALL

Parameter address for module DVS132S_CONFIG_DVS: if the output FIFO for this module is full, stall the chip readout and wait until it's free again, instead of just continuing reading and dropping the resulting events.

DVS132S_CONFIG_DVS_FILTER_AT_LEAST_2_UNSIGNED

Parameter address for module DVS132S_CONFIG_DVS:

DVS132S_CONFIG_DVS_FILTER_NOT_ALL_4_UNSIGNED

Parameter address for module DVS132S_CONFIG_DVS:

DVS132S_CONFIG_DVS_FILTER_AT_LEAST_2_SIGNED

Parameter address for module DVS132S_CONFIG_DVS:

DVS132S_CONFIG_DVS_FILTER_NOT_ALL_4_SIGNED

Parameter address for module DVS132S_CONFIG_DVS:

DVS132S_CONFIG_DVS_RESTART_TIME

Parameter address for module DVS132S_CONFIG_DVS: 7 bits, Time unit: 1us. Max: ~125us.

DVS132S_CONFIG_DVS_CAPTURE_INTERVAL

Parameter address for module DVS132S_CONFIG_DVS: 21 bits, Time unit: 1us. Max: ~2s.

DVS132S_CONFIG_DVS_ROW_ENABLE_31_TO_0

Parameter address for module DVS132S_CONFIG_DVS:

DVS132S_CONFIG_DVS_ROW_ENABLE_63_TO_32

Parameter address for module DVS132S_CONFIG_DVS:

DVS132S_CONFIG_DVS_ROW_ENABLE_65_TO_64

Parameter address for module DVS132S_CONFIG_DVS:

DVS132S_CONFIG_DVS_COLUMN_ENABLE_31_TO_0

Parameter address for module DVS132S_CONFIG_DVS:

DVS132S_CONFIG_DVS_COLUMN_ENABLE_51_TO_32

Parameter address for module DVS132S_CONFIG_DVS:

DVS132S_CONFIG_DVS_HAS_STATISTICS

Parameter address for module DVS132S_CONFIG_DVS: read-only parameter, information about the presence of the statistics feature. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get this information.

DVS132S_CONFIG_DVS_STATISTICS_TRANSACTIONS_SUCCESS

Parameter address for module DVS132S_CONFIG_DVS: read-only parameter, representing the number of event transactions completed successfully on the device. This is a 64bit value, and should always be read using the function: *caerDeviceConfigGet64()*.

DVS132S_CONFIG_DVS_STATISTICS_TRANSACTIONS_SKIPPED

Parameter address for module DVS132S_CONFIG_DVS: read-only parameter, representing the number of dropped transaction sequences on the device due to full buffers. This is a 64bit value, and should always be read using the function: *caerDeviceConfigGet64()*.

DVS132S_CONFIG_DVS_STATISTICS_TRANSACTIONS_ERRORED

Parameter address for module DVS132S_CONFIG_DVS: read-only parameter, representing the number of erroneous transaction sequences on the device due problems in the address or polarities returned by the chip.

DVS132S_CONFIG_IMU_TYPE

Parameter address for module DVS132S_CONFIG_IMU: read-only parameter, contains information on the type of IMU chip being used in this device: 0 - no IMU present 3 - Bosch BMI 160 This is reserved for internal use and should not be used by anything other than libcaer.

DVS132S_CONFIG_IMU_ORIENTATION_INFO

Parameter address for module DVS132S_CONFIG_IMU: read-only parameter, contains information on the orientation of the X/Y/Z axes, whether they should be flipped or not on the host when parsing incoming IMU data samples. Bit 2: imuFlipX Bit 1: imuFlipY Bit 0: imuFlipZ This is reserved for internal use and should not be used by anything other than libcaer. Generated IMU events are already properly flipped when returned to the user.

DVS132S_CONFIG_IMU_RUN_ACCELEROMETER

Parameter address for module DVS132S_CONFIG_IMU: enable the IMU's accelerometer. This takes the IMU chip out of sleep.

DVS132S_CONFIG_IMU_RUN_GYROSCOPE

Parameter address for module DVS132S_CONFIG_IMU: enable the IMU's gyroscope. This takes the IMU chip out of sleep.

DVS132S_CONFIG_IMU_RUN_TEMPERATURE

Parameter address for module DVS132S_CONFIG_IMU: enable the IMU's temperature sensor. This takes the IMU chip out of sleep.

DVS132S_CONFIG_IMU_ACCEL_DATA_RATE

Parameter address for module DVS132S_CONFIG_IMU: 8 settings: 0 - 12.5 Hz 1 - 25 Hz 2 - 50 Hz 3 - 100 Hz 4 - 200 Hz 5 - 400 Hz 6 - 800 Hz 7 - 1600 Hz

DVS132S_CONFIG_IMU_ACCEL_FILTER

Parameter address for module DVS132S_CONFIG_IMU: 3 settings: 0 - OSR4 1 - OSR2 2 - Normal

DVS132S_CONFIG_IMU_ACCEL_RANGE

Parameter address for module DVS132S_CONFIG_IMU: 4 settings: 0 - +- 2g 1 - +- 4g 2 - +- 8g 3 - +- 16g

DVS132S_CONFIG_IMU_GYRO_DATA_RATE

Parameter address for module DVS132S_CONFIG_IMU: 8 settings: 0 - 25 Hz 1 - 50 Hz 2 - 100 Hz 3 - 200 Hz 4 - 400 Hz 5 - 800 Hz 6 - 1600 Hz 7 - 3200 Hz

DVS132S_CONFIG_IMU_GYRO_FILTER

Parameter address for module DVS132S_CONFIG_IMU: 3 settings: 0 - OSR4 1 - OSR2 2 - Normal

DVS132S_CONFIG_IMU_GYRO_RANGE

Parameter address for module DVS132S_CONFIG_IMU: 5 settings: 0 - +- 2000°/s 1 - +- 1000°/s 2 - +- 500°/s 3 - +- 250°/s 4 - +- 125°/s

DVS132S_CONFIG_EXTINPUT_RUN_DETECTOR

Parameter address for module DVS132S_CONFIG_EXTINPUT: enable the signal detector module. It generates events when it sees certain types of signals, such as edges or pulses of a defined length, on the SIGNAL pin of the INPUT synchronization connector. This can be useful to inject events into the event stream in response to external stimuli or controls, such as turning on a LED lamp.

DVS132S_CONFIG_EXTINPUT_DETECT_RISING_EDGES

Parameter address for module DVS132S_CONFIG_EXTINPUT: send a special EXTERNAL_INPUT_RISING_EDGE event when a rising edge is detected (transition from low voltage to high).

DVS132S_CONFIG_EXTINPUT_DETECT_FALLING_EDGES

Parameter address for module DVS132S_CONFIG_EXTINPUT: send a special EXTERNAL_INPUT_FALLING_EDGE event when a falling edge is detected (transition from high voltage to low).

DVS132S_CONFIG_EXTINPUT_DETECT_PULSES

Parameter address for module DVS132S_CONFIG_EXTINPUT: send a special EXTERNAL_INPUT_PULSE event when a pulse, of a specified, configurable polarity and length, is detected. See DVS132S_CONFIG_EXTINPUT_DETECT_PULSE_POLARITY and DVS132S_CONFIG_EXTINPUT_DETECT_PULSE_LENGTH for more details.

DVS132S_CONFIG_EXTINPUT_DETECT_PULSE_POLARITY

Parameter address for module DVS132S_CONFIG_EXTINPUT: the polarity the pulse must exhibit to be detected as such. '1' means active high; a pulse will start when the signal goes from low to high and will continue to be seen as the same pulse as long as it stays high. '0' means active low; a pulse will start when the signal goes from high to low and will continue to be seen as the same pulse as long as it stays low.

DVS132S_CONFIG_EXTINPUT_DETECT_PULSE_LENGTH

Parameter address for module DVS132S_CONFIG_EXTINPUT: the minimal length that a pulse must have to trigger the sending of a special event. This is measured in cycles at LogicClock frequency (see 'struct *caer_davis_info*' for details on how to get the frequency).

DVS132S_CONFIG_EXTINPUT_HAS_GENERATOR

Parameter address for module DVS132S_CONFIG_EXTINPUT: read-only parameter, information about the presence of the signal generator feature. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get this information.

DVS132S_CONFIG_EXTINPUT_RUN_GENERATOR

Parameter address for module DVS132S_CONFIG_EXTINPUT: enable the signal generator module. It generates a PWM-like signal based on configurable parameters and outputs it on the OUT JACK signal.

DVS132S_CONFIG_EXTINPUT_GENERATE_PULSE_POLARITY

Parameter address for module DVS132S_CONFIG_EXTINPUT: polarity of the PWM-like signal to be generated. '1' means active high, '0' means active low.

DVS132S_CONFIG_EXTINPUT_GENERATE_PULSE_INTERVAL

Parameter address for module DVS132S_CONFIG_EXTINPUT: the interval between the start of two consecutive pulses, expressed in cycles at LogicClock frequency (see 'struct *caer_davis_info*' for details on how to get the frequency). This must be bigger or equal to DVS132S_CONFIG_EXTINPUT_GENERATE_PULSE_LENGTH. To generate a signal with 50% duty cycle, this would have to be exactly double of DVS132S_CONFIG_EXTINPUT_GENERATE_PULSE_LENGTH.

DVS132S_CONFIG_EXTINPUT_GENERATE_PULSE_LENGTH

Parameter address for module DVS132S_CONFIG_EXTINPUT: the length a pulse stays active, expressed in cycles at LogicClock frequency (see 'struct *caer_davis_info*' for details on how to get the frequency). This must be smaller or equal to DVS132S_CONFIG_EXTINPUT_GENERATE_PULSE_INTERVAL. To generate a signal with 50% duty cycle, this would have to be exactly half of DVS132S_CONFIG_EXTINPUT_GENERATE_PULSE_INTERVAL.

DVS132S_CONFIG_EXTINPUT_GENERATE_INJECT_ON_RISING_EDGE

Parameter address for module DVS132S_CONFIG_EXTINPUT: enables event injection when a rising edge occurs in the generated signal; a special event EXTERNAL_GENERATOR_RISING_EDGE is emitted into the event stream.

DVS132S_CONFIG_EXTINPUT_GENERATE_INJECT_ON_FALLING_EDGE

Parameter address for module DVS132S_CONFIG_EXTINPUT: enables event injection when a falling edge occurs in the generated signal; a special event EXTERNAL_GENERATOR_FALLING_EDGE is emitted into the event stream.

DVS132S_CONFIG_SYSINFO_LOGIC_VERSION

Parameter address for module DVS132S_CONFIG_SYSINFO: read-only parameter, the version of the logic currently running on the device's FPGA/CPLD. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_dvs132s_info*' documentation to get this information.

DVS132S_CONFIG_SYSINFO_CHIP_IDENTIFIER

Parameter address for module DVS132S_CONFIG_SYSINFO: read-only parameter, an integer used to identify the different types of sensor chips used on the device. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_dvs132s_info*' documentation to get this information.

DVS132S_CONFIG_SYSINFO_DEVICE_IS_MASTER

Parameter address for module DVS132S_CONFIG_SYSINFO: read-only parameter, whether the device is currently a timestamp master or slave when synchronizing multiple devices together. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_dvs132s_info*' documentation to get this information.

DVS132S_CONFIG_SYSINFO_LOGIC_CLOCK

Parameter address for module DVS132S_CONFIG_SYSINFO: read-only parameter, the frequency in MHz at which the main FPGA/CPLD logic is running. This is reserved for internal use and should not be used by anything other than libcaer.

DVS132S_CONFIG_SYSINFO_USB_CLOCK

Parameter address for module DVS132S_CONFIG_SYSINFO: read-only parameter, the frequency in MHz at which the FPGA/CPLD logic related to USB data transmission is running. This is reserved for internal use and should not be used by anything other than libcaer.

DVS132S_CONFIG_SYSINFO_CLOCK_DEVIATION

Parameter address for module DVS132S_CONFIG_SYSINFO: read-only parameter, the deviation factor for the clocks. Due to how FX3 generates the clocks, which are then used by FPGA/CPLD, they are not integers but have a fractional part. This is reserved for internal use and should not be used by anything other than libcaer.

DVS132S_CONFIG_SYSINFO_LOGIC_PATCH

Parameter address for module DVS132S_CONFIG_SYSINFO: read-only parameter, the patch version of the logic currently running on the device's FPGA/CPLD. This is reserved for internal use and should not be used by anything other than libcaer.

DVS132S_CONFIG_USB_RUN

Parameter address for module DVS132S_CONFIG_USB: enable the USB FIFO module, which transfers the data from the FPGA/CPLD to the USB chip, to be then sent to the host. Turning this off will suppress any USB data communication!

DVS132S_CONFIG_USB_EARLY_PACKET_DELAY

Parameter address for module DVS132S_CONFIG_USB: the time delay after which a packet of data is committed to USB, even if it is not full yet (short USB packet). The value is in 125µs time-slices, corresponding to how USB schedules its operations (a value of 4 for example would mean waiting at most 0.5ms until sending a short USB packet to the host).

DVS132S_CONFIG_BIAS_PRBP

Parameter address for module DVS132S_CONFIG_BIAS: DVS132S chip biases. Bias configuration values must be generated using the proper functions, which are:

- *caerBiasCoarseFine1024Generate()* for coarse-fine (current) biases. See 'https://inivation.com/support/hardware/biasing/'[24] for more details.

DVS132S_CONFIG_BIAS_PRSFBP

DVS132S_CONFIG_BIAS_BLPUBP

DVS132S_CONFIG_BIAS_BIASBUFBP

DVS132S_CONFIG_BIAS_OFFBN

DVS132S_CONFIG_BIAS_DIFFBN

DVS132S_CONFIG_BIAS_ONBN

DVS132S_CONFIG_BIAS_CASBN

DVS132S_CONFIG_BIAS_DPBN

DVS132S_CONFIG_BIAS_BIASBUFBN

DVS132S_CONFIG_BIAS_ABUFBN

Functions

LIBRARY_PUBLIC_VISIBILITY struct caer_dvs132s_info caerDVS132SInfoGet (caerDeviceHandle handle)

Return basic information on the device, such as its ID, its resolution, the logic version, and so on. See the 'struct *caer_dvs132s_info*' documentation for more details.

Parameters
handle – a valid device handle.

Returns
a copy of the device information structure if successful, an empty structure (all zeros) on failure.

LIBRARY_PUBLIC_VISIBILITY uint32_t caerBiasCoarseFine1024Generate (struct caer_bias_coarsefine1024

Transform simplified coarse-fine bias structure into internal integer representation, suited for sending directly to the device via *caerDeviceConfigSet()*.

Parameters
coarseFine1024Bias – coarse-fine bias structure.

Returns
internal integer representation for device configuration.

LIBRARY_PUBLIC_VISIBILITY struct caer_bias_coarsefine1024 caerBiasCoarseFine1024Parse (uint32_t coa

Transform internal integer representation, as received by calls to *caerDeviceConfigGet()*, into a simplified coarse-fine bias structure, for easier handling and understanding of the various parameters.

Parameters
coarseFine1024Bias – internal integer representation from device.

Returns
coarse-fine bias structure.

`LIBRARY_PUBLIC_VISIBILITY struct caer_bias_coarsefine1024 caerBiasCoarseFine1024FromCurrent (uint32`

Transform current value in pico-Ampere to coarse-fine bias structure. Limit is 1.0 micro-Ampere.

Parameters
picoAmps – desired current value in pico-Ampere.

Returns
coarse-fine bias structure.

`LIBRARY_PUBLIC_VISIBILITY uint32_t caerBiasCoarseFine1024ToCurrent (struct caer_bias_coarsefine1024`

Transform coarse-fine bias structure into corresponding current value in pico-Ampere.

Parameters
coarseFine1024Bias – coarse-fine bias structure.

Returns
corresponding current value in pico-Ampere.

file **dvxplorer.h**

#include *"../events/imu6.h"#include* *"../events/polarity.h"#include* *"../events/special.h"#include* *"imu_support.h"#include* *"usb.h"* DVXPLORER specific configuration defines and information structures.

Defines

`CAER_DEVICE_DVXPLORER`

Device type definition for iniVation DVXplorer.

`DVXPLORER_CHIP_ID`

Samsung chip identifier. 640x480, semi-synchronous readout.

`DVXPLORER_LITE_CHIP_ID`

Samsung chip identifier. 320x240, semi-synchronous readout.

`DVX_MUX`

Module address: device-side Multiplexer configuration. The Multiplexer is responsible for mixing, timestamping and outputting (via USB) the various event types generated by the device. It is also responsible for timestamp generation and synchronization.

`DVX_DVS`

Module address: device-side DVS configuration. The DVS state machine interacts with the DVS chip and gets the polarity events from it. It supports various configurable delays, as well as advanced filtering capabilities on the polarity events.

[24] https://inivation.com/support/hardware/biasing/'

DVX_IMU

Module address: device-side IMU (Inertial Measurement Unit) configuration. The IMU module connects to the external IMU chip and sends data on the device's movement in space. It can configure various options on the external chip, such as accelerometer range or gyroscope refresh rate.

DVX_EXTINPUT

Module address: device-side External Input (signal detector/generator) configuration. The External Input module is used to detect external signals on the external input jack and inject an event into the event stream when this happens. It can detect pulses of a specific length or rising and falling edges. On some systems, a signal generator module is also present, which can generate PWM-like pulsed signals with configurable timing.

DVX_SYSINFO

Module address: device-side system information. The system information module provides various details on the device, such as currently installed logic revision or clock speeds. All its parameters are read-only. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_dvx_info*' documentation for more details on what information is available.

DVX_USB

Module address: device-side USB output configuration. The USB output module forwards the data from the device and the FPGA/CPLD to the USB chip, usually a Cypress FX2 or FX3.

DVX_MUX_RUN

Parameter address for module DVX_MUX: run the Multiplexer state machine, which is responsible for mixing the various event types at the device level, timestamping them and outputting them via USB or other connectors.

DVX_MUX_TIMESTAMP_RUN

Parameter address for module DVX_MUX: run the Timestamp Generator inside the Multiplexer state machine, which will provide microsecond accurate timestamps to the events passing through.

DVX_MUX_TIMESTAMP_RESET

Parameter address for module DVX_MUX: reset the Timestamp Generator to zero. This also sends a reset pulse to all connected slave devices, resetting their timestamp too.

DVX_MUX_RUN_CHIP

Parameter address for module DVX_MUX: power up the chip's bias generator, enabling the chip to work.

DVX_MUX_DROP_EXTINPUT_ON_TRANSFER_STALL

Parameter address for module DVX_MUX: drop External Input events if the USB output FIFO is full, instead of having them pile up at the input FIFOs.

DVX_MUX_DROP_DVS_ON_TRANSFER_STALL

Parameter address for module DVX_MUX: drop DVS events if the USB output FIFO is full, instead of having them pile up at the input FIFOs.

DVX_MUX_HAS_STATISTICS

Parameter address for module DVX_MUX: read-only parameter, information about the presence of the

statistics feature. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get this information.

DVX_MUX_STATISTICS_EXTINPUT_DROPPED

Parameter address for module DVX_MUX: read-only parameter, representing the number of dropped External Input events on the device due to full USB buffers. This is a 64bit value, and should always be read using the function: *caerDeviceConfigGet64()*.

DVX_MUX_STATISTICS_DVS_DROPPED

Parameter address for module DVX_MUX: read-only parameter, representing the number of dropped DVS events on the device due to full USB buffers. This is a 64bit value, and should always be read using the function: *caerDeviceConfigGet64()*.

DVX_DVS_SIZE_COLUMNS

Parameter address for module DVX_DVS: read-only parameter, contains the X axis resolution of the DVS events returned by the camera. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get proper size information that already considers the rotation and orientation settings.

DVX_DVS_SIZE_ROWS

Parameter address for module DVX_DVS: read-only parameter, contains the Y axis resolution of the DVS events returned by the camera. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get proper size information that already considers the rotation and orientation settings.

DVX_DVS_ORIENTATION_INFO

Parameter address for module DVX_DVS: read-only parameter, contains information on the orientation of the X/Y axes, whether they should be inverted or not on the host when parsing incoming events. Bit 2: dvsInvertXY Bit 1: reserved Bit 0: reserved This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get proper size information that already considers the rotation and orientation settings.

DVX_DVS_RUN

Parameter address for module DVX_DVS: run the DVS state machine and read out polarity events from the chip.

DVX_DVS_HAS_STATISTICS

Parameter address for module DVX_DVS: read-only parameter, information about the presence of the statistics feature. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get this information.

DVX_DVS_STATISTICS_COLUMN

Parameter address for module DVX_DVS: read-only parameter, representing the number of column transactions completed successfully on the device. This is a 64bit value, and should always be read using the function: *caerDeviceConfigGet64()*.

DVX_DVS_STATISTICS_GROUP

Parameter address for module DVX_DVS: read-only parameter, representing the number of SGroup/MGroup transactions completed successfully on the device. This is a 64bit value, and should always be read using the function: *caerDeviceConfigGet64()*.

DVX_DVS_STATISTICS_DROPPED_COLUMN

Parameter address for module DVX_DVS: read-only parameter, representing the number of dropped column transactions on the device. This is a 64bit value, and should always be read using the function: *caerDeviceConfigGet64()*.

DVX_DVS_STATISTICS_DROPPED_GROUP

Parameter address for module DVX_DVS: read-only parameter, representing the number of dropped SGroup/MGroup transactions on the device. This is a 64bit value, and should always be read using the function: *caerDeviceConfigGet64()*.

DVX_IMU_TYPE

Parameter address for module DVX_IMU: read-only parameter, contains information on the type of IMU chip being used in this device: 0 - no IMU present 3 - Bosch BMI 160 This is reserved for internal use and should not be used by anything other than libcaer.

DVX_IMU_ORIENTATION_INFO

Parameter address for module DVX_IMU: read-only parameter, contains information on the orientation of the X/Y/Z axes, whether they should be flipped or not on the host when parsing incoming IMU data samples. Bit 2: imuFlipX Bit 1: imuFlipY Bit 0: imuFlipZ This is reserved for internal use and should not be used by anything other than libcaer. Generated IMU events are already properly flipped when returned to the user.

DVX_IMU_RUN_ACCELEROMETER

Parameter address for module DVX_IMU: enable the IMU's accelerometer. This takes the IMU chip out of sleep.

DVX_IMU_RUN_GYROSCOPE

Parameter address for module DVX_IMU: enable the IMU's gyroscope. This takes the IMU chip out of sleep.

DVX_IMU_RUN_TEMPERATURE

Parameter address for module DVX_IMU: enable the IMU's temperature sensor. This takes the IMU chip out of sleep.

DVX_IMU_ACCEL_DATA_RATE

Parameter address for module DVX_IMU: 8 settings: 0 - 12.5 Hz 1 - 25 Hz 2 - 50 Hz 3 - 100 Hz 4 - 200 Hz 5 - 400 Hz 6 - 800 Hz 7 - 1600 Hz

DVX_IMU_ACCEL_FILTER

Parameter address for module DVX_IMU: 3 settings: 0 - OSR4 1 - OSR2 2 - Normal

DVX_IMU_ACCEL_RANGE

Parameter address for module DVX_IMU: 4 settings: 0 - +- 2g 1 - +- 4g 2 - +- 8g 3 - +- 16g

DVX_IMU_GYRO_DATA_RATE

Parameter address for module DVX_IMU: 8 settings: 0 - 25 Hz 1 - 50 Hz 2 - 100 Hz 3 - 200 Hz 4 - 400 Hz 5 - 800 Hz 6 - 1600 Hz 7 - 3200 Hz

DVX_IMU_GYRO_FILTER

Parameter address for module DVX_IMU: 3 settings: 0 - OSR4 1 - OSR2 2 - Normal

DVX_IMU_GYRO_RANGE

Parameter address for module DVX_IMU: 5 settings: 0 - +- 2000°/s 1 - +- 1000°/s 2 - +- 500°/s 3 - +- 250°/s 4 - +- 125°/s

DVX_EXTINPUT_RUN_DETECTOR

Parameter address for module DVX_EXTINPUT: enable the signal detector module. It generates events when it sees certain types of signals, such as edges or pulses of a defined length, on the SIGNAL pin of the INPUT synchronization connector. This can be useful to inject events into the event stream in response to external stimuli or controls, such as turning on a LED lamp.

DVX_EXTINPUT_DETECT_RISING_EDGES

Parameter address for module DVX_EXTINPUT: send a special EXTERNAL_INPUT_RISING_EDGE event when a rising edge is detected (transition from low voltage to high).

DVX_EXTINPUT_DETECT_FALLING_EDGES

Parameter address for module DVX_EXTINPUT: send a special EXTERNAL_INPUT_FALLING_EDGE event when a falling edge is detected (transition from high voltage to low).

DVX_EXTINPUT_DETECT_PULSES

Parameter address for module DVX_EXTINPUT: send a special EXTERNAL_INPUT_PULSE event when a pulse, of a specified, configurable polarity and length, is detected. See DVX_EXTINPUT_DETECT_PULSE_POLARITY and DVX_EXTINPUT_DETECT_PULSE_LENGTH for more details.

DVX_EXTINPUT_DETECT_PULSE_POLARITY

Parameter address for module DVX_EXTINPUT: the polarity the pulse must exhibit to be detected as such. '1' means active high; a pulse will start when the signal goes from low to high and will continue to be seen as the same pulse as long as it stays high. '0' means active low; a pulse will start when the signal goes from high to low and will continue to be seen as the same pulse as long as it stays low.

DVX_EXTINPUT_DETECT_PULSE_LENGTH

Parameter address for module DVX_EXTINPUT: the minimal length that a pulse must have to trigger the sending of a special event. This is measured in cycles at LogicClock frequency (see 'struct *caer_davis_info*' for details on how to get the frequency).

DVX_EXTINPUT_HAS_GENERATOR

Parameter address for module DVX_EXTINPUT: read-only parameter, information about the presence of the signal generator feature. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_davis_info*' documentation to get this information.

DVX_EXTINPUT_RUN_GENERATOR

Parameter address for module DVX_EXTINPUT: enable the signal generator module. It generates a PWM-like signal based on configurable parameters and outputs it on the OUT JACK signal.

DVX_EXTINPUT_GENERATE_PULSE_POLARITY

Parameter address for module DVX_EXTINPUT: polarity of the PWM-like signal to be generated. '1' means active high, '0' means active low.

DVX_EXTINPUT_GENERATE_PULSE_INTERVAL

Parameter address for module DVX_EXTINPUT: the interval between the start of two consecutive pulses, expressed in cycles at LogicClock frequency (see 'struct *caer_davis_info*' for details on how to get the frequency). This must be bigger or equal to DVX_EXTINPUT_GENERATE_PULSE_LENGTH. To generate a signal with 50% duty cycle, this would have to be exactly double of DVX_EXTINPUT_GENERATE_PULSE_LENGTH.

DVX_EXTINPUT_GENERATE_PULSE_LENGTH

Parameter address for module DVX_EXTINPUT: the length a pulse stays active, expressed in cycles at LogicClock frequency (see 'struct *caer_davis_info*' for details on how to get the frequency). This must be smaller or equal to DVX_EXTINPUT_GENERATE_PULSE_INTERVAL. To generate a signal with 50% duty cycle, this would have to be exactly half of DVX_EXTINPUT_GENERATE_PULSE_INTERVAL.

DVX_EXTINPUT_GENERATE_INJECT_ON_RISING_EDGE

Parameter address for module DVX_EXTINPUT: enables event injection when a rising edge occurs in the generated signal; a special event EXTERNAL_GENERATOR_RISING_EDGE is emitted into the event stream.

DVX_EXTINPUT_GENERATE_INJECT_ON_FALLING_EDGE

Parameter address for module DVX_EXTINPUT: enables event injection when a falling edge occurs in the generated signal; a special event EXTERNAL_GENERATOR_FALLING_EDGE is emitted into the event stream.

DVX_SYSINFO_LOGIC_VERSION

Parameter address for module DVX_SYSINFO: read-only parameter, the version of the logic currently running on the device's FPGA/CPLD. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_dvx_info*' documentation to get this information.

DVX_SYSINFO_CHIP_IDENTIFIER

Parameter address for module DVX_SYSINFO: read-only parameter, an integer used to identify the different types of sensor chips used on the device. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_dvx_info*' documentation to get this information.

DVX_SYSINFO_DEVICE_IS_MASTER

Parameter address for module DVX_SYSINFO: read-only parameter, whether the device is currently a timestamp master or slave when synchronizing multiple devices together. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_dvx_info*' documentation to get this information.

DVX_SYSINFO_LOGIC_CLOCK

Parameter address for module DVX_SYSINFO: read-only parameter, the frequency in MHz at which the main FPGA/CPLD logic is running. This is reserved for internal use and should not be used by anything other than libcaer.

DVX_SYSINFO_USB_CLOCK

Parameter address for module DVX_SYSINFO: read-only parameter, the frequency in MHz at which the FPGA/CPLD logic related to USB data transmission is running. This is reserved for internal use and should not be used by anything other than libcaer.

DVX_SYSINFO_CLOCK_DEVIATION

Parameter address for module DVX_SYSINFO: read-only parameter, the deviation factor for the clocks. Due to how FX3 generates the clocks, which are then used by FPGA/CPLD, they are not integers but have a fractional part. This is reserved for internal use and should not be used by anything other than libcaer.

DVX_SYSINFO_LOGIC_PATCH

Parameter address for module DVX_SYSINFO: read-only parameter, the patch version of the logic currently running on the device's FPGA/CPLD. This is reserved for internal use and should not be used by anything other than libcaer.

DVX_USB_RUN

Parameter address for module DVX_USB: enable the USB FIFO module, which transfers the data from the FPGA/CPLD to the USB chip, to be then sent to the host. Turning this off will suppress any USB data communication!

DVX_USB_EARLY_PACKET_DELAY

Parameter address for module DVX_USB: the time delay after which a packet of data is committed to USB, even if it is not full yet (short USB packet). The value is in 125µs time-slices, corresponding to how USB schedules its operations (a value of 4 for example would mean waiting at most 0.5ms until sending a short USB packet to the host).

DVX_DVS_CHIP

Module address: device-side chip configuration. This state machine is responsible for configuring the Samsung DVS chip.

DVX_DVS_CHIP_MODE

DVX_DVS_CHIP_EVENT_FLATTEN

DVX_DVS_CHIP_EVENT_ON_ONLY

DVX_DVS_CHIP_EVENT_OFF_ONLY

DVX_DVS_CHIP_SUBSAMPLE_ENABLE

DVX_DVS_CHIP_AREA_BLOCKING_ENABLE

DVX_DVS_CHIP_DUAL_BINNING_ENABLE

DVX_DVS_CHIP_SUBSAMPLE_VERTICAL

DVX_DVS_CHIP_SUBSAMPLE_HORIZONTAL

DVX_DVS_CHIP_AREA_BLOCKING_0

DVX_DVS_CHIP_AREA_BLOCKING_1

DVX_DVS_CHIP_AREA_BLOCKING_2

DVX_DVS_CHIP_AREA_BLOCKING_3

DVX_DVS_CHIP_AREA_BLOCKING_4

DVX_DVS_CHIP_AREA_BLOCKING_5

DVX_DVS_CHIP_AREA_BLOCKING_6

DVX_DVS_CHIP_AREA_BLOCKING_7

DVX_DVS_CHIP_AREA_BLOCKING_8

DVX_DVS_CHIP_AREA_BLOCKING_9

DVX_DVS_CHIP_AREA_BLOCKING_10

DVX_DVS_CHIP_AREA_BLOCKING_11

DVX_DVS_CHIP_AREA_BLOCKING_12

DVX_DVS_CHIP_AREA_BLOCKING_13

DVX_DVS_CHIP_AREA_BLOCKING_14

DVX_DVS_CHIP_AREA_BLOCKING_15

DVX_DVS_CHIP_AREA_BLOCKING_16

DVX_DVS_CHIP_AREA_BLOCKING_17

DVX_DVS_CHIP_AREA_BLOCKING_18

DVX_DVS_CHIP_AREA_BLOCKING_19

DVX_DVS_CHIP_TIMESTAMP_RESET

DVX_DVS_CHIP_GLOBAL_RESET_ENABLE

DVX_DVS_CHIP_GLOBAL_RESET_DURING_READOUT

DVX_DVS_CHIP_GLOBAL_HOLD_ENABLE

DVX_DVS_CHIP_FIXED_READ_TIME_ENABLE

DVX_DVS_CHIP_EXTERNAL_TRIGGER_MODE

DVX_DVS_CHIP_TIMING_ED

DVX_DVS_CHIP_TIMING_GH2GRS

DVX_DVS_CHIP_TIMING_GRS

DVX_DVS_CHIP_TIMING_GH2SEL

DVX_DVS_CHIP_TIMING_SELW

DVX_DVS_CHIP_TIMING_SEL2AY_R

DVX_DVS_CHIP_TIMING_SEL2AY_F

DVX_DVS_CHIP_TIMING_SEL2R_R

DVX_DVS_CHIP_TIMING_SEL2R_F

DVX_DVS_CHIP_TIMING_NEXT_SEL

DVX_DVS_CHIP_TIMING_NEXT_GH

DVX_DVS_CHIP_TIMING_READ_FIXED

DVX_DVS_CHIP_DTAG_CONTROL

DVX_DVS_CHIP_MODE_OFF

DVX_DVS_CHIP_MODE_MONITOR

DVX_DVS_CHIP_MODE_STREAM

DVX_DVS_CHIP_DTAG_CONTROL_STOP

DVX_DVS_CHIP_DTAG_CONTROL_START

DVX_DVS_CHIP_DTAG_CONTROL_RESTART

DVX_DVS_CHIP_EXTERNAL_TRIGGER_MODE_TIMESTAMP_RESET

DVX_DVS_CHIP_EXTERNAL_TRIGGER_MODE_SINGLE_FRAME

DVX_DVS_CHIP_SUBSAMPLE_VERTICAL_NONE

DVX_DVS_CHIP_SUBSAMPLE_VERTICAL_HALF

DVX_DVS_CHIP_SUBSAMPLE_VERTICAL_FOURTH

DVX_DVS_CHIP_SUBSAMPLE_VERTICAL_EIGHTH

DVX_DVS_CHIP_SUBSAMPLE_HORIZONTAL_NONE

DVX_DVS_CHIP_SUBSAMPLE_HORIZONTAL_HALF

DVX_DVS_CHIP_SUBSAMPLE_HORIZONTAL_FOURTH

DVX_DVS_CHIP_SUBSAMPLE_HORIZONTAL_EIGHTH

DVX_DVS_CHIP_CROPPER

DVX_DVS_CHIP_CROPPER_ENABLE

DVX_DVS_CHIP_CROPPER_Y_START_ADDRESS

DVX_DVS_CHIP_CROPPER_Y_END_ADDRESS

DVX_DVS_CHIP_CROPPER_X_START_ADDRESS

DVX_DVS_CHIP_CROPPER_X_END_ADDRESS

DVX_DVS_CHIP_ACTIVITY_DECISION

DVX_DVS_CHIP_ACTIVITY_DECISION_ENABLE

DVX_DVS_CHIP_ACTIVITY_DECISION_POS_THRESHOLD

DVX_DVS_CHIP_ACTIVITY_DECISION_NEG_THRESHOLD

DVX_DVS_CHIP_ACTIVITY_DECISION_DEC_RATE

DVX_DVS_CHIP_ACTIVITY_DECISION_DEC_TIME

DVX_DVS_CHIP_ACTIVITY_DECISION_POS_MAX_COUNT

DVX_DVS_CHIP_BIAS

DVX_DVS_CHIP_BIAS_CURRENT_RANGE_LOG

DVX_DVS_CHIP_BIAS_CURRENT_RANGE_SF

DVX_DVS_CHIP_BIAS_CURRENT_RANGE_ON

DVX_DVS_CHIP_BIAS_CURRENT_RANGE_nRST

DVX_DVS_CHIP_BIAS_CURRENT_RANGE_LOGA

DVX_DVS_CHIP_BIAS_CURRENT_RANGE_LOGD

DVX_DVS_CHIP_BIAS_CURRENT_LEVEL_SF

DVX_DVS_CHIP_BIAS_CURRENT_LEVEL_nOFF

DVX_DVS_CHIP_BIAS_CURRENT_AMP

DVX_DVS_CHIP_BIAS_CURRENT_ON

DVX_DVS_CHIP_BIAS_CURRENT_OFF

DVX_DVS_CHIP_BIAS_SIMPLE

DVX_DVS_CHIP_BIAS_CURRENT_RANGE_LOG_5uA

DVX_DVS_CHIP_BIAS_CURRENT_RANGE_LOG_50uA

DVX_DVS_CHIP_BIAS_CURRENT_RANGE_SF_0_5uA

DVX_DVS_CHIP_BIAS_CURRENT_RANGE_SF_5uA

DVX_DVS_CHIP_BIAS_CURRENT_RANGE_ON_5uA

DVX_DVS_CHIP_BIAS_CURRENT_RANGE_ON_50uA

DVX_DVS_CHIP_BIAS_CURRENT_RANGE_nRST_0_5uA

DVX_DVS_CHIP_BIAS_CURRENT_RANGE_nRST_5uA

DVX_DVS_CHIP_BIAS_CURRENT_RANGE_LOGA_5uA

DVX_DVS_CHIP_BIAS_CURRENT_RANGE_LOGA_50uA

DVX_DVS_CHIP_BIAS_CURRENT_RANGE_LOGD_5uA

DVX_DVS_CHIP_BIAS_CURRENT_RANGE_LOGD_50uA

DVX_DVS_CHIP_BIAS_CURRENT_RANGE_LOGD_500uA

DVX_DVS_CHIP_BIAS_CURRENT_LEVEL_SF_x0_1

DVX_DVS_CHIP_BIAS_CURRENT_LEVEL_SF_x1

DVX_DVS_CHIP_BIAS_CURRENT_LEVEL_nOFF_x0_1

DVX_DVS_CHIP_BIAS_CURRENT_LEVEL_nOFF_x1

DVX_DVS_CHIP_BIAS_SIMPLE_VERY_LOW

DVX_DVS_CHIP_BIAS_SIMPLE_LOW

DVX_DVS_CHIP_BIAS_SIMPLE_DEFAULT

DVX_DVS_CHIP_BIAS_SIMPLE_HIGH

DVX_DVS_CHIP_BIAS_SIMPLE_VERY_HIGH

Functions

LIBRARY_PUBLIC_VISIBILITY struct caer_dvx_info caerDVXplorerInfoGet (caerDeviceHandle handle)

Return basic information on the device, such as its ID, its resolution, the logic version, and so on. See the 'struct *caer_dvx_info*' documentation for more details.

Parameters
handle – a valid device handle.

Returns
a copy of the device information structure if successful, an empty structure (all zeros) on failure.

file **dynapse.h**

#include "../events/special.h" #include "../events/spike.h" #include "usb.h" Dynap-se specific configuration defines and information structures.

Defines

CAER_DEVICE_DYNAPSE

Device type definition for aiCTX Dynap-se FX2-based boards.

DYNAPSE_CHIP_DYNAPSE

Dynap-se chip identifier.

DYNAPSE_CONFIG_MUX

Module address: device-side Multiplexer configuration. The Multiplexer is responsible for mixing, timestamping and outputting (via USB) the various event types generated by the device. It is also responsible for timestamp generation.

DYNAPSE_CONFIG_AER

Module address: device-side AER configuration (from chip). The AER state machine handshakes with the chip's AER bus and gets the spike events from it. It supports various configurable delays.

DYNAPSE_CONFIG_CHIP

Module address: device-side chip control configuration. This state machine is responsible for configuring the chip's internal control registers, to set special options and biases.

DYNAPSE_CONFIG_SYSINFO

Module address: device-side system information. The system information module provides various details on the device, such as currently installed logic revision or clock speeds. All its parameters are read-only. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_dynapse_info*' documentation for more details on what information is available.

DYNAPSE_CONFIG_USB

Module address: device-side USB output configuration. The USB output module forwards the data from the device and the FPGA/CPLD to the USB chip, usually a Cypress FX2 or FX3.

DYNAPSE_CONFIG_CLEAR_CAM

Clear CAM content, on all cores of a chip. No arguments are used. Remember to select the chip you want to configure before this!

DYNAPSE_CONFIG_DEFAULT_SRAM

Clear SRAM content, use one SRAM cell (cell 0, out of the four available) to monitor neurons via USB. 'paramAddr' is the chip ID on which to operate, other arguments are unused. Remember to also select the chip you want to configure before this!

DYNAPSE_CONFIG_MONITOR_NEU

Setup analog neuron monitoring via SMA connectors. 'paramAddr' takes the core ID to be monitored, 'param' the neuron ID. Remember to select the chip you want to configure before this!

DYNAPSE_CONFIG_DEFAULT_SRAM_EMPTY

Clear SRAM content, route nothing outside (all four SRAM cells zero). No arguments are used. Remember to select the chip you want to configure before this!

DYNAPSE_CONFIG_SRAM

Module address: Device side SRAM controller configuration. The module holds an address, a word to be written to SRAM, the most recent word read using a read command, and a read/write command. Reads/writes are triggered when the address field is changed. Example: caerDynapseWriteSramWords(devHandle, SRAMData, baseAddr, numWords); Writes numWords words from array SRAMData to the SRAM, starting at baseAddr. This define is for internal use of *caerDynapseWriteSramWords()*; it can be used on its own, but we recommend using the above function that hides all the internal details of writing to the FPGA SRAM.

DYNAPSE_CONFIG_SYNAPSERECONFIG

Module address: Device side Synapse Reconfiguration module configuration. Provides run control, selection between using a single kernel for all neurons and reading per-neuron kernels from SRAM, programming of the global kernel, as well as target output chip ID selection and SRAM kernel table base address.

DYNAPSE_CONFIG_SPIKEGEN

Module address: Device side spike generator module configuration. Provides start/stop control of spike train application and selection of fixed/variable inter-spike intervals and their location in memory.

DYNAPSE_CONFIG_TAU2_SET

Set certain neurons of a core to use the TAU2 neuron leak bias. By default neurons use the TAU1 neuron leak bias. You can also use DYNAPSE_CONFIG_TAU1_RESET and DYNAPSE_CONFIG_TAU2_RESET to reset all neurons in a core to the same bias. 'paramAddr' takes the core ID to be set, 'param' the neuron ID. Remember to select the chip you want to configure before this!

DYNAPSE_CONFIG_POISSONSPIKEGEN

Module address: Device side poisson generator configuration Provides run/stop control of poisson spike generation and rate setting for 1024 sources.

DYNAPSE_CONFIG_TAU1_RESET

Reset all neurons of a core to use the TAU1 neuron leak bias. 'paramAddr' takes the core ID to be reset, other arguments are unused. Remember to select the chip you want to configure before this!

DYNAPSE_CONFIG_TAU2_RESET

Reset all neurons of a core to use the TAU2 neuron leak bias. 'paramAddr' takes the core ID to be reset, other arguments are unused. Remember to select the chip you want to configure before this!

DYNAPSE_CONFIG_POISSONSPIKEGEN_RUN

Parameter address for module DYNAPSE_CONFIG_POISSONSPIKEGEN: Enables or disables generation of poisson spike trains.

DYNAPSE_CONFIG_POISSONSPIKEGEN_WRITEADDRESS

Parameter address for module DYNAPSE_CONFIG_POISSONSPIKEGEN: Selects the address of a poisson spike train source. Writing to this parameter will apply the rate previously written to the WRITEDATA field.

DYNAPSE_CONFIG_POISSONSPIKEGEN_WRITEDATA

Parameter address for module DYNAPSE_CONFIG_POISSONSPIKEGEN: Holds data that will be written to the address specified by WRITEADDRESS.

DYNAPSE_CONFIG_POISSONSPIKEGEN_CHIPID

Parameter address for module DYNAPSE_CONFIG_POISSONSPIKEGEN: Chip ID of the chip that will receive events generated by the poisson spike generator.

DYNAPSE_CONFIG_SPIKEGEN_RUN

Parameter address for module DYNAPSE_CONFIG_SPIKEGEN: Instructs the spike generator to start applying the configured spike train when the parameter changes from false to true.

DYNAPSE_CONFIG_SPIKEGEN_VARMODE

Parameter address for module DYNAPSE_CONFIG_SPIKEGEN: Selects variable inter-spike interval mode (true) or fixed inter-spike interval mode (false).

DYNAPSE_CONFIG_SPIKEGEN_BASEADDR

Parameter address for module DYNAPSE_CONFIG_SPIKEGEN: Sets the start address of a spike train in memory.

DYNAPSE_CONFIG_SPIKEGEN_STIMCOUNT

Paramter address for module DYNAPSE_CONFIG_SPIKEGEN: Sets the number of events to read from memory for a single application of a spike train.

DYNAPSE_CONFIG_SPIKEGEN_ISI

Parameter address for module DYNAPSE_CONFIG_SPIKEGEN: Sets the inter-spike interval that will be used in fixed ISI mode (VARMODE false).

DYNAPSE_CONFIG_SPIKEGEN_ISIBASE

Parameter address for module DYNAPSE_CONFIG_SPIKEGEN: Sets the time base resolution for inter-spike intervals as the number of FPGA clock cycles.

DYNAPSE_CONFIG_SPIKEGEN_REPEAT

Parameter address for module DYNAPSE_CONFIG_SPIKEGEN: Sets repeat mode to true or false.

DYNAPSE_CONFIG_SYNAPSERECONFIG_RUN

Parameter address for module DYNAPSE_CONFIG_SYNAPSERECONFIG: Run control. Starts and stops handshaking with DVS.

DYNAPSE_CONFIG_SYNAPSERECONFIG_GLOBALKERNEL

Parameter address for module DYNAPSE_CONFIG_SYNAPSERECONFIG: Bits 16 down to 12 select the address in the global kernel table and bits 11 down to 0 specify the data. The 12 data bits are split into 4*3 synaptic weight bits which map onto positive/negative polarity events from 2 DVS pixels.

DYNAPSE_CONFIG_SYNAPSERECONFIG_USESRAMKERNELS

Parameter address for module DYNAPSE_CONFIG_SYNAPSERECONFIG: Boolean parameter for selecting between using kernels stored in SRAM or the global kernel table. 1 for SRAM, 0 for global kernel table.

DYNAPSE_CONFIG_SYNAPSERECONFIG_CHIPSELECT

Parameter address for module DYNAPSE_CONFIG_SYNAPSERECONFIG: Select which chip outputs should go to.

DYNAPSE_CONFIG_SYNAPSERECONFIG_SRAMBASEADDR

Parameter address for module DYNAPSE_CONFIG_SYNAPSERECONFIG: SRAM base address configuration in increments of 32 Kib. Setting this to N will place the SRAM kernel LUT in the range [N*2^15,((N+1)*2^15)-1]

DYNAPSE_CONFIG_SRAM_ADDRESS

Parameter address for module DYNAPSE_CONFIG_SRAM: Holds the address that will be used for the next read/write. Writing or reading this field will trigger the command contained in the command register to be executed on the FPGA.

DYNAPSE_CONFIG_SRAM_READDATA

Parameter address for module DYNAPSE_CONFIG_SRAM: Holds the most recently read data from the SRAM. Read-only parameter.

DYNAPSE_CONFIG_SRAM_WRITEDATA

Parameter address for module DYNAPSE_CONFIG_SRAM: Holds the data that will be written on the next write. Example: caerDeviceConfigSet(devHandle, DYNAPSE_CONFIG_SRAM, DYNAPSE_CONFIG_SRAM_WRITEDATA, wData); caerDeviceConfigSet(devHandle, DYNAPSE_CONFIG_SRAM, DYNAPSE_CONFIG_SRAM_RWCOMMAND, DYNAPSE_CONFIG_SRAM_WRITE); caerDeviceConfigSet(devHandle, DYNAPSE_CONFIG_SRAM, DYNAPSE_CONFIG_SRAM_ADDRESS, wAddr); Writes wData to the address specified by wAddr.

DYNAPSE_CONFIG_SRAM_RWCOMMAND

Parameter address for module DYNAPSE_CONFIG_SRAM: Holds the command that will be executed when the address field is written to. Example: caerDeviceConfigSet(devHandle, DYNAPSE_CONFIG_SRAM, DYNAPSE_CONFIG_SRAM_RWCOMMAND, DYNAPSE_CONFIG_SRAM_WRITE); Sets the SRAM controller up for doing writes. DYNAPSE_CONFIG_SRAM_READ and DYNAPSE_CONFIG_SRAM_WRITE are supported.

DYNAPSE_CONFIG_SRAM_READ

Command for module DYNAPSE_CONFIG_SRAM: Read command for the RWCOMMAND field. Example: caerDeviceConfigSet(devHandle, DYNAPSE_CONFIG_SRAM, DYNAPSE_CONFIG_SRAM_RWCOMMAND, DYNAPSE_CONFIG_SRAM_READ); Sets the SRAM controller up for doing reads.

DYNAPSE_CONFIG_SRAM_WRITE

Command for module DYNAPSE_CONFIG_SRAM: Write command for the RWCOMMAND field. Example: caerDeviceConfigSet(devHandle, DYNAPSE_CONFIG_SRAM, DYNAPSE_CONFIG_SRAM_RWCOMMAND, DYNAPSE_CONFIG_SRAM_WRITE); Sets the SRAM controller up for doing writes.

DYNAPSE_CONFIG_SRAM_BURSTMODE

Parameter address for module DYNAPSE_CONFIG_SRAM: Burst mode enable for fast writing. Disables updates on address change and instead updates on data change, while automatically incrementing the writing address. Two 16-bit words are written per 32-bit word sent to the SPI controller starting with the least significant half word.

DYNAPSE_CONFIG_MUX_RUN

Parameter address for module DYNAPSE_CONFIG_MUX: run the Multiplexer state machine, which is responsible for mixing the various event types at the device level, timestamping them and outputting them via USB or other connectors.

DYNAPSE_CONFIG_MUX_TIMESTAMP_RUN

Parameter address for module DYNAPSE_CONFIG_MUX: run the Timestamp Generator inside the Multiplexer state machine, which will provide microsecond accurate timestamps to the events passing through.

DYNAPSE_CONFIG_MUX_TIMESTAMP_RESET

Parameter address for module DYNAPSE_CONFIG_MUX: reset the Timestamp Generator to zero. This also sends a reset pulse to all connected slave devices, resetting their timestamp too.

DYNAPSE_CONFIG_MUX_FORCE_CHIP_BIAS_ENABLE

Parameter address for module DYNAPSE_CONFIG_MUX: under normal circumstances, the chip's bias generator is only powered up when either the AER or the configuration state machines are running, to save power. This flag forces the bias generator to be powered up all the time.

DYNAPSE_CONFIG_MUX_DROP_AER_ON_TRANSFER_STALL

Parameter address for module DYNAPSE_CONFIG_MUX: drop AER events if the USB output FIFO is full, instead of having them pile up at the input FIFOs.

DYNAPSE_CONFIG_MUX_HAS_STATISTICS

Parameter address for module DYNAPSE_CONFIG_MUX: read-only parameter, information about the presence of the statistics feature. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_dynapse_info*' documentation to get this information.

DYNAPSE_CONFIG_MUX_STATISTICS_AER_DROPPED

Parameter address for module DYNAPSE_CONFIG_MUX: read-only parameter, representing the number of dropped AER (spike) events on the device due to full USB buffers. This is a 64bit value, and should always be read using the function: *caerDeviceConfigGet64()*.

DYNAPSE_CONFIG_AER_RUN

Parameter address for module DYNAPSE_CONFIG_AER: run the AER state machine and get spike events from the chip by handshaking with its AER bus.

DYNAPSE_CONFIG_AER_ACK_DELAY

Parameter address for module DYNAPSE_CONFIG_AER: delay capturing the data and acknowledging it on the AER bus for the events by this many LogicClock cycles.

DYNAPSE_CONFIG_AER_ACK_EXTENSION

Parameter address for module DYNAPSE_CONFIG_AER: extend the length of the acknowledge on the AER bus for the events by this many LogicClock cycles.

DYNAPSE_CONFIG_AER_WAIT_ON_TRANSFER_STALL

Parameter address for module DYNAPSE_CONFIG_AER: if the output FIFO for this module is full, stall the AER handshake with the chip and wait until it's free again, instead of just continuing the handshake and dropping the resulting events.

DYNAPSE_CONFIG_AER_EXTERNAL_AER_CONTROL

Parameter address for module DYNAPSE_CONFIG_AER: enable external AER control. This ensures the chip and the neuron array are running, but doesn't do the handshake and leaves the ACK pin in high-impedance, to allow for an external system to take over the AER communication with the chip. DYNAPSE_CONFIG_AER_RUN has to be turned off for this to work.

DYNAPSE_CONFIG_AER_HAS_STATISTICS

Parameter address for module DYNAPSE_CONFIG_AER: read-only parameter, information about the presence of the statistics feature. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_dynapse_info*' documentation to get this information.

DYNAPSE_CONFIG_AER_STATISTICS_EVENTS

Parameter address for module DYNAPSE_CONFIG_AER: read-only parameter, representing the number of event transactions completed on the device. This is a 64bit value, and should always be read using the function: *caerDeviceConfigGet64()*.

DYNAPSE_CONFIG_AER_STATISTICS_EVENTS_DROPPED

Parameter address for module DYNAPSE_CONFIG_AER: read-only parameter, representing the number of dropped transaction sequences on the device due to full buffers. This is a 64bit value, and should always be read using the function: *caerDeviceConfigGet64()*.

DYNAPSE_CONFIG_CHIP_RUN

Parameter address for module DYNAPSE_CONFIG_CHIP: enable the configuration AER state machine to send bias and control configuration to the chip.

DYNAPSE_CONFIG_CHIP_ID

Parameter address for module DYNAPSE_CONFIG_CHIP: set the chip ID to which configuration content is being sent.

DYNAPSE_CONFIG_CHIP_CONTENT

Parameter address for module DYNAPSE_CONFIG_CHIP: set the configuration content to send to the chip. Every time this changes, the chip ID is appended and the configuration is sent out to the chip.

DYNAPSE_CONFIG_CHIP_REQ_DELAY

Parameter address for module DYNAPSE_CONFIG_CHIP: delay doing the request after putting out the data by this many LogicClock cycles.

DYNAPSE_CONFIG_CHIP_REQ_EXTENSION

Parameter address for module DYNAPSE_CONFIG_CHIP: extend the request after receiving the ACK by this many LogicClock cycles.

DYNAPSE_CONFIG_SYSINFO_LOGIC_VERSION

Parameter address for module DYNAPSE_CONFIG_SYSINFO: read-only parameter, the version of the logic currently running on the device's FPGA/CPLD. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_dynapse_info*' documentation to get this information.

DYNAPSE_CONFIG_SYSINFO_CHIP_IDENTIFIER

Parameter address for module DYNAPSE_CONFIG_SYSINFO: read-only parameter, an integer used to identify the different types of sensor chips used on the device. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_dynapse_info*' documentation to get this information.

DYNAPSE_CONFIG_SYSINFO_DEVICE_IS_MASTER

Parameter address for module DYNAPSE_CONFIG_SYSINFO: read-only parameter, whether the device is currently a timestamp master or slave when synchronizing multiple devices together. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_dynapse_info*' documentation to get this information.

DYNAPSE_CONFIG_SYSINFO_LOGIC_CLOCK

Parameter address for module DYNAPSE_CONFIG_SYSINFO: read-only parameter, the frequency in MHz at which the main FPGA/CPLD logic is running. This is reserved for internal use and should not be used by anything other than libcaer. Please see the 'struct *caer_dynapse_info*' documentation to get this information.

DYNAPSE_CONFIG_USB_RUN

Parameter address for module DYNAPSE_CONFIG_USB: enable the USB FIFO module, which transfers the data from the FPGA/CPLD to the USB chip, to be then sent to the host. Turning this off will suppress any USB data communication!

DYNAPSE_CONFIG_USB_EARLY_PACKET_DELAY

Parameter address for module DYNAPSE_CONFIG_USB: the time delay after which a packet of data is committed to USB, even if it is not full yet (short USB packet). The value is in 125µs time-slices, corresponding to how USB schedules its operations (a value of 4 for example would mean waiting at most 0.5ms until sending a short USB packet to the host).

DYNAPSE_CONFIG_SRAM_DIRECTION_POS

On-chip SRAM for spike routing.

DYNAPSE_CONFIG_SRAM_DIRECTION_NEG

DYNAPSE_CONFIG_SRAM_DIRECTION_Y_NORTH

DYNAPSE_CONFIG_SRAM_DIRECTION_Y_SOUTH

DYNAPSE_CONFIG_SRAM_DIRECTION_X_EAST

DYNAPSE_CONFIG_SRAM_DIRECTION_X_WEST

DYNAPSE_X4BOARD_NUMCHIPS
: Number of chips on the board.

DYNAPSE_X4BOARD_NEUX
: Number of neurons in the x direction of the board.

DYNAPSE_X4BOARD_NEUY
: Number of neurons in the y direction of the board.

DYNAPSE_X4BOARD_COREX
: Number of cores in the x direction of the board.

DYNAPSE_X4BOARD_COREY
: Number of cores in the y direction of the board.

DYNAPSE_CONFIG_DYNAPSE_U0
: Chip 0 ID.

DYNAPSE_CONFIG_DYNAPSE_U1
: Chip 1 ID.

DYNAPSE_CONFIG_DYNAPSE_U2
: Chip 2 ID.

DYNAPSE_CONFIG_DYNAPSE_U3
: Chip 3 ID.

DYNAPSE_CONFIG_NUMCORES
: Number of cores per chip.

DYNAPSE_CONFIG_NUMNEURONS
: Number of neurons in single chip.

DYNAPSE_CONFIG_NUMNEURONS_CORE
: Number of neurons per core.

DYNAPSE_CONFIG_XCHIPSIZE

Number of columns of neurons in a chip.

DYNAPSE_CONFIG_YCHIPSIZE

Number of rows of neurons in a core.

DYNAPSE_CONFIG_NEUCOL

Number of columns of neurons in a core.

DYNAPSE_CONFIG_NEUROW

Number of rows of neurons in a core.

DYNAPSE_CONFIG_CAMCOL

Number of columns of CAMs in a core.

DYNAPSE_CONFIG_NUMCAM_NEU

Number of CAMs per neuron.

DYNAPSE_CONFIG_NUMSRAM_NEU

Number of SRAM cells per neuron.

DYNAPSE_CONFIG_CAMTYPE_F_EXC

Fast excitatory synapse.

DYNAPSE_CONFIG_CAMTYPE_S_EXC

Slow excitatory synapse.

DYNAPSE_CONFIG_CAMTYPE_F_INH

Fast inhibitory synapse.

DYNAPSE_CONFIG_CAMTYPE_S_INH

Slow inhibitory synapse.

DYNAPSE_CONFIG_BIAS_C0_PULSE_PWLK_P

Parameter address for module DYNAPSE_CONFIG_BIAS: DYNAPSE chip biases. Bias configuration values must be generated using the proper functions, which are:

- *caerBiasDynapseGenerate()* for Dynap-se coarse-fine (current) biases. See 'https://ai-ctx.com/support/'[25] section 'Neuron's behaviors and parameters tuning'.

DYNAPSE_CONFIG_BIAS_C0_PS_WEIGHT_INH_S_N

DYNAPSE_CONFIG_BIAS_C0_PS_WEIGHT_INH_F_N

DYNAPSE_CONFIG_BIAS_C0_PS_WEIGHT_EXC_S_N

DYNAPSE_CONFIG_BIAS_C0_PS_WEIGHT_EXC_F_N

DYNAPSE_CONFIG_BIAS_C0_IF_RFR_N

DYNAPSE_CONFIG_BIAS_C0_IF_TAU1_N

DYNAPSE_CONFIG_BIAS_C0_IF_AHTAU_N

DYNAPSE_CONFIG_BIAS_C0_IF_CASC_N

DYNAPSE_CONFIG_BIAS_C0_IF_TAU2_N

DYNAPSE_CONFIG_BIAS_C0_IF_BUF_P

DYNAPSE_CONFIG_BIAS_C0_IF_AHTHR_N

DYNAPSE_CONFIG_BIAS_C0_IF_THR_N

DYNAPSE_CONFIG_BIAS_C0_NPDPIE_THR_S_P

DYNAPSE_CONFIG_BIAS_C0_NPDPIE_THR_F_P

DYNAPSE_CONFIG_BIAS_C0_NPDPII_THR_F_P

DYNAPSE_CONFIG_BIAS_C0_NPDPII_THR_S_P

DYNAPSE_CONFIG_BIAS_C0_IF_NMDA_N

DYNAPSE_CONFIG_BIAS_C0_IF_DC_P

DYNAPSE_CONFIG_BIAS_C0_IF_AHW_P

DYNAPSE_CONFIG_BIAS_C0_NPDPII_TAU_S_P

DYNAPSE_CONFIG_BIAS_C0_NPDPII_TAU_F_P

DYNAPSE_CONFIG_BIAS_C0_NPDPIE_TAU_F_P

DYNAPSE_CONFIG_BIAS_C0_NPDPIE_TAU_S_P

DYNAPSE_CONFIG_BIAS_C0_R2R_P

DYNAPSE_CONFIG_BIAS_C1_PULSE_PWLK_P

DYNAPSE_CONFIG_BIAS_C1_PS_WEIGHT_INH_S_N

DYNAPSE_CONFIG_BIAS_C1_PS_WEIGHT_INH_F_N

DYNAPSE_CONFIG_BIAS_C1_PS_WEIGHT_EXC_S_N

DYNAPSE_CONFIG_BIAS_C1_PS_WEIGHT_EXC_F_N

DYNAPSE_CONFIG_BIAS_C1_IF_RFR_N

DYNAPSE_CONFIG_BIAS_C1_IF_TAU1_N

DYNAPSE_CONFIG_BIAS_C1_IF_AHTAU_N

DYNAPSE_CONFIG_BIAS_C1_IF_CASC_N

DYNAPSE_CONFIG_BIAS_C1_IF_TAU2_N

DYNAPSE_CONFIG_BIAS_C1_IF_BUF_P

DYNAPSE_CONFIG_BIAS_C1_IF_AHTHR_N

DYNAPSE_CONFIG_BIAS_C1_IF_THR_N

DYNAPSE_CONFIG_BIAS_C1_NPDPIE_THR_S_P

DYNAPSE_CONFIG_BIAS_C1_NPDPIE_THR_F_P

DYNAPSE_CONFIG_BIAS_C1_NPDPII_THR_F_P

DYNAPSE_CONFIG_BIAS_C1_NPDPII_THR_S_P

DYNAPSE_CONFIG_BIAS_C1_IF_NMDA_N

DYNAPSE_CONFIG_BIAS_C1_IF_DC_P

DYNAPSE_CONFIG_BIAS_C1_IF_AHW_P

DYNAPSE_CONFIG_BIAS_C1_NPDPII_TAU_S_P

DYNAPSE_CONFIG_BIAS_C1_NPDPII_TAU_F_P

DYNAPSE_CONFIG_BIAS_C1_NPDPIE_TAU_F_P

DYNAPSE_CONFIG_BIAS_C1_NPDPIE_TAU_S_P

DYNAPSE_CONFIG_BIAS_C1_R2R_P

DYNAPSE_CONFIG_BIAS_U_BUFFER

DYNAPSE_CONFIG_BIAS_U_SSP

DYNAPSE_CONFIG_BIAS_U_SSN

DYNAPSE_CONFIG_BIAS_C2_PULSE_PWLK_P

DYNAPSE_CONFIG_BIAS_C2_PS_WEIGHT_INH_S_N

DYNAPSE_CONFIG_BIAS_C2_PS_WEIGHT_INH_F_N

DYNAPSE_CONFIG_BIAS_C2_PS_WEIGHT_EXC_S_N

DYNAPSE_CONFIG_BIAS_C2_PS_WEIGHT_EXC_F_N

DYNAPSE_CONFIG_BIAS_C2_IF_RFR_N

DYNAPSE_CONFIG_BIAS_C2_IF_TAU1_N

DYNAPSE_CONFIG_BIAS_C2_IF_AHTAU_N

DYNAPSE_CONFIG_BIAS_C2_IF_CASC_N

DYNAPSE_CONFIG_BIAS_C2_IF_TAU2_N

DYNAPSE_CONFIG_BIAS_C2_IF_BUF_P

DYNAPSE_CONFIG_BIAS_C2_IF_AHTHR_N

DYNAPSE_CONFIG_BIAS_C2_IF_THR_N

DYNAPSE_CONFIG_BIAS_C2_NPDPIE_THR_S_P

DYNAPSE_CONFIG_BIAS_C2_NPDPIE_THR_F_P

DYNAPSE_CONFIG_BIAS_C2_NPDPII_THR_F_P

DYNAPSE_CONFIG_BIAS_C2_NPDPII_THR_S_P

DYNAPSE_CONFIG_BIAS_C2_IF_NMDA_N

DYNAPSE_CONFIG_BIAS_C2_IF_DC_P

DYNAPSE_CONFIG_BIAS_C2_IF_AHW_P

DYNAPSE_CONFIG_BIAS_C2_NPDPII_TAU_S_P

DYNAPSE_CONFIG_BIAS_C2_NPDPII_TAU_F_P

DYNAPSE_CONFIG_BIAS_C2_NPDPIE_TAU_F_P

DYNAPSE_CONFIG_BIAS_C2_NPDPIE_TAU_S_P

DYNAPSE_CONFIG_BIAS_C2_R2R_P

DYNAPSE_CONFIG_BIAS_C3_PULSE_PWLK_P

DYNAPSE_CONFIG_BIAS_C3_PS_WEIGHT_INH_S_N

DYNAPSE_CONFIG_BIAS_C3_PS_WEIGHT_INH_F_N

DYNAPSE_CONFIG_BIAS_C3_PS_WEIGHT_EXC_S_N

DYNAPSE_CONFIG_BIAS_C3_PS_WEIGHT_EXC_F_N

DYNAPSE_CONFIG_BIAS_C3_IF_RFR_N

DYNAPSE_CONFIG_BIAS_C3_IF_TAU1_N

DYNAPSE_CONFIG_BIAS_C3_IF_AHTAU_N

DYNAPSE_CONFIG_BIAS_C3_IF_CASC_N

DYNAPSE_CONFIG_BIAS_C3_IF_TAU2_N

DYNAPSE_CONFIG_BIAS_C3_IF_BUF_P

DYNAPSE_CONFIG_BIAS_C3_IF_AHTHR_N

DYNAPSE_CONFIG_BIAS_C3_IF_THR_N

DYNAPSE_CONFIG_BIAS_C3_NPDPIE_THR_S_P

DYNAPSE_CONFIG_BIAS_C3_NPDPIE_THR_F_P

DYNAPSE_CONFIG_BIAS_C3_NPDPII_THR_F_P

DYNAPSE_CONFIG_BIAS_C3_NPDPII_THR_S_P

DYNAPSE_CONFIG_BIAS_C3_IF_NMDA_N

DYNAPSE_CONFIG_BIAS_C3_IF_DC_P

DYNAPSE_CONFIG_BIAS_C3_IF_AHW_P

DYNAPSE_CONFIG_BIAS_C3_NPDPII_TAU_S_P

DYNAPSE_CONFIG_BIAS_C3_NPDPII_TAU_F_P

DYNAPSE_CONFIG_BIAS_C3_NPDPIE_TAU_F_P

DYNAPSE_CONFIG_BIAS_C3_NPDPIE_TAU_S_P

DYNAPSE_CONFIG_BIAS_C3_R2R_P

DYNAPSE_CONFIG_BIAS_D_BUFFER

DYNAPSE_CONFIG_BIAS_D_SSP

DYNAPSE_CONFIG_BIAS_D_SSN

Functions

LIBRARY_PUBLIC_VISIBILITY struct caer_dynapse_info caerDynapseInfoGet (caerDeviceHandle handle)

Return basic information on the device, such as its ID, the logic version, and so on. See the 'struct *caer_dynapse_info*' documentation for more details.

Parameters
handle – a valid device handle.

Returns
a copy of the device information structure if successful, an empty structure (all zeros) on failure.

LIBRARY_PUBLIC_VISIBILITY uint32_t caerBiasDynapseGenerate (const struct caer_bias_dynapse dynapseB

Transform coarse-fine bias structure into internal integer representation, suited for sending directly to the device via *caerDeviceConfigSet()*.

Parameters
dynapseBias – coarse-fine bias structure.

Returns
internal integer representation for device configuration.

LIBRARY_PUBLIC_VISIBILITY struct caer_bias_dynapse caerBiasDynapseParse (const uint32_t dynapseBias

Transform internal integer representation, as received by calls to *caerDeviceConfigGet()*, into a coarse-fine bias structure, for easier handling and understanding of the various parameters.

Parameters
dynapseBias – internal integer representation from device.

Returns
coarse-fine bias structure.

LIBRARY_PUBLIC_VISIBILITY bool caerDynapseWriteSramWords (caerDeviceHandle handle, const uint16_t *data, uint32_t baseAddr, size_t numWords)

Transfer 16bit words from memory to device SRAM, with configurable starting address and number of words. This works on the FPGA SRAM!

Parameters

- **handle** – a valid device handle.
- **data** – array from which to read data to send to SRAM.
- **baseAddr** – SRAM start address where to put the data.
- **numWords** – number of 16bit words to transfer.

Returns
true on success, false otherwise.

LIBRARY_PUBLIC_VISIBILITY bool caerDynapseWritePoissonSpikeRate (caerDeviceHandle handle, uint16_t neuronAddr, float rateHz)

Specifies the poisson spike generator's spike rate.

Parameters

- **handle** – a valid device handle.

- **neuronAddr** – The target neuron of the poisson spike train, range [0,1023].
- **rateHz** – The rate in Hz of the spike train, this will be quantized to the nearest supported level, range [0,4300].

Returns
true on success, false otherwise.

LIBRARY_PUBLIC_VISIBILITY bool caerDynapseWriteSram (caerDeviceHandle handle, uint8_t coreId, uint8_t neuronAddrCore, uint8_t virtualCoreId, bool sx, uint8_t dx, bool sy, uint8_t dy, uint8_t sramId, uint8_t destinationCore)

THIS FUNCTION IS DEPRECATED. USE *caerDynapseWriteSramN()* INSTEAD! The new function uses the global neuron ID (range [0,1023]) like all others, instead of the separate core ID/neuron ID syntax. Also the arguments are in the same order as *caerDynapseGenerateSramBits()*, in particular the 'sramId' comes right after 'neuronId'.

Write one of the 4 SRAMs of a single neuron. Writing the SRAM means writing the destination address of where the spikes will be routed to. This works on the on-chip SRAM!

Remember to select the chip you want to configure before calling this function!

Parameters

- **handle** – a valid device handle.
- **coreId** – the chip's core ID, range [0,3].
- **neuronAddrCore** – the neuron's address within this core, range [0,255].
- **virtualCoreId** – fake source core ID, set it to this value instead of the actual source core ID, range [0,3].
- **sx** – X direction, can be one of: [DYNAPSE_CONFIG_SRAM_DIRECTION_X_EAST,DYNAPSE_CONFIG_S
- **dx** – X delta, number of chips to jumps before reaching destination, range is [0,3].
- **sy** – Y direction, can be one of: [DYNAPSE_CONFIG_SRAM_DIRECTION_Y_NORTH,DYNAPSE_CONFIG_
- **dy** – Y delta, number of chips to jumps before reaching destination, range is [0,3].
- **sramId** – SRAM address (one of four cells), range [0,3].
- **destinationCore** – spike destination core, uses one-hot coding for the 4 cores: [C3,C2,C1,C0] -> [0,0,0,0] (0 decimal) no core, [1,1,1,1] (15 decimal) all cores

Returns
true on success, false otherwise.

LIBRARY_PUBLIC_VISIBILITY bool caerDynapseWriteSramN (caerDeviceHandle handle, uint16_t neuronAddr, uint8_t sramId, uint8_t virtualCoreId, bool sx, uint8_t dx, bool sy, uint8_t dy, uint8_t destinationCore)

Write one of the 4 SRAMs of a single neuron. Writing the SRAM means writing the destination address of where the spikes will be routed to. This works on the on-chip SRAM!

Remember to select the chip you want to configure before calling this function!

Parameters

- **handle** – a valid device handle.
- **neuronAddr** – the neuron to program, range [0,1023] (use *caerDynapseCoreXYToNeuronId()* for a 2D mapping).
- **sramId** – SRAM address (one of four cells), range [0,3].

- **virtualCoreId** – fake source core ID, set it to this value instead of the actual source core ID, range [0,3].
- **sx** – X direction, can be one of: [DYNAPSE_CONFIG_SRAM_DIRECTION_X_EAST,DYNAPSE_CONFIG_S
- **dx** – X delta, number of chips to jumps before reaching destination, range is [0,3].
- **sy** – Y direction, can be one of: [DYNAPSE_CONFIG_SRAM_DIRECTION_Y_NORTH,DYNAPSE_CONFIG_
- **dy** – Y delta, number of chips to jumps before reaching destination, range is [0,3].
- **destinationCore** – spike destination core, uses one-hot coding for the 4 cores: [C3,C2,C1,C0] -> [0,0,0,0] (0 decimal) no core, [1,1,1,1] (15 decimal) all cores

Returns
true on success, false otherwise.

LIBRARY_PUBLIC_VISIBILITY bool caerDynapseWriteCam (caerDeviceHandle handle, uint16_t inputNeuronAddr, uint16_t neuronAddr, uint8_t camId, uint8_t synapseType)

Write a single CAM, to specify which spikes are allowed as input into a neuron.

Remember to select the chip you want to configure before calling this function!

Parameters

- **handle** – a valid device handle.
- **inputNeuronAddr** – the neuron address that should be let in as input to this neuron, range [0,1023].
- **neuronAddr** – the neuron address whose CAM should be programmed, range [0,1023].
- **camId** – CAM address (synapse), each neuron has 64, range [0,63].
- **synapseType** – one of the four possible synaptic weights: [DYNAPSE_CONFIG_CAMTYPE_F_EXC,DYNAPSE_CONFIG_CAMTYPE_S_EXC,DYNAPSE_CONFIG_CA

Returns
true on success, false otherwise.

LIBRARY_PUBLIC_VISIBILITY bool caerDynapseSendDataToUSB (caerDeviceHandle handle, const uint32_t *data, size_t numConfig)

Send array of configuration parameters to the device via USB.

Remember to select the chip you want to configure before calling this function!

Parameters

- **handle** – a valid device handle.
- **data** – an array of integers holding configuration data.
- **numConfig** – number of configuration parameters to send.

Returns
true on success, false otherwise.

LIBRARY_PUBLIC_VISIBILITY uint32_t caerDynapseGenerateCamBits (uint16_t inputNeuronAddr, uint16_t neuronAddr, uint8_t camId, uint8_t synapseType)

Generate bits to write a single CAM, to specify which spikes are allowed as input into a neuron.

Parameters

- **inputNeuronAddr** – the neuron address that should be let in as input to this neuron, range [0,1023] (use *caerDynapseCoreXYToNeuronId()* for a 2D mapping).
- **neuronAddr** – the neuron to program, range [0,1023] (use *caerDynapseCoreXYToNeuronId()* for a 2D mapping).
- **camId** – CAM address (synapse), each neuron has 64, range [0,63].
- **synapseType** – one of the four possible synaptic weights: [DYNAPSE_CONFIG_CAMTYPE_F_EXC,DYNAPSE_CONFIG_CAMTYPE_S_EXC,DYNAPSE_CONFIG_CA

Returns
bits to send to device.

LIBRARY_PUBLIC_VISIBILITY uint32_t caerDynapseGenerateSramBits (uint16_t neuronAddr, uint8_t sramId, uint8_t virtualCoreId, bool sx, uint8_t dx, bool sy, uint8_t dy, uint8_t destinationCore)

Generate bits to write one of the 4 SRAMs of a single neuron. Writing the SRAM means writing the destination address of where the spikes will be routed to. This works on the on-chip SRAM!

Parameters

- **neuronAddr** – the neuron to program, range [0,1023] (use *caerDynapseCoreXYToNeuronId()* for a 2D mapping).
- **sramId** – SRAM address (one of four cells), range [0,3].
- **virtualCoreId** – fake source core ID, set it to this value instead of the actual source core ID, range [0,3].
- **sx** – X direction, can be one of: [DYNAPSE_CONFIG_SRAM_DIRECTION_X_EAST,DYNAPSE_CONFIG_S
- **dx** – X delta, number of chips to jumps before reaching destination, range is [0,3].
- **sy** – Y direction, can be one of: [DYNAPSE_CONFIG_SRAM_DIRECTION_Y_NORTH,DYNAPSE_CONFIG_
- **dy** – Y delta, number of chips to jumps before reaching destination, range is [0,3].
- **destinationCore** – spike destination core, uses one-hot coding for the 4 cores: [C3,C2,C1,C0] -> [0,0,0,0] (0 decimal) no core, [1,1,1,1] (15 decimal) all cores

Returns
bits to send to device.

LIBRARY_PUBLIC_VISIBILITY uint16_t caerDynapseCoreXYToNeuronId (uint8_t coreId, uint8_t columnX, uint8_t rowY)

Map core ID and column/row address to the correct chip global neuron address.

Parameters

- **coreId** – the chip's core ID, range [0,3].
- **columnX** – the neuron's column address, range [0,15].
- **rowY** – the neuron's row address, range [0,15].

Returns
chip global neuron address.

LIBRARY_PUBLIC_VISIBILITY uint16_t caerDynapseCoreAddrToNeuronId (uint8_t coreId, uint8_t neuronAddrCore)

Map core ID and per-core neuron address to the correct chip global neuron address.

Parameters

- **coreId** – the chip's core ID, range [0,3].
- **neuronAddrCore** – the neuron's address within this core, range [0,255].

Returns
chip global neuron address.

`LIBRARY_PUBLIC_VISIBILITY uint16_t caerDynapseSpikeEventGetX (caerSpikeEventConst event)`

Get the X (column) address for a spike event, in pixels. The (0, 0) address is in the upper left corner.

Parameters
event – a valid SpikeEvent pointer. Cannot be NULL.

Returns
the event X address in pixels.

`LIBRARY_PUBLIC_VISIBILITY uint16_t caerDynapseSpikeEventGetY (caerSpikeEventConst event)`

Get the Y (row) address for a spike event, in pixels. The (0, 0) address is in the upper left corner.

Parameters
event – a valid SpikeEvent pointer. Cannot be NULL.

Returns
the event Y address in pixels.

`LIBRARY_PUBLIC_VISIBILITY struct caer_spike_event caerDynapseSpikeEventFromXY (uint16_t x, uint16_t y)`

Get the chip ID, core ID and neuron ID from the X and Y coordinates. This is the reverse transform to: *caerDynapseSpikeEventGetX()* / *caerDynapseSpikeEventGetY()*. The return value is a 'struct caer_spike_event' because it already has functions to get/set all the needed values.

Parameters

- **x** – a X coordinate as returned by *caerDynapseSpikeEventGetX()*.
- **y** – a Y coordinate as returned by *caerDynapseSpikeEventGetY()*.

Returns
a SpikeEvent struct holding chip ID, core ID and neuron ID.

file **edvs.h**

*#include "../events/polarity.h"#include "../events/special.h"#include "*serial.h*"* EDVS-4337 specific configuration defines and information structures.

[25] https://ai-ctx.com/support/'

Defines

CAER_DEVICE_EDVS

Device type definition for iniVation EDVS-4337.

EDVS_CONFIG_DVS

Module address: device-side DVS configuration.

EDVS_CONFIG_BIAS

Module address: device-side chip bias generator configuration.

EDVS_CONFIG_DVS_RUN

Parameter address for module EDVS_CONFIG_DVS: run the DVS chip and generate polarity event data.

EDVS_CONFIG_DVS_TIMESTAMP_RESET

Parameter address for module EDVS_CONFIG_DVS: reset the time-stamp counter of the device. This is a temporary configuration switch and will reset itself right away.

EDVS_CONFIG_BIAS_CAS

Parameter address for module EDVS_CONFIG_BIAS: First stage amplifier cascode bias. See 'https://inivation.com/support/hardware/biasing/'[26] for more details.

EDVS_CONFIG_BIAS_INJGND

Parameter address for module EDVS_CONFIG_BIAS: Injected ground bias. See 'https://inivation.com/support/hardware/biasing/'[27] for more details.

EDVS_CONFIG_BIAS_REQPD

Parameter address for module EDVS_CONFIG_BIAS: Pull down on chip request (AER). See 'https://inivation.com/support/hardware/biasing/'[28] for more details.

EDVS_CONFIG_BIAS_PUX

Parameter address for module EDVS_CONFIG_BIAS: Pull up on request from X arbiter (AER). See 'https://inivation.com/support/hardware/biasing/'[29] for more details.

EDVS_CONFIG_BIAS_DIFFOFF

Parameter address for module EDVS_CONFIG_BIAS: Off events threshold bias. See 'https://inivation.com/support/hardware/biasing/'[30] for more details.

EDVS_CONFIG_BIAS_REQ

Parameter address for module EDVS_CONFIG_BIAS: Pull down for passive load inverters in digital AER pixel circuitry. See 'https://inivation.com/support/hardware/biasing/'[31] for more details.

EDVS_CONFIG_BIAS_REFR

Parameter address for module EDVS_CONFIG_BIAS: Refractory period bias. See 'https://inivation.com/support/hardware/biasing/'[32] for more details.

EDVS_CONFIG_BIAS_PUY

Parameter address for module EDVS_CONFIG_BIAS: Pull up on request from Y arbiter (AER). See 'https://inivation.com/support/hardware/biasing/'[33] for more details.

EDVS_CONFIG_BIAS_DIFFON

Parameter address for module EDVS_CONFIG_BIAS: On events threshold bias. See 'https://inivation.com/support/hardware/biasing/'[34] for more details.

EDVS_CONFIG_BIAS_DIFF

Parameter address for module EDVS_CONFIG_BIAS: Differential (second stage amplifier) bias. See 'https://inivation.com/support/hardware/biasing/'[35] for more details.

EDVS_CONFIG_BIAS_FOLL

Parameter address for module EDVS_CONFIG_BIAS: Source follower bias. See 'https://inivation.com/support/hardware/biasing/'[36] for more details.

EDVS_CONFIG_BIAS_PR

Parameter address for module EDVS_CONFIG_BIAS: Photoreceptor bias. See 'https://inivation.com/support/hardware/biasing/'[37] for more details.

Functions

LIBRARY_PUBLIC_VISIBILITY struct caer_edvs_info caerEDVSInfoGet (caerDeviceHandle handle)

Return basic information on the device, such as its ID, its resolution, the logic version, and so on. See the 'struct *caer_edvs_info*' documentation for more details.

Parameters
handle – a valid device handle.

Returns
a copy of the device information structure if successful, an empty structure (all zeros) on failure.

file **imu_support.h**

[26] https://inivation.com/support/hardware/biasing/'
[27] https://inivation.com/support/hardware/biasing/'
[28] https://inivation.com/support/hardware/biasing/'
[29] https://inivation.com/support/hardware/biasing/'
[30] https://inivation.com/support/hardware/biasing/'
[31] https://inivation.com/support/hardware/biasing/'
[32] https://inivation.com/support/hardware/biasing/'
[33] https://inivation.com/support/hardware/biasing/'
[34] https://inivation.com/support/hardware/biasing/'
[35] https://inivation.com/support/hardware/biasing/'
[36] https://inivation.com/support/hardware/biasing/'
[37] https://inivation.com/support/hardware/biasing/'

Enums

enum **caer_imu_types**

List of supported IMU models.

Values:

enumerator **IMU_NONE**

enumerator **IMU_INVENSENSE_6050_6150**

enumerator **IMU_INVENSENSE_9250**

enumerator **IMU_BOSCH_BMI_160**

enum **caer_imu_invensense_accel_scale**

List of accelerometer scale settings for InvenSense IMUs.

Values:

enumerator **ACCEL_2G**

enumerator **ACCEL_4G**

enumerator **ACCEL_8G**

enumerator **ACCEL_16G**

enum **caer_imu_invensense_gyro_scale**

List of gyroscope scale settings for InvenSense IMUs.

Values:

enumerator **GYRO_250DPS**

enumerator **GYRO_500DPS**

enumerator **GYRO_1000DPS**

enumerator **GYRO_2000DPS**

enum **caer_imu_bosch_accel_scale**

List of accelerometer scale settings for Bosch IMU.

Values:

enumerator **BOSCH_ACCEL_2G**

enumerator **BOSCH_ACCEL_4G**

enumerator **BOSCH_ACCEL_8G**

enumerator **BOSCH_ACCEL_16G**

enum **caer_imu_bosch_accel_data_rate**

List of accelerometer data rate settings for Bosch IMU.

Values:

enumerator **BOSCH_ACCEL_12_5HZ**

enumerator **BOSCH_ACCEL_25HZ**

enumerator **BOSCH_ACCEL_50HZ**

enumerator **BOSCH_ACCEL_100HZ**

enumerator **BOSCH_ACCEL_200HZ**

enumerator **BOSCH_ACCEL_400HZ**

enumerator **BOSCH_ACCEL_800HZ**

enumerator **BOSCH_ACCEL_1600HZ**

enum **caer_imu_bosch_accel_filter**

List of accelerometer filter settings for Bosch IMU.

Values:

enumerator **BOSCH_ACCEL_OSR4**

enumerator **BOSCH_ACCEL_OSR2**

enumerator **BOSCH_ACCEL_NORMAL**

enum **caer_imu_bosch_gyro_scale**

List of gyroscope scale settings for Bosch IMU.

Values:

enumerator **BOSCH_GYRO_2000DPS**

enumerator **BOSCH_GYRO_1000DPS**

enumerator **BOSCH_GYRO_500DPS**

enumerator **BOSCH_GYRO_250DPS**

enumerator **BOSCH_GYRO_125DPS**

enum **caer_imu_bosch_gyro_data_rate**

List of gyroscope data rate settings for Bosch IMU.

Values:

enumerator **BOSCH_GYRO_25HZ**

enumerator **BOSCH_GYRO_50HZ**

enumerator **BOSCH_GYRO_100HZ**

enumerator **BOSCH_GYRO_200HZ**

enumerator **BOSCH_GYRO_400HZ**

enumerator **BOSCH_GYRO_800HZ**

enumerator **BOSCH_GYRO_1600HZ**

enumerator **BOSCH_GYRO_3200HZ**

enum **caer_imu_bosch_gyro_filter**

List of gyroscope filter settings for Bosch IMU.

Values:

enumerator **BOSCH_GYRO_OSR4**

enumerator **BOSCH_GYRO_OSR2**

enumerator **BOSCH_GYRO_NORMAL**

file **samsung_evk.h**

#include "../events/polarity.h"#include "../events/special.h"#include "usb.h" SAMSUNG_EVK specific configuration defines and information structures.

Defines

CAER_DEVICE_SAMSUNG_EVK

Device type definition for Samsung EVK.

SAMSUNG_EVK_CHIP_ID

Samsung chip identifier. 640x480, semi-synchronous readout.

SAMSUNG_EVK_DVS

SAMSUNG_EVK_DVS_MODE

SAMSUNG_EVK_DVS_EVENT_FLATTEN

SAMSUNG_EVK_DVS_EVENT_ON_ONLY

SAMSUNG_EVK_DVS_EVENT_OFF_ONLY

SAMSUNG_EVK_DVS_SUBSAMPLE_ENABLE

SAMSUNG_EVK_DVS_AREA_BLOCKING_ENABLE

SAMSUNG_EVK_DVS_DUAL_BINNING_ENABLE

SAMSUNG_EVK_DVS_SUBSAMPLE_VERTICAL

SAMSUNG_EVK_DVS_SUBSAMPLE_HORIZONTAL

SAMSUNG_EVK_DVS_AREA_BLOCKING_0

SAMSUNG_EVK_DVS_AREA_BLOCKING_1

SAMSUNG_EVK_DVS_AREA_BLOCKING_2

SAMSUNG_EVK_DVS_AREA_BLOCKING_3

SAMSUNG_EVK_DVS_AREA_BLOCKING_4

SAMSUNG_EVK_DVS_AREA_BLOCKING_5

SAMSUNG_EVK_DVS_AREA_BLOCKING_6

SAMSUNG_EVK_DVS_AREA_BLOCKING_7

SAMSUNG_EVK_DVS_AREA_BLOCKING_8

SAMSUNG_EVK_DVS_AREA_BLOCKING_9

SAMSUNG_EVK_DVS_AREA_BLOCKING_10

SAMSUNG_EVK_DVS_AREA_BLOCKING_11

SAMSUNG_EVK_DVS_AREA_BLOCKING_12

SAMSUNG_EVK_DVS_AREA_BLOCKING_13

SAMSUNG_EVK_DVS_AREA_BLOCKING_14

SAMSUNG_EVK_DVS_AREA_BLOCKING_15

SAMSUNG_EVK_DVS_AREA_BLOCKING_16

SAMSUNG_EVK_DVS_AREA_BLOCKING_17

SAMSUNG_EVK_DVS_AREA_BLOCKING_18

SAMSUNG_EVK_DVS_AREA_BLOCKING_19

SAMSUNG_EVK_DVS_TIMESTAMP_RESET

SAMSUNG_EVK_DVS_GLOBAL_RESET_ENABLE

SAMSUNG_EVK_DVS_GLOBAL_RESET_DURING_READOUT

SAMSUNG_EVK_DVS_GLOBAL_HOLD_ENABLE

SAMSUNG_EVK_DVS_FIXED_READ_TIME_ENABLE

SAMSUNG_EVK_DVS_EXTERNAL_TRIGGER_MODE

SAMSUNG_EVK_DVS_TIMING_ED

SAMSUNG_EVK_DVS_TIMING_GH2GRS

SAMSUNG_EVK_DVS_TIMING_GRS

SAMSUNG_EVK_DVS_TIMING_GH2SEL

SAMSUNG_EVK_DVS_TIMING_SELW

SAMSUNG_EVK_DVS_TIMING_SEL2AY_R

SAMSUNG_EVK_DVS_TIMING_SEL2AY_F

SAMSUNG_EVK_DVS_TIMING_SEL2R_R

SAMSUNG_EVK_DVS_TIMING_SEL2R_F

SAMSUNG_EVK_DVS_TIMING_NEXT_SEL

SAMSUNG_EVK_DVS_TIMING_NEXT_GH

SAMSUNG_EVK_DVS_TIMING_READ_FIXED

SAMSUNG_EVK_DVS_MODE_OFF

SAMSUNG_EVK_DVS_MODE_MONITOR

SAMSUNG_EVK_DVS_MODE_STREAM

SAMSUNG_EVK_DVS_EXTERNAL_TRIGGER_MODE_TIMESTAMP_RESET

SAMSUNG_EVK_DVS_EXTERNAL_TRIGGER_MODE_SINGLE_FRAME

SAMSUNG_EVK_DVS_SUBSAMPLE_VERTICAL_NONE

SAMSUNG_EVK_DVS_SUBSAMPLE_VERTICAL_HALF

SAMSUNG_EVK_DVS_SUBSAMPLE_VERTICAL_FOURTH

SAMSUNG_EVK_DVS_SUBSAMPLE_VERTICAL_EIGHTH

SAMSUNG_EVK_DVS_SUBSAMPLE_HORIZONTAL_NONE

SAMSUNG_EVK_DVS_SUBSAMPLE_HORIZONTAL_HALF

SAMSUNG_EVK_DVS_SUBSAMPLE_HORIZONTAL_FOURTH

SAMSUNG_EVK_DVS_SUBSAMPLE_HORIZONTAL_EIGHTH

SAMSUNG_EVK_DVS_CROPPER

SAMSUNG_EVK_DVS_CROPPER_ENABLE

SAMSUNG_EVK_DVS_CROPPER_Y_START_ADDRESS

SAMSUNG_EVK_DVS_CROPPER_Y_END_ADDRESS

SAMSUNG_EVK_DVS_CROPPER_X_START_ADDRESS

SAMSUNG_EVK_DVS_CROPPER_X_END_ADDRESS

SAMSUNG_EVK_DVS_ACTIVITY_DECISION

SAMSUNG_EVK_DVS_ACTIVITY_DECISION_ENABLE

SAMSUNG_EVK_DVS_ACTIVITY_DECISION_POS_THRESHOLD

SAMSUNG_EVK_DVS_ACTIVITY_DECISION_NEG_THRESHOLD

SAMSUNG_EVK_DVS_ACTIVITY_DECISION_DEC_RATE

SAMSUNG_EVK_DVS_ACTIVITY_DECISION_DEC_TIME

SAMSUNG_EVK_DVS_ACTIVITY_DECISION_POS_MAX_COUNT

SAMSUNG_EVK_DVS_BIAS

SAMSUNG_EVK_DVS_BIAS_CURRENT_RANGE_LOG

SAMSUNG_EVK_DVS_BIAS_CURRENT_RANGE_SF

SAMSUNG_EVK_DVS_BIAS_CURRENT_RANGE_ON

SAMSUNG_EVK_DVS_BIAS_CURRENT_RANGE_nRST

SAMSUNG_EVK_DVS_BIAS_CURRENT_RANGE_LOGA

SAMSUNG_EVK_DVS_BIAS_CURRENT_RANGE_LOGD

SAMSUNG_EVK_DVS_BIAS_CURRENT_LEVEL_SF

SAMSUNG_EVK_DVS_BIAS_CURRENT_LEVEL_nOFF

SAMSUNG_EVK_DVS_BIAS_CURRENT_AMP

SAMSUNG_EVK_DVS_BIAS_CURRENT_ON

SAMSUNG_EVK_DVS_BIAS_CURRENT_OFF

SAMSUNG_EVK_DVS_BIAS_SIMPLE

SAMSUNG_EVK_DVS_BIAS_CURRENT_RANGE_LOG_5uA

SAMSUNG_EVK_DVS_BIAS_CURRENT_RANGE_LOG_50uA

SAMSUNG_EVK_DVS_BIAS_CURRENT_RANGE_SF_0_5uA

SAMSUNG_EVK_DVS_BIAS_CURRENT_RANGE_SF_5uA

SAMSUNG_EVK_DVS_BIAS_CURRENT_RANGE_ON_5uA

SAMSUNG_EVK_DVS_BIAS_CURRENT_RANGE_ON_50uA

SAMSUNG_EVK_DVS_BIAS_CURRENT_RANGE_nRST_0_5uA

SAMSUNG_EVK_DVS_BIAS_CURRENT_RANGE_nRST_5uA

SAMSUNG_EVK_DVS_BIAS_CURRENT_RANGE_LOGA_5uA

SAMSUNG_EVK_DVS_BIAS_CURRENT_RANGE_LOGA_50uA

SAMSUNG_EVK_DVS_BIAS_CURRENT_RANGE_LOGD_5uA

SAMSUNG_EVK_DVS_BIAS_CURRENT_RANGE_LOGD_50uA

SAMSUNG_EVK_DVS_BIAS_CURRENT_RANGE_LOGD_500uA

SAMSUNG_EVK_DVS_BIAS_CURRENT_LEVEL_SF_x0_1

SAMSUNG_EVK_DVS_BIAS_CURRENT_LEVEL_SF_x1

SAMSUNG_EVK_DVS_BIAS_CURRENT_LEVEL_nOFF_x0_1

SAMSUNG_EVK_DVS_BIAS_CURRENT_LEVEL_nOFF_x1

SAMSUNG_EVK_DVS_BIAS_SIMPLE_VERY_LOW

SAMSUNG_EVK_DVS_BIAS_SIMPLE_LOW

SAMSUNG_EVK_DVS_BIAS_SIMPLE_DEFAULT

SAMSUNG_EVK_DVS_BIAS_SIMPLE_HIGH

SAMSUNG_EVK_DVS_BIAS_SIMPLE_VERY_HIGH

Functions

LIBRARY_PUBLIC_VISIBILITY struct caer_samsung_evk_info caerSamsungEVKInfoGet (caerDeviceHandle hand

Return basic information on the device, such as its ID, its resolution, the logic version, and so on. See the 'struct *caer_samsung_evk_info*' documentation for more details.

Parameters
handle – a valid device handle.

Returns
a copy of the device information structure if successful, an empty structure (all zeros) on failure.

file **serial.h**

*#include "*device.h*"* Common functions to access, configure and exchange data with supported serial port devices. Also contains defines for serial port specific configuration options.

Defines

CAER_HOST_CONFIG_SERIAL

Module address: host-side serial port configuration.

CAER_HOST_CONFIG_SERIAL_READ_SIZE

Parameter address for module CAER_HOST_CONFIG_SERIAL: read size for serial port communication.

CAER_HOST_CONFIG_SERIAL_BAUD_RATE_2M

Parameter values for module CAER_HOST_CONFIG_SERIAL: possible baud-rates for serial port communication.

CAER_HOST_CONFIG_SERIAL_BAUD_RATE_4M

CAER_HOST_CONFIG_SERIAL_BAUD_RATE_8M

CAER_HOST_CONFIG_SERIAL_BAUD_RATE_12M

Functions

LIBRARY_PUBLIC_VISIBILITY caerDeviceHandle caerDeviceOpenSerial (uint16_t deviceID, uint16_t deviceType, const char *serialPortName, uint32_t serialBaudRate)

Open a specified serial port device, assign an ID to it and return a handle for further usage. Various means can be employed to limit the selection of the device.

Parameters

- **deviceID** – a unique ID to identify the device from others. Will be used as the source for EventPackets being generated from its data.
- **deviceType** – type of the device to open. Currently supported are: CAER_DEVICE_EDVS
- **serialPortName** – name of the serial port device to open.
- **serialBaudRate** – baud-rate for serial port communication.

Returns

a valid device handle that can be used with the other libcaer functions, or NULL on error. Always check for this! On error, errno is also set to provide more precise information about the failure cause.

file **usb.h**

*#include "*device.h*"* Common functions to access, configure and exchange data with supported USB devices. Also contains defines for USB specific configuration options.

Defines

CAER_HOST_CONFIG_USB

Module address: host-side USB configuration.

CAER_HOST_CONFIG_USB_BUFFER_NUMBER

Parameter address for module CAER_HOST_CONFIG_USB: set number of buffers used by libusb for asynchronous data transfers with the USB device. The default values are usually fine, only change them if you're running into I/O limits.

CAER_HOST_CONFIG_USB_BUFFER_SIZE

Parameter address for module CAER_HOST_CONFIG_USB: set size of each buffer used by libusb for asynchronous data transfers with the USB device. The default values are usually fine, only change them if you're running into I/O limits.

Functions

LIBRARY_PUBLIC_VISIBILITY caerDeviceHandle caerDeviceOpen (uint16_t deviceID, uint16_t deviceType, uint8_t busNumberRestrict, uint8_t devAddressRestrict, const char *serialNumberRestrict)

Open a specified USB device, assign an ID to it and return a handle for further usage. Various means can be employed to limit the selection of the device.

Parameters

- **deviceID** – a unique ID to identify the device from others. Will be used as the source for EventPackets being generated from its data.
- **deviceType** – type of the device to open. Currently supported are: CAER_DEVICE_DVS128, CAER_DEVICE_DAVIS, CAER_DEVICE_DYNAPSE, CAER_DEVICE_DVS132S, CAER_DEVICE_DVXPLORER, CAER_DEVICE_SAMSUNG_EVK
- **busNumberRestrict** – restrict the search for viable devices to only this USB bus number.
- **devAddressRestrict** – restrict the search for viable devices to only this USB device address.
- **serialNumberRestrict** – restrict the search for viable devices to only devices which do possess the given Serial Number in their USB SerialNumber descriptor.

Returns

a valid device handle that can be used with the other libcaer functions, or NULL on error. Always check for this! On error, errno is also set to provide more precise information about the failure cause.

file **common.h**

#include "../libcaer.h" Common EventPacket header format definition and handling functions. Every EventPacket, of any type, has as a first member a common header, which describes various properties of the contained events. This allows easy parsing of events. See the 'struct caer_event_packet_header' documentation for more details.

Defines

caerLogEHO

VALID_MARK_SHIFT

Generic validity mark: this bit is used to mark whether an event is still valid or not, and can be used to efficiently filter out events from a packet. The caerXXXEventValidate() and caerXXXEventInvalidate() functions should be used to toggle this! 0 in the 0th bit of the first byte means invalid, 1 means valid. This way zeroing-out an event packet sets all its events to invalid. Care must be taken to put the field containing the validity mark always as the first member of an event.

VALID_MARK_MASK

TS_OVERFLOW_SHIFT

64bit timestamp support: since timestamps wrap around after some time, being only 31 bit (32 bit signed int), another timestamp at the packet level provides another 31 bit (32 bit signed int), to enable the generation

of a 62 bit (64 bit signed int) microsecond timestamp which is guaranteed to never wrap around (in the next 146'138 years at least). The TSOverflow needs to be shifted by 31 thus when constructing such a timestamp.

CAER_DEFAULT_EVENT_TYPES_COUNT

Number of default event types that are part of libcaer. Corresponds to the count of definitions inside the 'enum caer_default_event_types' enumeration.

CAER_EVENT_PACKET_HEADER_SIZE

Size of the EventPacket header. This is constant across all supported systems.

CAER_ITERATOR_ALL_START(PACKET_HEADER, EVENT_TYPE)

Generic iterator over all events in a packet. Returns the current index in the 'caerIteratorCounter' variable of type 'int32_t' and the current event in the 'caerIteratorElement' variable of type EVENT_TYPE.

PACKET_HEADER: a valid EventPacket header pointer. Cannot be NULL. EVENT_TYPE: the event pointer type for this EventPacket (ie. caerPolarityEvent or caerFrameEvent).

CAER_ITERATOR_ALL_END

Generic iterator close statement.

CAER_ITERATOR_VALID_START(PACKET_HEADER, EVENT_TYPE)

Generic iterator over only the valid events in a packet. Returns the current index in the 'caerIteratorCounter' variable of type 'int32_t' and the current event in the 'caerIteratorElement' variable of type EVENT_TYPE.

PACKET_HEADER: a valid EventPacket header pointer. Cannot be NULL. EVENT_TYPE: the event pointer type for this EventPacket (ie. caerPolarityEvent or caerFrameEvent).

CAER_ITERATOR_VALID_END

Generic iterator close statement.

Typedefs

typedef struct caer_event_packet_header ***caerEventPacketHeader**

Type for pointer to EventPacket header data structure.

typedef const struct caer_event_packet_header ***caerEventPacketHeaderConst**

Enums

enum **caer_default_event_types**

List of supported event types. Each event type has its own integer representation. All event types below 100 are reserved for use by libcaer and cAER. DO NOT USE THEM FOR YOUR OWN EVENT TYPES!

Values:

enumerator **SPECIAL_EVENT**

Special events.

enumerator **POLARITY_EVENT**

Polarity (change, DVS) events.

enumerator **FRAME_EVENT**

Frame (intensity, APS) events.

enumerator **IMU6_EVENT**

6 axes IMU events.

enumerator **IMU9_EVENT**

9 axes IMU events.

enumerator **SAMPLE_EVENT**

ADC sample events (deprecated).

enumerator **EAR_EVENT**

Ear (cochlea) events (deprecated).

enumerator **CONFIG_EVENT**

Device configuration events (deprecated).

enumerator **POINT1D_EVENT**

1D measurement events (deprecated).

enumerator **POINT2D_EVENT**

2D measurement events (deprecated).

enumerator **POINT3D_EVENT**

3D measurement events (deprecated).

enumerator **POINT4D_EVENT**

4D measurement events (deprecated).

enumerator **SPIKE_EVENT**

Spike events.

enumerator **MATRIX4x4_EVENT**

4D matrix events (deprecated).

Functions

PACKED_STRUCT (struct caer_event_packet_header { int16_t eventType; int16_t eventSource;int32_t eventSize;int32_t eventTSOffset;int32_t eventTSOverflow; int32_t eventCapacity;int32_t eventNumber;int32_t eventValid;})

EventPacket header data structure definition. The size, also defined in CAER_EVENT_PACKET_HEADER_SIZE, must always be constant. The header is common to all types of event packets and is always the very first member of an event packet data structure. Signed integers are used for compatibility with languages that do not have unsigned ones, such as Java.

static inline int16_t **caerEventPacketHeaderGetEventType**(*caerEventPacketHeaderConst* header)

Return the numerical event type ID, representing the event type this EventPacket is containing.

Parameters
header – a valid EventPacket header pointer. Cannot be NULL.

Returns
the numerical event type (see 'enum caer_default_event_types').

static inline void **caerEventPacketHeaderSetEventType**(*caerEventPacketHeader* header, int16_t eventType)

Set the numerical event type ID, representing the event type this EventPacket will contain. All event types below 100 are reserved for use by libcaer and cAER. DO NOT USE THEM FOR YOUR OWN EVENT TYPES!

Parameters

- **header** – a valid EventPacket header pointer. Cannot be NULL.
- **eventType** – the numerical event type (see 'enum caer_default_event_types').

static inline int16_t **caerEventPacketHeaderGetEventSource**(*caerEventPacketHeaderConst* header)

Get the numerical event source ID, representing the event source that generated all the events present in this packet.

Parameters
header – a valid EventPacket header pointer. Cannot be NULL.

Returns
the numerical event source ID.

static inline void **caerEventPacketHeaderSetEventSource**(*caerEventPacketHeader* header, int16_t eventSource)

Set the numerical event source ID, representing the event source that generated all the events present in this packet. This ID should be unique at least within a process, if not within the whole system, to guarantee correct identification of who generated an event later on.

Parameters

- **header** – a valid EventPacket header pointer. Cannot be NULL.
- **eventSource** – the numerical event source ID.

static inline int32_t **caerEventPacketHeaderGetEventSize**(*caerEventPacketHeaderConst* header)

Get the size of a single event, in bytes. All events inside an event packet always have the same size.

Parameters
header – a valid EventPacket header pointer. Cannot be NULL.

Returns
the event size in bytes.

static inline void **caerEventPacketHeaderSetEventSize**(*caerEventPacketHeader* header, int32_t eventSize)

Set the size of a single event, in bytes. All events inside an event packet always have the same size.

Parameters

- **header** – a valid EventPacket header pointer. Cannot be NULL.
- **eventSize** – the event size in bytes.

static inline int32_t **caerEventPacketHeaderGetEventTSOffset**(*caerEventPacketHeaderConst* header)

Get the offset, in bytes, to where the field with the main 32 bit timestamp is stored. This is useful for generic access to the timestamp field, given that different event types might have it at different offsets or might even have multiple timestamps, in which case this offset references the 'main' timestamp, the most representative one.

Parameters
header – a valid EventPacket header pointer. Cannot be NULL.

Returns
the event timestamp offset in bytes.

static inline void **caerEventPacketHeaderSetEventTSOffset**(*caerEventPacketHeader* header, int32_t eventTSOffset)

Set the offset, in bytes, to where the field with the main 32 bit timestamp is stored. This is useful for generic access to the timestamp field, given that different event types might have it at different offsets or might even have multiple timestamps, in which case this offset references the 'main' timestamp, the most representative one.

Parameters

- **header** – a valid EventPacket header pointer. Cannot be NULL.
- **eventTSOffset** – the event timestamp offset in bytes.

static inline int32_t **caerEventPacketHeaderGetEventTSOverflow**(*caerEventPacketHeaderConst* header)

Get the 32 bit timestamp overflow counter (in microseconds). This is per-packet and is used to generate a 64 bit timestamp that never wraps around. Since timestamps wrap around after some time, being only 31 bit (32 bit signed int), another timestamp at the packet level provides another 31 bit (32 bit signed int), to enable the generation of a 62 bit (64 bit signed int) microsecond timestamp which is guaranteed to never wrap around (in the next 146'138 years at least).

Parameters
header – a valid EventPacket header pointer. Cannot be NULL.

Returns
the packet-level timestamp overflow counter, in microseconds.

static inline void **caerEventPacketHeaderSetEventTSOverflow**(*caerEventPacketHeader* header, int32_t eventTSOverflow)

Set the 32 bit timestamp overflow counter (in microseconds). This is per-packet and is used to generate a 64 bit timestamp that never wraps around. Since timestamps wrap around after some time, being only 31 bit (32 bit signed int), another timestamp at the packet level provides another 31 bit (32 bit signed int), to enable the generation of a 62 bit (64 bit signed int) microsecond timestamp which is guaranteed to never wrap around (in the next 146'138 years at least).

Parameters

- **header** – a valid EventPacket header pointer. Cannot be NULL.
- **eventTSOverflow** – the packet-level timestamp overflow counter, in microseconds.

static inline int32_t **caerEventPacketHeaderGetEventCapacity**(*caerEventPacketHeaderConst* header)

Get the maximum number of events this packet can store.

Parameters
header – a valid EventPacket header pointer. Cannot be NULL.

Returns
the number of events this packet can hold.

static inline void **caerEventPacketHeaderSetEventCapacity**(*caerEventPacketHeader* header, int32_t eventsCapacity)

Set the maximum number of events this packet can store. This is determined at packet allocation time and should not be changed during the life-time of the packet.

Parameters

- **header** – a valid EventPacket header pointer. Cannot be NULL.
- **eventsCapacity** – the number of events this packet can hold.

static inline int32_t **caerEventPacketHeaderGetEventNumber**(*caerEventPacketHeaderConst* header)

Get the number of events currently stored in this packet, considering both valid and invalid events.

Parameters
header – a valid EventPacket header pointer. Cannot be NULL.

Returns
the number of events in this packet.

static inline void **caerEventPacketHeaderSetEventNumber**(*caerEventPacketHeader* header, int32_t eventsNumber)

Set the number of events currently stored in this packet, considering both valid and invalid events.

Parameters

- **header** – a valid EventPacket header pointer. Cannot be NULL.
- **eventsNumber** – the number of events in this packet.

static inline int32_t **caerEventPacketHeaderGetEventValid**(*caerEventPacketHeaderConst* header)

Get the number of valid events in this packet, disregarding invalid ones (where the invalid mark is set).

Parameters
header – a valid EventPacket header pointer. Cannot be NULL.

Returns
the number of valid events in this packet.

static inline void **caerEventPacketHeaderSetEventValid**(*caerEventPacketHeader* header, int32_t eventsValid)

Set the number of valid events in this packet, disregarding invalid ones (where the invalid mark is set).

Parameters

- **header** – a valid EventPacket header pointer. Cannot be NULL.
- **eventsValid** – the number of valid events in this packet.

static inline const void *caerGenericEventGetEvent(*caerEventPacketHeaderConst* headerPtr, int32_t n)

Get a generic pointer to an event, without having to know what event type the packet is containing.

Parameters

- **headerPtr** – a valid EventPacket header pointer. Cannot be NULL.
- **n** – the index of the returned event. Must be within [0,eventNumber[bounds.

Returns

a generic pointer to the requested event. NULL on error. This points to unmodifiable memory, as it should never be used for anything other than read operations, such as caerGenericEventGetTimestamp(). Don't modify the memory, you have no idea what it is! If you do know, just use the proper typed packet functions.

static inline int32_t caerGenericEventGetTimestamp(const void *eventPtr, *caerEventPacketHeaderConst* headerPtr)

Get the main 32 bit timestamp for a generic event, without having to know what event type the packet is containing.

Parameters

- **eventPtr** – a generic pointer to an event. Cannot be NULL.
- **headerPtr** – a valid EventPacket header pointer. Cannot be NULL.

Returns

the main 32 bit timestamp of this event.

static inline int64_t caerGenericEventGetTimestamp64(const void *eventPtr, *caerEventPacketHeaderConst* headerPtr)

Get the main 64 bit timestamp for a generic event, without having to know what event type the packet is containing. This takes the per-packet timestamp into account too, generating a timestamp that doesn't suffer from overflow problems.

Parameters

- **eventPtr** – a generic pointer to an event. Cannot be NULL.
- **headerPtr** – a valid EventPacket header pointer. Cannot be NULL.

Returns

the main 64 bit timestamp of this event.

static inline bool caerGenericEventIsValid(const void *eventPtr)

Check if the given generic event is valid or not.

Parameters

eventPtr – a generic pointer to an event. Cannot be NULL.

Returns

true if the event is valid, false otherwise.

static inline bool caerGenericEventCopy(void *eventPtrDestination, const void *eventPtrSource, *caerEventPacketHeaderConst* headerPtrDestination, *caerEventPacketHeaderConst* headerPtrSource)

Copy a given event's content to another location in memory.

Parameters

- **eventPtrDestination** – a generic pointer to an event to copy to. Cannot be NULL.
- **eventPtrSource** – a generic pointer to an event to copy from. Cannot be NULL.

- **headerPtrDestination** – a valid EventPacket header pointer from the destination packet. Cannot be NULL.
- **headerPtrSource** – a valid EventPacket header pointer from the source packet. Cannot be NULL.

Returns
true on successful copy, false otherwise.

static inline int64_t **caerEventPacketGetDataSize**(*caerEventPacketHeaderConst* header)

Get the data size of an event packet, in bytes. This is only the size of the data portion, excluding the header.

Parameters
header – a valid EventPacket header pointer. Cannot be NULL.

Returns
the event packet data size in bytes.

static inline int64_t **caerEventPacketGetSize**(*caerEventPacketHeaderConst* header)

Get the full size of an event packet, in bytes. This includes both the header and the data portion.

Parameters
header – a valid EventPacket header pointer. Cannot be NULL.

Returns
the event packet size in bytes.

static inline int64_t **caerEventPacketGetDataSizeEvents**(*caerEventPacketHeaderConst* header)

Get the data size of an event packet, in bytes, up to its last actual event. This means only up to EventNumber, not up to EventCapacity. This is only the size of the data portion, excluding the header.

Parameters
header – a valid EventPacket header pointer. Cannot be NULL.

Returns
the event packet data size in bytes (up to event number).

static inline int64_t **caerEventPacketGetSizeEvents**(*caerEventPacketHeaderConst* header)

Get the full size of an event packet, in bytes, up to its last actual event. This means only up to EventNumber, not up to EventCapacity. This includes both the header and the data portion.

Parameters
header – a valid EventPacket header pointer. Cannot be NULL.

Returns
the event packet size in bytes (up to event number).

static inline bool **caerEventPacketEquals**(*caerEventPacketHeaderConst* firstPacket, *caerEventPacketHeaderConst* secondPacket)

Verify if two event packets are equal. This means that the header and all events are equal.

Parameters

- **firstPacket** – an event packet to be compared.
- **secondPacket** – the other event packet to compare against.

Returns
true if both are the same, false otherwise.

static inline void **caerEventPacketClear**(*caerEventPacketHeader* packet)

Clear a packet by zeroing out all events. Capacity doesn't change, event number is set to zero.

Parameters
packet – an event packet to clear out.

static inline void **caerEventPacketClean**(*caerEventPacketHeader* packet)

Clean a packet by removing all invalid events, so that the total number of events is the number of valid events. The packet's capacity doesn't change.

Parameters
packet – an event packet to clean.

static inline *caerEventPacketHeader* **caerEventPacketResize**(*caerEventPacketHeader* packet, int32_t newEventCapacity)

Resize an event packet. First, the packet is cleaned (all invalid events removed), then:

- If the old and new event capacity are equal, nothing else changes.
- If the new capacity is bigger, the packet is enlarged and the new events are initialized to all zeros (invalid).
- If the new capacity is smaller, the packet is truncated at the given point. Use free() to reclaim this memory afterwards.

Parameters

- **packet** – the current event packet.
- **newEventCapacity** – the new maximum number of events this packet can hold. Cannot be zero.

Returns
a valid event packet handle or NULL on error. On success, the old packet handle is to be considered invalid and not to be used anymore. On failure, the old packet handle is still valid, but will have been cleaned of all invalid events!

static inline *caerEventPacketHeader* **caerEventPacketGrow**(*caerEventPacketHeader* packet, int32_t newEventCapacity)

Grows an event packet. This only supports strictly increasing the size of a packet. For a more flexible resize operation, see caerEventPacketResize(). Use free() to reclaim this memory afterwards.

Parameters

- **packet** – the current event packet.
- **newEventCapacity** – the new maximum number of events this packet can hold. Cannot be zero.

Returns
a valid event packet handle or NULL on error. On success, the old packet handle is to be considered invalid and not to be used anymore. On failure, the old packet handle is not touched in any way.

static inline *caerEventPacketHeader* **caerEventPacketAppend**(*caerEventPacketHeader* packet, *caerEventPacketHeader* appendPacket)

Appends an event packet to another. This is a simple append operation, no timestamp reordering is done. Please ensure time is monotonically increasing over the two packets! Use free() to reclaim this memory afterwards.

Parameters

- **packet** – the main events packet.
- **appendPacket** – the events packet to append on the main one.

Returns
a valid event packet handle or NULL on error. On success, the old packet handle is to be considered invalid and not to be used anymore. On failure, the old packet handle is not touched in any way. The appendPacket handle is never touched in any way.

static inline *caerEventPacketHeader* **caerEventPacketCopy**(*caerEventPacketHeaderConst* packet)

Make a full copy of an event packet (up to eventCapacity).

Parameters
packet – an event packet to copy.

Returns
a full copy of an event packet.

static inline *caerEventPacketHeader* **caerEventPacketCopyOnlyEvents**(*caerEventPacketHeaderConst* packet)

Make a copy of an event packet, sized down to only include the currently present events (eventNumber, valid+invalid), and not including the possible extra unused events (up to eventCapacity).

Parameters
packet – an event packet to copy.

Returns
a sized down copy of an event packet.

static inline *caerEventPacketHeader* **caerEventPacketCopyOnlyValidEvents**(*caerEventPacketHeaderConst* packet)

Make a copy of an event packet, sized down to only include the currently valid events (eventValid), and discarding everything else.

Parameters
packet – an event packet to copy.

Returns
a copy of an event packet, containing only valid events.

static inline *caerEventPacketHeader* **caerEventPacketAllocate**(int32_t eventCapacity, int16_t eventSource, int32_t tsOverflow, int16_t eventType, int32_t eventSize, int32_t eventTSOffset)

Allocate memory for an event packet and fill its header with the proper information. THIS FUNCTION IS INTENDED FOR INTERNAL USE ONLY BY THE VARIOUS EVENT PACKET TYPES FOR MEMORY ALLOCATION.

Returns
memory for an event packet, NULL on error.

file **frame.h**

#include "common.h" Frame Events format definition and handling functions. This event type encodes intensity frames, like you would get from a normal APS camera. It supports multiple channels for color, color filter information, as well as multiple Regions of Interest (ROI). The (0, 0) pixel is in the upper left corner of the screen, like in OpenCV/computer graphics. The pixel array is laid out row by row (increasing X axis), going from top to bottom (increasing Y axis). To copy a frame event, the usual assignment operator = cannot be used. Please use caerGenericEventCopy() to copy frame events!

Defines

FRAME_COLOR_CHANNELS_SHIFT

Shift and mask values for the color channels number, the color filter arrangement and the ROI identifier contained in the 'info' field of the frame event. Multiple channels (RGB for example) are possible, see the 'enum caer_frame_event_color_channels'. To understand the original color filter arrangement to interpolate color images, see the 'enum caer_frame_event_color_filter'. Also, up to 128 different Regions of Interest (ROI) can be tracked. Bit 0 is the valid mark, see '*common.h*' for more details.

FRAME_COLOR_CHANNELS_MASK

FRAME_COLOR_FILTER_SHIFT

FRAME_COLOR_FILTER_MASK

FRAME_ROI_IDENTIFIER_SHIFT

FRAME_ROI_IDENTIFIER_MASK

CAER_FRAME_ITERATOR_ALL_START(FRAME_PACKET)

Iterator over all frame events in a packet. Returns the current index in the 'caerFrameIteratorCounter' variable of type 'int32_t' and the current event in the 'caerFrameIteratorElement' variable of type caerFrameEvent.

FRAME_PACKET: a valid FrameEventPacket pointer. Cannot be NULL.

CAER_FRAME_CONST_ITERATOR_ALL_START(FRAME_PACKET)

Const-Iterator over all frame events in a packet. Returns the current index in the 'caerFrameIteratorCounter' variable of type 'int32_t' and the current read-only event in the 'caerFrameIteratorElement' variable of type caerFrameEventConst.

FRAME_PACKET: a valid FrameEventPacket pointer. Cannot be NULL.

CAER_FRAME_ITERATOR_ALL_END

Iterator close statement.

CAER_FRAME_ITERATOR_VALID_START(FRAME_PACKET)

Iterator over only the valid frame events in a packet. Returns the current index in the 'caerFrameIteratorCounter' variable of type 'int32_t' and the current event in the 'caerFrameIteratorElement' variable of type caerFrameEvent.

FRAME_PACKET: a valid FrameEventPacket pointer. Cannot be NULL.

CAER_FRAME_CONST_ITERATOR_VALID_START(FRAME_PACKET)

Const-Iterator over only the valid frame events in a packet. Returns the current index in the 'caerFrameIteratorCounter' variable of type 'int32_t' and the current read-only event in the 'caerFrameIteratorElement' variable of type caerFrameEventConst.

FRAME_PACKET: a valid FrameEventPacket pointer. Cannot be NULL.

CAER_FRAME_ITERATOR_VALID_END

Iterator close statement.

CAER_FRAME_REVERSE_ITERATOR_ALL_START(FRAME_PACKET)

Reverse iterator over all frame events in a packet. Returns the current index in the 'caerFrameIteratorCounter' variable of type 'int32_t' and the current event in the 'caerFrameIteratorElement' variable of type caerFrameEvent.

FRAME_PACKET: a valid FrameEventPacket pointer. Cannot be NULL.

CAER_FRAME_CONST_REVERSE_ITERATOR_ALL_START(FRAME_PACKET)

Const-Reverse iterator over all frame events in a packet. Returns the current index in the 'caerFrameIteratorCounter' variable of type 'int32_t' and the current read-only event in the 'caerFrameIteratorElement' variable of type caerFrameEventConst.

FRAME_PACKET: a valid FrameEventPacket pointer. Cannot be NULL.

CAER_FRAME_REVERSE_ITERATOR_ALL_END

Reverse iterator close statement.

CAER_FRAME_REVERSE_ITERATOR_VALID_START(FRAME_PACKET)

Reverse iterator over only the valid frame events in a packet. Returns the current index in the 'caerFrameIteratorCounter' variable of type 'int32_t' and the current event in the 'caerFrameIteratorElement' variable of type caerFrameEvent.

FRAME_PACKET: a valid FrameEventPacket pointer. Cannot be NULL.

CAER_FRAME_CONST_REVERSE_ITERATOR_VALID_START(FRAME_PACKET)

Const-Reverse iterator over only the valid frame events in a packet. Returns the current index in the 'caerFrameIteratorCounter' variable of type 'int32_t' and the current read-only event in the 'caerFrameIteratorElement' variable of type caerFrameEventConst.

FRAME_PACKET: a valid FrameEventPacket pointer. Cannot be NULL.

CAER_FRAME_REVERSE_ITERATOR_VALID_END

Reverse iterator close statement.

Typedefs

typedef struct caer_frame_event ***caerFrameEvent**

Type for pointer to frame event data structure.

typedef const struct caer_frame_event ***caerFrameEventConst**

typedef struct caer_frame_event_packet ***caerFrameEventPacket**

Type for pointer to frame event packet data structure.

typedef const struct caer_frame_event_packet ***caerFrameEventPacketConst**

Enums

enum **caer_frame_event_color_channels**

List of all frame event color channel identifiers. Used to interpret the frame event color channel field.

Values:

enumerator **GRAYSCALE**

Grayscale, one channel only.

enumerator **RGB**

Red Green Blue, 3 color channels.

enumerator **RGBA**

Red Green Blue Alpha, 3 color channels plus transparency.

enum **caer_frame_event_color_filter**

List of all frame event color filter identifiers. Used to interpret the frame event color filter field.

Values:

enumerator **MONO**

No color filter present, all light passes.

enumerator **RGBG**

Standard Bayer color filter, 1 red 2 green 1 blue. Variation 1.

enumerator **GRGB**

Standard Bayer color filter, 1 red 2 green 1 blue. Variation 2.

enumerator **GBGR**

Standard Bayer color filter, 1 red 2 green 1 blue. Variation 3.

enumerator **BGRG**

Standard Bayer color filter, 1 red 2 green 1 blue. Variation 4.

enumerator **RGBW**

Modified Bayer color filter, with white (pass all light) instead of extra green. Variation 1.

enumerator **GRWB**

Modified Bayer color filter, with white (pass all light) instead of extra green. Variation 2.

enumerator **WBGR**

Modified Bayer color filter, with white (pass all light) instead of extra green. Variation 3.

enumerator **BWRG**

Modified Bayer color filter, with white (pass all light) instead of extra green. Variation 4.

Functions

PACKED_STRUCT (struct caer_frame_event { uint32_t info;int32_t ts_startframe; int32_t ts_endframe;int32_t ts_startexposure;int32_t ts_endexposure;int32_t lengthX; int32_t lengthY;int32_t positionX;int32_t positionY;uint16_t pixels[1];})

Frame event data structure definition. This contains the actual information on the frame (ROI, color channels, color filter), several timestamps to signal start and end of capture and of exposure, as well as the actual pixels, in a 16 bit normalized format. The (0, 0) address is in the upper left corner, like in OpenCV/computer graphics. The pixel array is laid out row by row (increasing X axis), going from top to bottom (increasing Y axis). Signed integers are used for fields that are to be interpreted directly, for compatibility with languages that do not have unsigned integer types, such as Java. To copy a frame event, the usual assignment operator = cannot be used. Please use caerGenericEventCopy() to copy frame events!

PACKED_STRUCT (struct caer_frame_event_packet { struct caer_event_packet_header packetHeader; })

Frame event packet data structure definition. EventPackets are always made up of the common packet header, followed by 'eventCapacity' events. Everything has to be in one contiguous memory block. Direct access to the events array is not possible for Frame events. To calculate position offsets, use the 'eventSize' field in the packet header.

static inline *caerFrameEventPacket* **caerFrameEventPacketAllocateNumPixels**(int32_t eventCapacity, int16_t eventSource, int32_t tsOverflow, int32_t maxNumPixels, int16_t maxChannelNumber)

Allocate a new frame events packet, passing the total number of maximum pixels instead of the maximum X/Y dimensions expected. Use free() to reclaim this memory. The frame events allocate memory for a maximum sized pixels array, depending on the parameters passed to this function, so that every event occupies the same amount of memory (constant size). The actual frames inside of it might be smaller than that, for example when using ROI, and their actual size is stored inside the frame event and should always be queried from there. The unused part of a pixels array is guaranteed to be zeros.

Parameters

- **eventCapacity** – the maximum number of events this packet will hold.
- **eventSource** – the unique ID representing the source/generator of this packet.
- **tsOverflow** – the current timestamp overflow counter value for this packet.
- **maxNumPixels** – the maximum number of pixels that can be held by a frame event.
- **maxChannelNumber** – the maximum expected number of channels for frames in this packet.

Returns

a valid FrameEventPacket handle or NULL on error.

static inline *caerFrameEventPacket* **caerFrameEventPacketAllocate**(int32_t eventCapacity, int16_t eventSource, int32_t tsOverflow, int32_t maxLengthX, int32_t maxLengthY, int16_t maxChannelNumber)

Allocate a new frame events packet. Use free() to reclaim this memory. The frame events allocate memory for a maximum sized pixels array, depending on the parameters passed to this function, so that every event

occupies the same amount of memory (constant size). The actual frames inside of it might be smaller than that, for example when using ROI, and their actual size is stored inside the frame event and should always be queried from there. The unused part of a pixels array is guaranteed to be zeros.

Parameters

- **eventCapacity** – the maximum number of events this packet will hold.
- **eventSource** – the unique ID representing the source/generator of this packet.
- **tsOverflow** – the current timestamp overflow counter value for this packet.
- **maxLengthX** – the maximum expected X axis size for frames in this packet.
- **maxLengthY** – the maximum expected Y axis size for frames in this packet.
- **maxChannelNumber** – the maximum expected number of channels for frames in this packet.

Returns

a valid FrameEventPacket handle or NULL on error.

static inline *caerFrameEventPacket* **caerFrameEventPacketFromPacketHeader**(*caerEventPacketHeader* header)

Transform a generic event packet header into a Frame event packet. This takes care of proper casting and checks that the packet type really matches the intended conversion type.

Parameters

header – a valid event packet header pointer. Cannot be NULL.

Returns

a properly converted, typed event packet pointer.

static inline *caerFrameEventPacketConst* **caerFrameEventPacketFromPacketHeaderConst**(*caerEventPacketHeaderConst* header)

Transform a generic read-only event packet header into a read-only Frame event packet. This takes care of proper casting and checks that the packet type really matches the intended conversion type.

Parameters

header – a valid read-only event packet header pointer. Cannot be NULL.

Returns

a properly converted, read-only typed event packet pointer.

static inline *caerFrameEvent* **caerFrameEventPacketGetEvent**(*caerFrameEventPacket* packet, int32_t n)

Get the frame event at the given index from the event packet. To copy a frame event, the usual assignment operator = cannot be used. Please use caerGenericEventCopy() to copy frame events!

Parameters

- **packet** – a valid FrameEventPacket pointer. Cannot be NULL.
- **n** – the index of the returned event. Must be within [0,eventCapacity[bounds.

Returns

the requested frame event. NULL on error.

static inline *caerFrameEventConst* **caerFrameEventPacketGetEventConst**(*caerFrameEventPacketConst* packet, int32_t n)

Get the frame event at the given index from the event packet. This is a read-only event, do not change its contents in any way! To copy a frame event, the usual assignment operator = cannot be used. Please use caerGenericEventCopy() to copy frame events!

Parameters

- **packet** – a valid FrameEventPacket pointer. Cannot be NULL.
- **n** – the index of the returned event. Must be within [0,eventCapacity[bounds.

Returns
the requested read-only frame event. NULL on error.

static inline int32_t **caerFrameEventGetTSStartOfFrame**(*caerFrameEventConst* event)

Get the 32bit start of frame capture timestamp, in microseconds. Be aware that this wraps around! You can either ignore this fact, or handle the special 'TIMESTAMP_WRAP' event that is generated when this happens, or use the 64bit timestamp which never wraps around. See 'caerEventPacketHeaderGetEventTSOverflow()' documentation for more details on the 64bit timestamp.

Parameters
event – a valid FrameEvent pointer. Cannot be NULL.

Returns
this event's 32bit microsecond start of frame timestamp.

static inline int64_t **caerFrameEventGetTSStartOfFrame64**(*caerFrameEventConst* event, *caerFrameEventPacketConst* packet)

Get the 64bit start of frame capture timestamp, in microseconds. See 'caerEventPacketHeaderGetEventTSOverflow()' documentation for more details on the 64bit timestamp.

Parameters

- **event** – a valid FrameEvent pointer. Cannot be NULL.
- **packet** – the FrameEventPacket pointer for the packet containing this event. Cannot be NULL.

Returns
this event's 64bit microsecond start of frame timestamp.

static inline void **caerFrameEventSetTSStartOfFrame**(*caerFrameEvent* event, int32_t startFrame)

Set the 32bit start of frame capture timestamp, the value has to be in microseconds.

Parameters

- **event** – a valid FrameEvent pointer. Cannot be NULL.
- **startFrame** – a positive 32bit microsecond timestamp.

static inline int32_t **caerFrameEventGetTSEndOfFrame**(*caerFrameEventConst* event)

Get the 32bit end of frame capture timestamp, in microseconds. Be aware that this wraps around! You can either ignore this fact, or handle the special 'TIMESTAMP_WRAP' event that is generated when this happens, or use the 64bit timestamp which never wraps around. See 'caerEventPacketHeaderGetEventTSOverflow()' documentation for more details on the 64bit timestamp.

Parameters
event – a valid FrameEvent pointer. Cannot be NULL.

Returns
this event's 32bit microsecond end of frame timestamp.

static inline int64_t **caerFrameEventGetTSEndOfFrame64**(*caerFrameEventConst* event, *caerFrameEventPacketConst* packet)

Get the 64bit end of frame capture timestamp, in microseconds. See 'caerEventPacketHeaderGetEventTSOverflow()' documentation for more details on the 64bit timestamp.

Parameters

- **event** – a valid FrameEvent pointer. Cannot be NULL.
- **packet** – the FrameEventPacket pointer for the packet containing this event. Cannot be NULL.

Returns
this event's 64bit microsecond end of frame timestamp.

static inline void **caerFrameEventSetTSEndOfFrame**(*caerFrameEvent* event, int32_t endFrame)

Set the 32bit end of frame capture timestamp, the value has to be in microseconds.

Parameters

- **event** – a valid FrameEvent pointer. Cannot be NULL.
- **endFrame** – a positive 32bit microsecond timestamp.

static inline int32_t **caerFrameEventGetTSStartOfExposure**(*caerFrameEventConst* event)

Get the 32bit start of exposure timestamp, in microseconds. Be aware that this wraps around! You can either ignore this fact, or handle the special 'TIMESTAMP_WRAP' event that is generated when this happens, or use the 64bit timestamp which never wraps around. See 'caerEventPacketHeaderGetEventTSOverflow()' documentation for more details on the 64bit timestamp.

Parameters
event – a valid FrameEvent pointer. Cannot be NULL.

Returns
this event's 32bit microsecond start of exposure timestamp.

static inline int64_t **caerFrameEventGetTSStartOfExposure64**(*caerFrameEventConst* event, *caerFrameEventPacketConst* packet)

Get the 64bit start of exposure timestamp, in microseconds. See 'caerEventPacketHeaderGetEventTSOverflow()' documentation for more details on the 64bit timestamp.

Parameters

- **event** – a valid FrameEvent pointer. Cannot be NULL.
- **packet** – the FrameEventPacket pointer for the packet containing this event. Cannot be NULL.

Returns
this event's 64bit microsecond start of exposure timestamp.

static inline void **caerFrameEventSetTSStartOfExposure**(*caerFrameEvent* event, int32_t startExposure)

Set the 32bit start of exposure timestamp, the value has to be in microseconds.

Parameters

- **event** – a valid FrameEvent pointer. Cannot be NULL.
- **startExposure** – a positive 32bit microsecond timestamp.

static inline int32_t **caerFrameEventGetTSEndOfExposure**(*caerFrameEventConst* event)

Get the 32bit end of exposure timestamp, in microseconds. Be aware that this wraps around! You can either ignore this fact, or handle the special 'TIMESTAMP_WRAP' event that is generated when this happens, or use the 64bit timestamp which never wraps around. See 'caerEventPacketHeaderGetEventTSOverflow()' documentation for more details on the 64bit timestamp.

Parameters
event – a valid FrameEvent pointer. Cannot be NULL.

Returns
this event's 32bit microsecond end of exposure timestamp.

static inline int64_t **caerFrameEventGetTSEndOfExposure64**(*caerFrameEventConst* event, *caerFrameEventPacketConst* packet)

Get the 64bit end of exposure timestamp, in microseconds. See 'caerEventPacketHeaderGetEventTSOverflow()' documentation for more details on the 64bit timestamp.

Parameters

- **event** – a valid FrameEvent pointer. Cannot be NULL.
- **packet** – the FrameEventPacket pointer for the packet containing this event. Cannot be NULL.

Returns
this event's 64bit microsecond end of exposure timestamp.

static inline void **caerFrameEventSetTSEndOfExposure**(*caerFrameEvent* event, int32_t endExposure)

Set the 32bit end of exposure timestamp, the value has to be in microseconds.

Parameters

- **event** – a valid FrameEvent pointer. Cannot be NULL.
- **endExposure** – a positive 32bit microsecond timestamp.

static inline int32_t **caerFrameEventGetExposureLength**(*caerFrameEventConst* event)

The total length, in microseconds, of the frame exposure time.

Parameters
event – a valid FrameEvent pointer. Cannot be NULL.

Returns
the exposure time in microseconds.

static inline int32_t **caerFrameEventGetTimestamp**(*caerFrameEventConst* event)

Get the 32bit event timestamp, in microseconds. This is a median of the exposure timestamps. Be aware that this wraps around! You can either ignore this fact, or handle the special 'TIMESTAMP_WRAP' event that is generated when this happens, or use the 64bit timestamp which never wraps around. See 'caerEventPacketHeaderGetEventTSOverflow()' documentation for more details on the 64bit timestamp.

Parameters
event – a valid FrameEvent pointer. Cannot be NULL.

Returns
this event's 32bit microsecond timestamp.

static inline int64_t **caerFrameEventGetTimestamp64**(*caerFrameEventConst* event, *caerFrameEventPacketConst* packet)

Get the 64bit event timestamp, in microseconds. This is a median of the exposure timestamps. See 'caerEventPacketHeaderGetEventTSOverflow()' documentation for more details on the 64bit timestamp.

Parameters

- **event** – a valid FrameEvent pointer. Cannot be NULL.
- **packet** – the FrameEventPacket pointer for the packet containing this event. Cannot be NULL.

Returns
this event's 64bit microsecond timestamp.

static inline bool **caerFrameEventIsValid**(*caerFrameEventConst* event)

Check if this frame event is valid.

Parameters
event – a valid FrameEvent pointer. Cannot be NULL.

Returns
true if valid, false if not.

static inline void **caerFrameEventValidate**(*caerFrameEvent* event, *caerFrameEventPacket* packet)

Validate the current event by setting its valid bit to true and increasing the event packet's event count and valid event count. Only works on events that are invalid. DO NOT CALL THIS AFTER HAVING PREVIOUSLY ALREADY INVALIDATED THIS EVENT, the total count will be incorrect.

Parameters

- **event** – a valid FrameEvent pointer. Cannot be NULL.
- **packet** – the FrameEventPacket pointer for the packet containing this event. Cannot be NULL.

static inline void **caerFrameEventInvalidate**(*caerFrameEvent* event, *caerFrameEventPacket* packet)

Invalidate the current event by setting its valid bit to false and decreasing the number of valid events held in the packet. Only works with events that are already valid!

Parameters

- **event** – a valid FrameEvent pointer. Cannot be NULL.
- **packet** – the FrameEventPacket pointer for the packet containing this event. Cannot be NULL.

static inline size_t **caerFrameEventPacketGetPixelsSize**(*caerFrameEventPacketConst* packet)

Get the maximum size of the pixels array in bytes, based upon how much memory was allocated to it by 'caerFrameEventPacketAllocate()'.

Parameters
packet – a valid FrameEventPacket pointer. Cannot be NULL.

Returns
maximum pixels array size in bytes.

static inline size_t **caerFrameEventPacketGetPixelsMaxIndex**(*caerFrameEventPacketConst* packet)

Get the maximum index into the pixels array, based upon how much memory was allocated to it by 'caerFrameEventPacketAllocate()'.

Parameters
packet – a valid FrameEventPacket pointer. Cannot be NULL.

Returns
maximum pixels array index.

static inline uint8_t **caerFrameEventGetROIIdentifier**(*caerFrameEventConst* event)

Get the numerical identifier for the Region of Interest (ROI) region, to distinguish between multiple of them.

Parameters
event – a valid FrameEvent pointer. Cannot be NULL.

Returns
numerical ROI identifier.

static inline void **caerFrameEventSetROIIdentifier**(*caerFrameEvent* event, uint8_t roiIdentifier)

Set the numerical identifier for the Region of Interest (ROI) region, to distinguish between multiple of them.

Parameters

- **event** – a valid FrameEvent pointer. Cannot be NULL.
- **roiIdentifier** – numerical ROI identifier.

static inline enum *caer_frame_event_color_filter* **caerFrameEventGetColorFilter**(*caerFrameEventConst* event)

Get the identifier for the color filter used by the sensor. Useful for interpolating color images.

Parameters

event – a valid FrameEvent pointer. Cannot be NULL.

Returns

color filter identifier.

static inline void **caerFrameEventSetColorFilter**(*caerFrameEvent* event, enum *caer_frame_event_color_filter* colorFilter)

Set the identifier for the color filter used by the sensor. Useful for interpolating color images.

Parameters

- **event** – a valid FrameEvent pointer. Cannot be NULL.
- **colorFilter** – color filter identifier.

static inline int32_t **caerFrameEventGetLengthX**(*caerFrameEventConst* event)

Get the actual X axis length for the current frame.

Parameters

event – a valid FrameEvent pointer. Cannot be NULL.

Returns

frame X axis length.

static inline int32_t **caerFrameEventGetLengthY**(*caerFrameEventConst* event)

Get the actual Y axis length for the current frame.

Parameters

event – a valid FrameEvent pointer. Cannot be NULL.

Returns

frame Y axis length.

static inline enum *caer_frame_event_color_channels* **caerFrameEventGetChannelNumber**(*caerFrameEventConst* event)

Get the actual color channels number for the current frame. This can be used to store RGB frames for example.

Parameters

event – a valid FrameEvent pointer. Cannot be NULL.

Returns

frame color channels number.

static inline void **caerFrameEventSetLengthXLengthYChannelNumber**(*caerFrameEvent* event, int32_t lengthX, int32_t lengthY, enum *caer_frame_event_color_channels* channelNumber, *caerFrameEventPacketConst* packet)

Set the X and Y axes length and the color channels number for a frame, while taking into account the maximum amount of memory available for the pixel array, as allocated in 'caerFrameEventPacketAllocate()'.

Parameters

- **event** – a valid FrameEvent pointer. Cannot be NULL.
- **lengthX** – the frame's X axis length.
- **lengthY** – the frame's Y axis length.
- **channelNumber** – the number of color channels for this frame.
- **packet** – the FrameEventPacket pointer for the packet containing this event. Cannot be NULL.

static inline size_t **caerFrameEventGetPixelsMaxIndex**(*caerFrameEventConst* event)

Get the maximum valid index into the pixel array, at which you can still get valid pixels.

Parameters
event – a valid FrameEvent pointer. Cannot be NULL.

Returns
maximum valid pixels array index.

static inline size_t **caerFrameEventGetPixelsSize**(*caerFrameEventConst* event)

Get the maximum size of the pixels array in bytes, in which you can still get valid pixels.

Parameters
event – a valid FrameEvent pointer. Cannot be NULL.

Returns
maximum valid pixels array size in bytes.

static inline int32_t **caerFrameEventGetPositionX**(*caerFrameEventConst* event)

Get the X axis position offset. This is used to place partial frames, like the ones gotten from ROI readouts, in the visual space.

Parameters
event – a valid FrameEvent pointer. Cannot be NULL.

Returns
X axis position offset.

static inline void **caerFrameEventSetPositionX**(*caerFrameEvent* event, int32_t positionX)

Set the X axis position offset. This is used to place partial frames, like the ones gotten from ROI readouts, in the visual space.

Parameters

- **event** – a valid FrameEvent pointer. Cannot be NULL.
- **positionX** – X axis position offset.

static inline int32_t **caerFrameEventGetPositionY**(*caerFrameEventConst* event)

Get the Y axis position offset. This is used to place partial frames, like the ones gotten from ROI readouts, in the visual space.

Parameters
event – a valid FrameEvent pointer. Cannot be NULL.

Returns
Y axis position offset.

static inline void **caerFrameEventSetPositionY**(*caerFrameEvent* event, int32_t positionY)

Set the Y axis position offset. This is used to place partial frames, like the ones gotten from ROI readouts, in the visual space.

Parameters

- **event** – a valid FrameEvent pointer. Cannot be NULL.
- **positionY** – Y axis position offset.

static inline uint16_t **caerFrameEventGetPixel**(*caerFrameEventConst* event, int32_t xAddress, int32_t yAddress)

Get the pixel value at the specified (X, Y) address. (X, Y) are checked against the actual possible values for this frame. Different channels are not taken into account! The (0, 0) pixel is in the upper left corner, like in OpenCV/computer graphics.

Parameters

- **event** – a valid FrameEvent pointer. Cannot be NULL.
- **xAddress** – X address value (checked).
- **yAddress** – Y address value (checked).

Returns
pixel value (normalized to 16 bit depth).

static inline void **caerFrameEventSetPixel**(*caerFrameEvent* event, int32_t xAddress, int32_t yAddress, uint16_t pixelValue)

Set the pixel value at the specified (X, Y) address. (X, Y) are checked against the actual possible values for this frame. Different channels are not taken into account! The (0, 0) pixel is in the upper left corner, like in OpenCV/computer graphics.

Parameters

- **event** – a valid FrameEvent pointer. Cannot be NULL.
- **xAddress** – X address value (checked).
- **yAddress** – Y address value (checked).
- **pixelValue** – pixel value (normalized to 16 bit depth).

static inline uint16_t **caerFrameEventGetPixelForChannel**(*caerFrameEventConst* event, int32_t xAddress, int32_t yAddress, uint8_t channel)

Get the pixel value at the specified (X, Y) address, taking into account the specified channel. (X, Y) and the channel number are checked against the actual possible values for this frame. The (0, 0) pixel is in the upper left corner, like in OpenCV/computer graphics.

Parameters

- **event** – a valid FrameEvent pointer. Cannot be NULL.
- **xAddress** – X address value (checked).

- **yAddress** – Y address value (checked).
- **channel** – the channel number (checked).

Returns
pixel value (normalized to 16 bit depth).

static inline void **caerFrameEventSetPixelForChannel**(*caerFrameEvent* event, int32_t xAddress, int32_t yAddress, uint8_t channel, uint16_t pixelValue)

Set the pixel value at the specified (X, Y) address, taking into account the specified channel. (X, Y) and the channel number are checked against the actual possible values for this frame. The (0, 0) pixel is in the upper left corner, like in OpenCV/computer graphics.

Parameters

- **event** – a valid FrameEvent pointer. Cannot be NULL.
- **xAddress** – X address value (checked).
- **yAddress** – Y address value (checked).
- **channel** – the channel number (checked).
- **pixelValue** – pixel value (normalized to 16 bit depth).

static inline uint16_t **caerFrameEventGetPixelUnsafe**(*caerFrameEventConst* event, int32_t xAddress, int32_t yAddress)

Get the pixel value at the specified (X, Y) address. No checks on (X, Y) are performed! The (0, 0) pixel is in the upper left corner, like in OpenCV/computer graphics.

Parameters

- **event** – a valid FrameEvent pointer. Cannot be NULL.
- **xAddress** – X address value (unchecked).
- **yAddress** – Y address value (unchecked).

Returns
pixel value (normalized to 16 bit depth).

static inline void **caerFrameEventSetPixelUnsafe**(*caerFrameEvent* event, int32_t xAddress, int32_t yAddress, uint16_t pixelValue)

Set the pixel value at the specified (X, Y) address. No checks on (X, Y) are performed! The (0, 0) pixel is in the upper left corner, like in OpenCV/computer graphics.

Parameters

- **event** – a valid FrameEvent pointer. Cannot be NULL.
- **xAddress** – X address value (unchecked).
- **yAddress** – Y address value (unchecked).
- **pixelValue** – pixel value (normalized to 16 bit depth).

static inline uint16_t **caerFrameEventGetPixelForChannelUnsafe**(*caerFrameEventConst* event, int32_t xAddress, int32_t yAddress, uint8_t channel)

Get the pixel value at the specified (X, Y) address, taking into account the specified channel. No checks on (X, Y) and the channel number are performed! The (0, 0) pixel is in the upper left corner, like in OpenCV/computer graphics.

Parameters

- **event** – a valid FrameEvent pointer. Cannot be NULL.
- **xAddress** – X address value (unchecked).
- **yAddress** – Y address value (unchecked).
- **channel** – the channel number (unchecked).

Returns
pixel value (normalized to 16 bit depth).

static inline void **caerFrameEventSetPixelForChannelUnsafe**(*caerFrameEvent* event, int32_t xAddress, int32_t yAddress, uint8_t channel, uint16_t pixelValue)

Set the pixel value at the specified (X, Y) address, taking into account the specified channel. No checks on (X, Y) and the channel number are performed! The (0, 0) pixel is in the upper left corner, like in OpenCV/computer graphics.

Parameters

- **event** – a valid FrameEvent pointer. Cannot be NULL.
- **xAddress** – X address value (unchecked).
- **yAddress** – Y address value (unchecked).
- **channel** – the channel number (unchecked).
- **pixelValue** – pixel value (normalized to 16 bit depth).

static inline uint16_t ***caerFrameEventGetPixelArrayUnsafe**(*caerFrameEvent* event)

Get a direct pointer to the underlying pixels array. This can be used to both get and set values. No checks at all are performed at any point, nor any conversions, use this at your own risk! Remember that the 16 bit pixel values are in little-endian! The pixel array is laid out row by row (increasing X axis), going from top to bottom (increasing Y axis).

Parameters
event – a valid FrameEvent pointer. Cannot be NULL.

Returns
the pixels array (16 bit integers are little-endian).

static inline const uint16_t ***caerFrameEventGetPixelArrayUnsafeConst**(*caerFrameEventConst* event)

Get a direct read-only pointer to the underlying pixels array. This can be used to only get values. No checks at all are performed at any point, nor any conversions, use this at your own risk! Remember that the 16 bit pixel values are in little-endian! The pixel array is laid out row by row (increasing X axis), going from top to bottom (increasing Y axis).

Parameters
event – a valid FrameEvent pointer. Cannot be NULL.

Returns
the read-only pixels array (16 bit integers are little-endian).

file **imu6.h**

*#include "*common.h*"* IMU6 (6 axes) Events format definition and handling functions. This contains data coming from the Inertial Measurement Unit chip, with the 3-axes accelerometer and 3-axes gyroscope. Temperature is also included.

Defines

CAER_IMU6_ITERATOR_ALL_START(IMU6_PACKET)

Iterator over all IMU6 events in a packet. Returns the current index in the 'caerIMU6IteratorCounter' variable of type 'int32_t' and the current event in the 'caerIMU6IteratorElement' variable of type caerIMU6Event.

IMU6_PACKET: a valid IMU6EventPacket pointer. Cannot be NULL.

CAER_IMU6_CONST_ITERATOR_ALL_START(IMU6_PACKET)

Const-Iterator over all IMU6 events in a packet. Returns the current index in the 'caerIMU6IteratorCounter' variable of type 'int32_t' and the current read-only event in the 'caerIMU6IteratorElement' variable of type caerIMU6EventConst.

IMU6_PACKET: a valid IMU6EventPacket pointer. Cannot be NULL.

CAER_IMU6_ITERATOR_ALL_END

Iterator close statement.

CAER_IMU6_ITERATOR_VALID_START(IMU6_PACKET)

Iterator over only the valid IMU6 events in a packet. Returns the current index in the 'caerIMU6IteratorCounter' variable of type 'int32_t' and the current event in the 'caerIMU6IteratorElement' variable of type caerIMU6Event.

IMU6_PACKET: a valid IMU6EventPacket pointer. Cannot be NULL.

CAER_IMU6_CONST_ITERATOR_VALID_START(IMU6_PACKET)

Const-Iterator over only the valid IMU6 events in a packet. Returns the current index in the 'caerIMU6IteratorCounter' variable of type 'int32_t' and the current read-only event in the 'caerIMU6IteratorElement' variable of type caerIMU6EventConst.

IMU6_PACKET: a valid IMU6EventPacket pointer. Cannot be NULL.

CAER_IMU6_ITERATOR_VALID_END

Iterator close statement.

CAER_IMU6_REVERSE_ITERATOR_ALL_START(IMU6_PACKET)

Reverse iterator over all IMU6 events in a packet. Returns the current index in the 'caerIMU6IteratorCounter' variable of type 'int32_t' and the current event in the 'caerIMU6IteratorElement' variable of type caerIMU6Event.

IMU6_PACKET: a valid IMU6EventPacket pointer. Cannot be NULL.

CAER_IMU6_CONST_REVERSE_ITERATOR_ALL_START(IMU6_PACKET)

Const-Reverse iterator over all IMU6 events in a packet. Returns the current index in the 'caerIMU6IteratorCounter' variable of type 'int32_t' and the current read-only event in the 'caerIMU6IteratorElement' variable of type caerIMU6EventConst.

IMU6_PACKET: a valid IMU6EventPacket pointer. Cannot be NULL.

CAER_IMU6_REVERSE_ITERATOR_ALL_END

Reverse iterator close statement.

CAER_IMU6_REVERSE_ITERATOR_VALID_START(IMU6_PACKET)

Reverse iterator over only the valid IMU6 events in a packet. Returns the current index in the 'caerIMU6IteratorCounter' variable of type 'int32_t' and the current event in the 'caerIMU6IteratorElement' variable of type caerIMU6Event.

IMU6_PACKET: a valid IMU6EventPacket pointer. Cannot be NULL.

CAER_IMU6_CONST_REVERSE_ITERATOR_VALID_START(IMU6_PACKET)

Const-Reverse iterator over only the valid IMU6 events in a packet. Returns the current index in the 'caerIMU6IteratorCounter' variable of type 'int32_t' and the current read-only event in the 'caerIMU6IteratorElement' variable of type caerIMU6EventConst.

IMU6_PACKET: a valid IMU6EventPacket pointer. Cannot be NULL.

CAER_IMU6_REVERSE_ITERATOR_VALID_END

Reverse iterator close statement.

Typedefs

typedef struct caer_imu6_event ***caerIMU6Event**

Type for pointer to IMU 6-axes event data structure.

typedef const struct caer_imu6_event ***caerIMU6EventConst**

typedef struct caer_imu6_event_packet ***caerIMU6EventPacket**

Type for pointer to IMU 6-axes event packet data structure.

typedef const struct caer_imu6_event_packet ***caerIMU6EventPacketConst**

Functions

**PACKED_STRUCT (struct caer_imu6_event { uint32_t info;int32_t timestamp;
float accel_x;float accel_y;float accel_z;float gyro_x;float gyro_y;float gyro_z;
float temp;})**

IMU 6-axes event data structure definition. This contains accelerometer and gyroscope headings, plus temperature. The X, Y and Z axes are referred to the camera plane. X increases to the right, Y going up and Z towards where the lens is pointing. Rotation for the gyroscope is counter-clockwise along the increasing axis, for all three axes. Floats are in IEEE 754-2008 binary32 format. Signed integers are used for fields that are to be interpreted directly, for compatibility with languages that do not have unsigned integer types, such as Java.

**PACKED_STRUCT (struct caer_imu6_event_packet { struct caer_event_packet_header packetHeader;
struct caer_imu6_event events[];})**

IMU 6-axes event packet data structure definition. EventPackets are always made up of the common packet header, followed by 'eventCapacity' events. Everything has to be in one contiguous memory block.

static inline *caerIMU6EventPacket* **caerIMU6EventPacketAllocate**(int32_t eventCapacity, int16_t eventSource, int32_t tsOverflow)

Allocate a new IMU 6-axes events packet. Use free() to reclaim this memory.

Parameters

- **eventCapacity** – the maximum number of events this packet will hold.
- **eventSource** – the unique ID representing the source/generator of this packet.

- **tsOverflow** – the current timestamp overflow counter value for this packet.

Returns
a valid IMU6EventPacket handle or NULL on error.

static inline *caerIMU6EventPacket* **caerIMU6EventPacketFromPacketHeader**(*caerEventPacketHeader* header)

Transform a generic event packet header into an IMU 6-axes event packet. This takes care of proper casting and checks that the packet type really matches the intended conversion type.

Parameters
header – a valid event packet header pointer. Cannot be NULL.

Returns
a properly converted, typed event packet pointer.

static inline *caerIMU6EventPacketConst* **caerIMU6EventPacketFromPacketHeaderConst**(*caerEventPacketHeaderConst* header)

Transform a generic read-only event packet header into a read-only IMU 6-axes event packet. This takes care of proper casting and checks that the packet type really matches the intended conversion type.

Parameters
header – a valid read-only event packet header pointer. Cannot be NULL.

Returns
a properly converted, read-only typed event packet pointer.

static inline *caerIMU6Event* **caerIMU6EventPacketGetEvent**(*caerIMU6EventPacket* packet, int32_t n)

Get the IMU 6-axes event at the given index from the event packet.

Parameters

- **packet** – a valid IMU6EventPacket pointer. Cannot be NULL.
- **n** – the index of the returned event. Must be within [0,eventCapacity[bounds.

Returns
the requested IMU 6-axes event. NULL on error.

static inline *caerIMU6EventConst* **caerIMU6EventPacketGetEventConst**(*caerIMU6EventPacketConst* packet, int32_t n)

Get the IMU 6-axes event at the given index from the event packet. This is a read-only event, do not change its contents in any way!

Parameters

- **packet** – a valid IMU6EventPacket pointer. Cannot be NULL.
- **n** – the index of the returned event. Must be within [0,eventCapacity[bounds.

Returns
the requested read-only IMU 6-axes event. NULL on error.

static inline int32_t **caerIMU6EventGetTimestamp**(*caerIMU6EventConst* event)

Get the 32bit event timestamp, in microseconds. Be aware that this wraps around! You can either ignore this fact, or handle the special 'TIMESTAMP_WRAP' event that is generated when this happens, or use the 64bit timestamp which never wraps around. See 'caerEventPacketHeaderGetEventTSOverflow()' documentation for more details on the 64bit timestamp.

Parameters
event – a valid IMU6Event pointer. Cannot be NULL.

Returns
this event's 32bit microsecond timestamp.

static inline int64_t **caerIMU6EventGetTimestamp64**(*caerIMU6EventConst* event, *caerIMU6EventPacketConst* packet)

Get the 64bit event timestamp, in microseconds. See 'caerEventPacketHeaderGetEventTSOverflow()' documentation for more details on the 64bit timestamp.

Parameters

- **event** – a valid IMU6Event pointer. Cannot be NULL.
- **packet** – the IMU6EventPacket pointer for the packet containing this event. Cannot be NULL.

Returns
this event's 64bit microsecond timestamp.

static inline void **caerIMU6EventSetTimestamp**(*caerIMU6Event* event, int32_t timestamp)

Set the 32bit event timestamp, the value has to be in microseconds.

Parameters

- **event** – a valid IMU6Event pointer. Cannot be NULL.
- **timestamp** – a positive 32bit microsecond timestamp.

static inline bool **caerIMU6EventIsValid**(*caerIMU6EventConst* event)

Check if this IMU 6-axes event is valid.

Parameters
event – a valid IMU6Event pointer. Cannot be NULL.

Returns
true if valid, false if not.

static inline void **caerIMU6EventValidate**(*caerIMU6Event* event, *caerIMU6EventPacket* packet)

Validate the current event by setting its valid bit to true and increasing the event packet's event count and valid event count. Only works on events that are invalid. DO NOT CALL THIS AFTER HAVING PREVIOUSLY ALREADY INVALIDATED THIS EVENT, the total count will be incorrect.

Parameters

- **event** – a valid IMU6Event pointer. Cannot be NULL.
- **packet** – the IMU6EventPacket pointer for the packet containing this event. Cannot be NULL.

static inline void **caerIMU6EventInvalidate**(*caerIMU6Event* event, *caerIMU6EventPacket* packet)

Invalidate the current event by setting its valid bit to false and decreasing the number of valid events held in the packet. Only works with events that are already valid!

Parameters

- **event** – a valid IMU6Event pointer. Cannot be NULL.
- **packet** – the IMU6EventPacket pointer for the packet containing this event. Cannot be NULL.

static inline float **caerIMU6EventGetAccelX**(*caerIMU6EventConst* event)

Get the X axis acceleration reading (from accelerometer). This is in g (1 g = 9.81 m/s^2).

Parameters
event – a valid IMU6Event pointer. Cannot be NULL.

Returns
acceleration on the X axis.

static inline void **caerIMU6EventSetAccelX**(*caerIMU6Event* event, float accelX)

Set the X axis acceleration reading (from accelerometer). This is in g (1 g = 9.81 m/s^2).

Parameters

- **event** – a valid IMU6Event pointer. Cannot be NULL.
- **accelX** – acceleration on the X axis.

static inline float **caerIMU6EventGetAccelY**(*caerIMU6EventConst* event)

Get the Y axis acceleration reading (from accelerometer). This is in g (1 g = 9.81 m/s^2).

Parameters
event – a valid IMU6Event pointer. Cannot be NULL.

Returns
acceleration on the Y axis.

static inline void **caerIMU6EventSetAccelY**(*caerIMU6Event* event, float accelY)

Set the Y axis acceleration reading (from accelerometer). This is in g (1 g = 9.81 m/s^2).

Parameters

- **event** – a valid IMU6Event pointer. Cannot be NULL.
- **accelY** – acceleration on the Y axis.

static inline float **caerIMU6EventGetAccelZ**(*caerIMU6EventConst* event)

Get the Z axis acceleration reading (from accelerometer). This is in g (1 g = 9.81 m/s^2).

Parameters
event – a valid IMU6Event pointer. Cannot be NULL.

Returns
acceleration on the Z axis.

static inline void **caerIMU6EventSetAccelZ**(*caerIMU6Event* event, float accelZ)

Set the Z axis acceleration reading (from accelerometer). This is in g (1 g = 9.81 m/s^2).

Parameters

- **event** – a valid IMU6Event pointer. Cannot be NULL.
- **accelZ** – acceleration on the Z axis.

static inline float **caerIMU6EventGetGyroX**(*caerIMU6EventConst* event)

Get the X axis (roll) angular velocity reading (from gyroscope). This is in °/s (deg/sec).

Parameters
event – a valid IMU6Event pointer. Cannot be NULL.

Returns
angular velocity on the X axis (roll).

static inline void **caerIMU6EventSetGyroX**(*caerIMU6Event* event, float gyroX)

Set the X axis (roll) angular velocity reading (from gyroscope). This is in °/s (deg/sec).

Parameters

- **event** – a valid IMU6Event pointer. Cannot be NULL.
- **gyroX** – angular velocity on the X axis (roll).

static inline float **caerIMU6EventGetGyroY**(*caerIMU6EventConst* event)

Get the Y axis (pitch) angular velocity reading (from gyroscope). This is in °/s (deg/sec).

Parameters
event – a valid IMU6Event pointer. Cannot be NULL.

Returns
angular velocity on the Y axis (pitch).

static inline void **caerIMU6EventSetGyroY**(*caerIMU6Event* event, float gyroY)

Set the Y axis (pitch) angular velocity reading (from gyroscope). This is in °/s (deg/sec).

Parameters

- **event** – a valid IMU6Event pointer. Cannot be NULL.
- **gyroY** – angular velocity on the Y axis (pitch).

static inline float **caerIMU6EventGetGyroZ**(*caerIMU6EventConst* event)

Get the Z axis (yaw) angular velocity reading (from gyroscope). This is in °/s (deg/sec).

Parameters
event – a valid IMU6Event pointer. Cannot be NULL.

Returns
angular velocity on the Z axis (yaw).

static inline void **caerIMU6EventSetGyroZ**(*caerIMU6Event* event, float gyroZ)

Set the Z axis (yaw) angular velocity reading (from gyroscope). This is in °/s (deg/sec).

Parameters

- **event** – a valid IMU6Event pointer. Cannot be NULL.
- **gyroZ** – angular velocity on the Z axis (yaw).

static inline float **caerIMU6EventGetTemp**(*caerIMU6EventConst* event)

Get the temperature reading. This is in °C.

Parameters
event – a valid IMU6Event pointer. Cannot be NULL.

Returns
temperature in °C.

static inline void **caerIMU6EventSetTemp**(*caerIMU6Event* event, float temp)

Set the temperature reading. This is in °C.

Parameters

- **event** – a valid IMU6Event pointer. Cannot be NULL.
- **temp** – temperature in °C.

file **imu9.h**

*#include "*common.h*"* IMU9 (9 axes) Events format definition and handling functions. This contains data coming from the Inertial Measurement Unit chip, with the 3-axes accelerometer and 3-axes gyroscope. Temperature is also included. Further, 3-axes from the magnetometer are included, which can be used to get a compass-like heading.

Defines

CAER_IMU9_ITERATOR_ALL_START(IMU9_PACKET)

Iterator over all IMU9 events in a packet. Returns the current index in the 'caerIMU9IteratorCounter' variable of type 'int32_t' and the current event in the 'caerIMU9IteratorElement' variable of type caerIMU9Event.

IMU9_PACKET: a valid IMU9EventPacket pointer. Cannot be NULL.

CAER_IMU9_CONST_ITERATOR_ALL_START(IMU9_PACKET)

Const-Iterator over all IMU9 events in a packet. Returns the current index in the 'caerIMU9IteratorCounter' variable of type 'int32_t' and the current read-only event in the 'caerIMU9IteratorElement' variable of type caerIMU9EventConst.

IMU9_PACKET: a valid IMU9EventPacket pointer. Cannot be NULL.

CAER_IMU9_ITERATOR_ALL_END

Iterator close statement.

CAER_IMU9_ITERATOR_VALID_START(IMU9_PACKET)

Iterator over only the valid IMU9 events in a packet. Returns the current index in the 'caerIMU9IteratorCounter' variable of type 'int32_t' and the current event in the 'caerIMU9IteratorElement' variable of type caerIMU9Event.

IMU9_PACKET: a valid IMU9EventPacket pointer. Cannot be NULL.

CAER_IMU9_CONST_ITERATOR_VALID_START(IMU9_PACKET)

Const-Iterator over only the valid IMU9 events in a packet. Returns the current index in the 'caerIMU9IteratorCounter' variable of type 'int32_t' and the current read-only event in the 'caerIMU9IteratorElement' variable of type caerIMU9EventConst.

IMU9_PACKET: a valid IMU9EventPacket pointer. Cannot be NULL.

CAER_IMU9_ITERATOR_VALID_END

Iterator close statement.

CAER_IMU9_REVERSE_ITERATOR_ALL_START(IMU9_PACKET)

Reverse iterator over all IMU9 events in a packet. Returns the current index in the 'caerIMU9IteratorCounter' variable of type 'int32_t' and the current event in the 'caerIMU9IteratorElement' variable of type caerIMU9Event.

IMU9_PACKET: a valid IMU9EventPacket pointer. Cannot be NULL.

CAER_IMU9_CONST_REVERSE_ITERATOR_ALL_START(IMU9_PACKET)

Const-Reverse iterator over all IMU9 events in a packet. Returns the current index in the 'caerIMU9IteratorCounter' variable of type 'int32_t' and the current read-only event in the 'caerIMU9IteratorElement' variable of type caerIMU9EventConst.

IMU9_PACKET: a valid IMU9EventPacket pointer. Cannot be NULL.

CAER_IMU9_REVERSE_ITERATOR_ALL_END

Reverse iterator close statement.

CAER_IMU9_REVERSE_ITERATOR_VALID_START(IMU9_PACKET)

Reverse iterator over only the valid IMU9 events in a packet. Returns the current index in the 'caerIMU9IteratorCounter' variable of type 'int32_t' and the current event in the 'caerIMU9IteratorElement' variable of type caerIMU9Event.

IMU9_PACKET: a valid IMU9EventPacket pointer. Cannot be NULL.

CAER_IMU9_CONST_REVERSE_ITERATOR_VALID_START(IMU9_PACKET)

Const-Reverse iterator over only the valid IMU9 events in a packet. Returns the current index in the 'caerIMU9IteratorCounter' variable of type 'int32_t' and the current read-only event in the 'caerIMU9IteratorElement' variable of type caerIMU9EventConst.

IMU9_PACKET: a valid IMU9EventPacket pointer. Cannot be NULL.

CAER_IMU9_REVERSE_ITERATOR_VALID_END

Reverse iterator close statement.

Typedefs

typedef struct caer_imu9_event ***caerIMU9Event**

Type for pointer to IMU 9-axes event data structure.

typedef const struct caer_imu9_event ***caerIMU9EventConst**

typedef struct caer_imu9_event_packet ***caerIMU9EventPacket**

Type for pointer to IMU 9-axes event packet data structure.

typedef const struct caer_imu9_event_packet ***caerIMU9EventPacketConst**

Functions

**PACKED_STRUCT (struct caer_imu9_event { uint32_t info;int32_t timestamp;
float accel_x;float accel_y;float accel_z;float gyro_x;float gyro_y;float gyro_z;
float temp;float comp_x;float comp_y;float comp_z;})**

IMU 9-axes event data structure definition. This contains accelerometer and gyroscope headings, plus temperature, and magnetometer readings. The X, Y and Z axes are referred to the camera plane. X increases to the right, Y going up and Z towards where the lens is pointing. Rotation for the gyroscope is counter-clockwise along the increasing axis, for all three axes. Floats are in IEEE 754-2008 binary32 format. Signed integers are used for fields that are to be interpreted directly, for compatibility with languages that do not have unsigned integer types, such as Java.

**PACKED_STRUCT (struct caer_imu9_event_packet { struct caer_event_packet_header packetHeader;
struct caer_imu9_event events[];})**

IMU 9-axes event packet data structure definition. EventPackets are always made up of the common packet header, followed by 'eventCapacity' events. Everything has to be in one contiguous memory block.

static inline *caerIMU9EventPacket* **caerIMU9EventPacketAllocate**(int32_t eventCapacity, int16_t eventSource, int32_t tsOverflow)

Allocate a new IMU 9-axes events packet. Use free() to reclaim this memory.

Parameters

- **eventCapacity** – the maximum number of events this packet will hold.
- **eventSource** – the unique ID representing the source/generator of this packet.

- **tsOverflow** – the current timestamp overflow counter value for this packet.

Returns
a valid IMU9EventPacket handle or NULL on error.

static inline *caerIMU9EventPacket* **caerIMU9EventPacketFromPacketHeader**(*caerEventPacketHeader* header)

Transform a generic event packet header into an IMU 9-axes event packet. This takes care of proper casting and checks that the packet type really matches the intended conversion type.

Parameters
header – a valid event packet header pointer. Cannot be NULL.

Returns
a properly converted, typed event packet pointer.

static inline *caerIMU9EventPacketConst* **caerIMU9EventPacketFromPacketHeaderConst**(*caerEventPacketHeaderConst* header)

Transform a generic read-only event packet header into a read-only IMU 9-axes event packet. This takes care of proper casting and checks that the packet type really matches the intended conversion type.

Parameters
header – a valid read-only event packet header pointer. Cannot be NULL.

Returns
a properly converted, read-only typed event packet pointer.

static inline *caerIMU9Event* **caerIMU9EventPacketGetEvent**(*caerIMU9EventPacket* packet, int32_t n)

Get the IMU 9-axes event at the given index from the event packet.

Parameters

- **packet** – a valid IMU9EventPacket pointer. Cannot be NULL.
- **n** – the index of the returned event. Must be within [0,eventCapacity[bounds.

Returns
the requested IMU 9-axes event. NULL on error.

static inline *caerIMU9EventConst* **caerIMU9EventPacketGetEventConst**(*caerIMU9EventPacketConst* packet, int32_t n)

Get the IMU 9-axes event at the given index from the event packet. This is a read-only event, do not change its contents in any way!

Parameters

- **packet** – a valid IMU9EventPacket pointer. Cannot be NULL.
- **n** – the index of the returned event. Must be within [0,eventCapacity[bounds.

Returns
the requested read-only IMU 9-axes event. NULL on error.

static inline int32_t **caerIMU9EventGetTimestamp**(*caerIMU9EventConst* event)

Get the 32bit event timestamp, in microseconds. Be aware that this wraps around! You can either ignore this fact, or handle the special 'TIMESTAMP_WRAP' event that is generated when this happens, or use the 64bit timestamp which never wraps around. See 'caerEventPacketHeaderGetEventTSOverflow()' documentation for more details on the 64bit timestamp.

Parameters
event – a valid IMU9Event pointer. Cannot be NULL.

Returns
this event's 32bit microsecond timestamp.

static inline int64_t **caerIMU9EventGetTimestamp64**(*caerIMU9EventConst* event, *caerIMU9EventPacketConst* packet)

Get the 64bit event timestamp, in microseconds. See 'caerEventPacketHeaderGetEventTSOverflow()' documentation for more details on the 64bit timestamp.

Parameters

- **event** – a valid IMU9Event pointer. Cannot be NULL.
- **packet** – the IMU9EventPacket pointer for the packet containing this event. Cannot be NULL.

Returns
this event's 64bit microsecond timestamp.

static inline void **caerIMU9EventSetTimestamp**(*caerIMU9Event* event, int32_t timestamp)

Set the 32bit event timestamp, the value has to be in microseconds.

Parameters

- **event** – a valid IMU9Event pointer. Cannot be NULL.
- **timestamp** – a positive 32bit microsecond timestamp.

static inline bool **caerIMU9EventIsValid**(*caerIMU9EventConst* event)

Check if this IMU 9-axes event is valid.

Parameters
event – a valid IMU9Event pointer. Cannot be NULL.

Returns
true if valid, false if not.

static inline void **caerIMU9EventValidate**(*caerIMU9Event* event, *caerIMU9EventPacket* packet)

Validate the current event by setting its valid bit to true and increasing the event packet's event count and valid event count. Only works on events that are invalid. DO NOT CALL THIS AFTER HAVING PREVIOUSLY ALREADY INVALIDATED THIS EVENT, the total count will be incorrect.

Parameters

- **event** – a valid IMU9Event pointer. Cannot be NULL.
- **packet** – the IMU9EventPacket pointer for the packet containing this event. Cannot be NULL.

static inline void **caerIMU9EventInvalidate**(*caerIMU9Event* event, *caerIMU9EventPacket* packet)

Invalidate the current event by setting its valid bit to false and decreasing the number of valid events held in the packet. Only works with events that are already valid!

Parameters

- **event** – a valid IMU9Event pointer. Cannot be NULL.
- **packet** – the IMU9EventPacket pointer for the packet containing this event. Cannot be NULL.

static inline float **caerIMU9EventGetAccelX**(*caerIMU9EventConst* event)

Get the X axis acceleration reading (from accelerometer). This is in g (1 g = 9.81 m/s^2).

Parameters
event – a valid IMU9Event pointer. Cannot be NULL.

Returns
acceleration on the X axis.

static inline void **caerIMU9EventSetAccelX**(*caerIMU9Event* event, float accelX)

Set the X axis acceleration reading (from accelerometer). This is in g (1 g = 9.81 m/s^2).

Parameters

- **event** – a valid IMU9Event pointer. Cannot be NULL.
- **accelX** – acceleration on the X axis.

static inline float **caerIMU9EventGetAccelY**(*caerIMU9EventConst* event)

Get the Y axis acceleration reading (from accelerometer). This is in g (1 g = 9.81 m/s^2).

Parameters
event – a valid IMU9Event pointer. Cannot be NULL.

Returns
acceleration on the Y axis.

static inline void **caerIMU9EventSetAccelY**(*caerIMU9Event* event, float accelY)

Set the Y axis acceleration reading (from accelerometer). This is in g (1 g = 9.81 m/s^2).

Parameters

- **event** – a valid IMU9Event pointer. Cannot be NULL.
- **accelY** – acceleration on the Y axis.

static inline float **caerIMU9EventGetAccelZ**(*caerIMU9EventConst* event)

Get the Z axis acceleration reading (from accelerometer). This is in g (1 g = 9.81 m/s^2).

Parameters
event – a valid IMU9Event pointer. Cannot be NULL.

Returns
acceleration on the Z axis.

static inline void **caerIMU9EventSetAccelZ**(*caerIMU9Event* event, float accelZ)

Set the Z axis acceleration reading (from accelerometer). This is in g (1 g = 9.81 m/s^2).

Parameters

- **event** – a valid IMU9Event pointer. Cannot be NULL.
- **accelZ** – acceleration on the Z axis.

static inline float **caerIMU9EventGetGyroX**(*caerIMU9EventConst* event)

Get the X axis (roll) angular velocity reading (from gyroscope). This is in °/s (deg/sec).

Parameters
event – a valid IMU9Event pointer. Cannot be NULL.

Returns
angular velocity on the X axis (roll).

static inline void **caerIMU9EventSetGyroX**(*caerIMU9Event* event, float gyroX)

Set the X axis (roll) angular velocity reading (from gyroscope). This is in °/s (deg/sec).

Parameters

- **event** – a valid IMU9Event pointer. Cannot be NULL.
- **gyroX** – angular velocity on the X axis (roll).

static inline float **caerIMU9EventGetGyroY**(*caerIMU9EventConst* event)

Get the Y axis (pitch) angular velocity reading (from gyroscope). This is in °/s (deg/sec).

Parameters
event – a valid IMU9Event pointer. Cannot be NULL.

Returns
angular velocity on the Y axis (pitch).

static inline void **caerIMU9EventSetGyroY**(*caerIMU9Event* event, float gyroY)

Set the Y axis (pitch) angular velocity reading (from gyroscope). This is in °/s (deg/sec).

Parameters

- **event** – a valid IMU9Event pointer. Cannot be NULL.
- **gyroY** – angular velocity on the Y axis (pitch).

static inline float **caerIMU9EventGetGyroZ**(*caerIMU9EventConst* event)

Get the Z axis (yaw) angular velocity reading (from gyroscope). This is in °/s (deg/sec).

Parameters
event – a valid IMU9Event pointer. Cannot be NULL.

Returns
angular velocity on the Z axis (yaw).

static inline void **caerIMU9EventSetGyroZ**(*caerIMU9Event* event, float gyroZ)

Set the Z axis (yaw) angular velocity reading (from gyroscope). This is in °/s (deg/sec).

Parameters

- **event** – a valid IMU9Event pointer. Cannot be NULL.
- **gyroZ** – angular velocity on the Z axis (yaw).

static inline float **caerIMU9EventGetTemp**(*caerIMU9EventConst* event)

Get the temperature reading. This is in °C.

Parameters
event – a valid IMU9Event pointer. Cannot be NULL.

Returns
temperature in °C.

static inline void **caerIMU9EventSetTemp**(*caerIMU9Event* event, float temp)

Set the temperature reading. This is in °C.

Parameters

- **event** – a valid IMU9Event pointer. Cannot be NULL.
- **temp** – temperature in °C.

static inline float **caerIMU9EventGetCompX**(*caerIMU9EventConst* event)

Get the X axis compass heading (from magnetometer). This is in µT.

Parameters
event – a valid IMU9Event pointer. Cannot be NULL.

Returns
X axis compass heading.

static inline void **caerIMU9EventSetCompX**(*caerIMU9Event* event, float compX)

Set the X axis compass heading (from magnetometer). This is in µT.

Parameters

- **event** – a valid IMU9Event pointer. Cannot be NULL.
- **compX** – X axis compass heading.

static inline float **caerIMU9EventGetCompY**(*caerIMU9EventConst* event)

Get the Y axis compass heading (from magnetometer). This is in µT.

Parameters
event – a valid IMU9Event pointer. Cannot be NULL.

Returns
Y axis compass heading.

static inline void **caerIMU9EventSetCompY**(*caerIMU9Event* event, float compY)

Set the Y axis compass heading (from magnetometer). This is in µT.

Parameters

- **event** – a valid IMU9Event pointer. Cannot be NULL.
- **compY** – Y axis compass heading.

static inline float **caerIMU9EventGetCompZ**(*caerIMU9EventConst* event)

Get the Z axis compass heading (from magnetometer). This is in µT.

Parameters
event – a valid IMU9Event pointer. Cannot be NULL.

Returns
Z axis compass heading.

static inline void **caerIMU9EventSetCompZ**(*caerIMU9Event* event, float compZ)

Set the Z axis compass heading (from magnetometer). This is in µT.

Parameters

- **event** – a valid IMU9Event pointer. Cannot be NULL.
- **compZ** – Z axis compass heading.

file **packetContainer.h**

*#include "*common.h*"* EventPacketContainer format definition and handling functions. An EventPacketContainer is a logical construct that contains packets of events (EventPackets) of different event types, with the aim of keeping related events of differing types, such as DVS and IMU data, together. Such a relation is usually based on time intervals, trying to keep groups of event happening in a certain time-slice together. This time-order is based on the *main* time-stamp of an event, the one whose offset is referenced in the event packet header and that is used by the caerGenericEvent*() functions. It's guaranteed that all conforming input modules keep to this rule, generating containers that include all events from all types within the given time-slice. The smallest and largest timestamps are tracked at the packet container level as a convenience, to avoid having to examine all packets for this often useful piece of information. All integers are in their native host format, as this is a purely internal, in-memory data structure, never meant for exchange between different systems (and different endianness).

== Packet Containers and Input Modules == The "packeting system" works in this way: events are accumulated by type in a packet, and that packet is part of a packet container, by an input module. The packet container is then sent out for processing when either the configured time limit or the size limit are hit. The time limit is always active, in microseconds, and basically tells you the time-span an event packet covers. This enables regular,

constant delivery of packets, that cover a period of time. The size limit is an addon to prevent packets to grow to immense sizes (like if the time limit is high and there is lots of activity). As soon as a packet hits the number of events in the size limit, it is sent out. The regular time limit is not reset in this case. This size limit can be disabled by setting it to 0. The cAER DVS128/DAVIS/File/Network input modules call these two configuration variables "PacketContainerInterval" and "PacketContainerMaxPacketSize". Too small packet sizes or intervals simply mean more packets, which may negatively affect performance. It's usually a good idea to set the size to something around 4-8K, and the time to a good value based on the application you're building, so if you need ms-reaction-time, you probably want to set it to 1000µs, so that you do get new data every ms. If on the other hand you're looking at a static scene and just want to detect that something is passing by once every while, a higher number like 100ms might also be perfectly appropriate.

Defines

CAER_EVENT_PACKET_CONTAINER_ITERATOR_START(PACKET_CONTAINER)

Iterator over all event packets in an event packet container. Returns the current index in the 'caerEventPacketContainerIteratorCounter' variable of type 'int32_t' and the current event packet in the 'caerEventPacketContainerIteratorElement' variable of type caerEventPacketHeader. The current packet may be NULL, in which case it is skipped during iteration.

PACKET_CONTAINER: a valid EventPacketContainer handle. If NULL, no iteration is performed.

CAER_EVENT_PACKET_CONTAINER_CONST_ITERATOR_START(PACKET_CONTAINER)

Const-Iterator over all event packets in an event packet container. Returns the current index in the 'caerEventPacketContainerIteratorCounter' variable of type 'int32_t' and the current read-only event packet in the 'caerEventPacketContainerIteratorElement' variable of type caerEventPacketHeaderConst. The current packet may be NULL, in which case it is skipped during iteration.

PACKET_CONTAINER: a valid EventPacketContainer handle. If NULL, no iteration is performed.

CAER_EVENT_PACKET_CONTAINER_ITERATOR_END

Iterator close statement.

Typedefs

typedef struct caer_event_packet_container ***caerEventPacketContainer**

Type for pointer to EventPacketContainer data structure.

typedef const struct caer_event_packet_container ***caerEventPacketContainerConst**

Functions

PACKED_STRUCT (struct caer_event_packet_container { int64_t lowestEventTimestamp; int64_t highestEventTimestamp;int32_t eventsNumber;int32_t eventsValidNumber; int32_t eventPacketsNumber;caerEventPacketHeader eventPackets[];})

EventPacketContainer data structure definition. Signed integers are used for compatibility with languages that do not have unsigned ones, such as Java.

static inline *caerEventPacketContainer* **caerEventPacketContainerAllocate**(int32_t eventPacketsNumber)

Allocate a new EventPacketContainer with enough space to store up to the given number of EventPacket pointers. All packet pointers will be NULL initially.

Parameters
eventPacketsNumber – the maximum number of EventPacket pointers that can be stored in this container.

Returns
a valid EventPacketContainer handle or NULL on error.

static inline void **caerEventPacketContainerUpdateStatistics**(*caerEventPacketContainer* container)

Recalculates and updates all the packet-container level statistics (event counts and timestamps).

Parameters
container – a valid EventPacketContainer handle. If NULL, nothing happens.

static inline int32_t **caerEventPacketContainerGetEventPacketsNumber**(*caerEventPacketContainerConst* container)

Get the maximum number of EventPacket pointers that can be stored in this particular EventPacketContainer.

Parameters
container – a valid EventPacketContainer handle. If NULL, zero is returned.

Returns
the number of EventPacket pointers that can be contained.

static inline void **caerEventPacketContainerSetEventPacketsNumber**(*caerEventPacketContainer* container, int32_t eventPacketsNumber)

Set the maximum number of EventPacket pointers that can be stored in this particular EventPacketContainer. This should never be used directly, caerEventPacketContainerAllocate() sets this for you.

Parameters

- **container** – a valid EventPacketContainer handle. If NULL, nothing happens.
- **eventPacketsNumber** – the number of EventPacket pointers that can be contained.

static inline *caerEventPacketHeader* **caerEventPacketContainerGetEventPacket**(*caerEventPacketContainerConst* container, int32_t n)

Get the pointer to the EventPacket stored in this container at the given index.

Parameters

- **container** – a valid EventPacketContainer handle. If NULL, returns NULL too.
- **n** – the index of the EventPacket to get.

Returns
a pointer to an EventPacket or NULL on error.

static inline *caerEventPacketHeaderConst* **caerEventPacketContainerGetEventPacketConst**(*caerEventPacketContainerC* container, int32_t n)

Get the pointer to the EventPacket stored in this container at the given index. This is a read-only EventPacket, do not change its contents in any way!

Parameters

- **container** – a valid EventPacketContainer handle. If NULL, returns NULL too.

- **n** – the index of the EventPacket to get.

Returns
a pointer to a read-only EventPacket or NULL on error.

static inline void **caerEventPacketContainerSetEventPacket**(*caerEventPacketContainer* container, int32_t n, *caerEventPacketHeader* packetHeader)

Set the pointer to the EventPacket stored in this container at the given index.

Parameters

- **container** – a valid EventPacketContainer handle. If NULL, nothing happens.
- **n** – the index of the EventPacket to set.
- **packetHeader** – a pointer to an EventPacket's header. Can be NULL.

static inline void **caerEventPacketContainerFree**(*caerEventPacketContainer* container)

Free the memory occupied by an EventPacketContainer, as well as freeing all of its contained EventPackets and their memory. If you don't want the contained EventPackets to be freed, make sure that you set their pointers to NULL before calling this.

Parameters
container – the container to be freed.

static inline int64_t **caerEventPacketContainerGetLowestEventTimestamp**(*caerEventPacketContainerConst* container)

Get the lowest timestamp contained in this event packet container.

Parameters
container – a valid EventPacketContainer handle. If NULL, -1 is returned.

Returns
the lowest timestamp (in µs) or -1 if not initialized.

static inline int64_t **caerEventPacketContainerGetHighestEventTimestamp**(*caerEventPacketContainerConst* container)

Get the highest timestamp contained in this event packet container.

Parameters
container – a valid EventPacketContainer handle. If NULL, -1 is returned.

Returns
the highest timestamp (in µs) or -1 if not initialized.

static inline int32_t **caerEventPacketContainerGetEventsNumber**(*caerEventPacketContainerConst* container)

Get the number of events contained in this event packet container.

Parameters
container – a valid EventPacketContainer handle. If NULL, 0 is returned.

Returns
the number of events in this container.

static inline int32_t **caerEventPacketContainerGetEventsValidNumber**(*caerEventPacketContainerConst* container)

Get the number of valid events contained in this event packet container.

Parameters
container – a valid EventPacketContainer handle. If NULL, 0 is returned.

Returns
the number of valid events in this container.

static inline *caerEventPacketHeader* **caerEventPacketContainerFindEventPacketByType**(*caerEventPacketContainerCons* container, int16_t typeID)

Get the pointer to an EventPacket stored in this container with the given event type. This returns the first found event packet with that type ID, or NULL if we get to the end without finding any such event packet.

Parameters

- **container** – a valid EventPacketContainer handle. If NULL, returns NULL too.
- **typeID** – the event type to search for.

Returns
a pointer to an EventPacket with a certain type or NULL if none found.

static inline *caerEventPacketHeaderConst* **caerEventPacketContainerFindEventPacketByTypeConst**(*caerEventPacketCo* container, int16_t typeID)

Get the pointer to a read-only EventPacket stored in this container with the given event type. This returns the first found event packet with that type ID, or NULL if we get to the end without finding any such event packet.

Parameters

- **container** – a valid EventPacketContainer handle. If NULL, returns NULL too.
- **typeID** – the event type to search for.

Returns
a pointer to a read-only EventPacket with a certain type or NULL if none found.

static inline *caerEventPacketContainer* **caerEventPacketContainerCopyAllEvents**(*caerEventPacketContainerConst* container)

Make a deep copy of an event packet container and all of its event packets and their current events.

Parameters
container – an event packet container to copy.

Returns
a deep copy of an event packet container, containing all events.

static inline *caerEventPacketContainer* **caerEventPacketContainerCopyValidEvents**(*caerEventPacketContainerConst* container)

Make a deep copy of an event packet container, with its event packets sized down to only include the currently valid events (eventValid), and discarding everything else.

Parameters
container – an event packet container to copy.

Returns
a deep copy of an event packet container, containing only valid events.

file **polarity.h**

*#include "*common.h*"* Polarity Events format definition and handling functions. This event contains change

information, with an X/Y address and an ON/OFF polarity. The (0, 0) address is in the upper left corner of the screen, like in OpenCV/computer graphics.

Defines

POLARITY_SHIFT

Shift and mask values for the polarity, X and Y addresses of a polarity event. Addresses up to 15 bit are supported. Polarity is ON(=1) or OFF(=0). Bit 0 is the valid mark, see '*common.h*' for more details.

POLARITY_MASK

POLARITY_Y_ADDR_SHIFT

POLARITY_Y_ADDR_MASK

POLARITY_X_ADDR_SHIFT

POLARITY_X_ADDR_MASK

CAER_POLARITY_ITERATOR_ALL_START(POLARITY_PACKET)

Iterator over all polarity events in a packet. Returns the current index in the 'caerPolarityIteratorCounter' variable of type 'int32_t' and the current event in the 'caerPolarityIteratorElement' variable of type caerPolarityEvent.

POLARITY_PACKET: a valid PolarityEventPacket pointer. Cannot be NULL.

CAER_POLARITY_CONST_ITERATOR_ALL_START(POLARITY_PACKET)

Const-Iterator over all polarity events in a packet. Returns the current index in the 'caerPolarityIteratorCounter' variable of type 'int32_t' and the current read-only event in the 'caerPolarityIteratorElement' variable of type caerPolarityEventConst.

POLARITY_PACKET: a valid PolarityEventPacket pointer. Cannot be NULL.

CAER_POLARITY_ITERATOR_ALL_END

Iterator close statement.

CAER_POLARITY_ITERATOR_VALID_START(POLARITY_PACKET)

Iterator over only the valid polarity events in a packet. Returns the current index in the 'caerPolarityIteratorCounter' variable of type 'int32_t' and the current event in the 'caerPolarityIteratorElement' variable of type caerPolarityEvent.

POLARITY_PACKET: a valid PolarityEventPacket pointer. Cannot be NULL.

CAER_POLARITY_CONST_ITERATOR_VALID_START(POLARITY_PACKET)

Const-Iterator over only the valid polarity events in a packet. Returns the current index in the 'caerPolarityIteratorCounter' variable of type 'int32_t' and the current read-only event in the 'caerPolarityIteratorElement' variable of type caerPolarityEventConst.

POLARITY_PACKET: a valid PolarityEventPacket pointer. Cannot be NULL.

CAER_POLARITY_ITERATOR_VALID_END

Iterator close statement.

CAER_POLARITY_REVERSE_ITERATOR_ALL_START(POLARITY_PACKET)

Reverse iterator over all polarity events in a packet. Returns the current index in the 'caerPolarityIteratorCounter' variable of type 'int32_t' and the current event in the 'caerPolarityIteratorElement' variable of type caerPolarityEvent.

POLARITY_PACKET: a valid PolarityEventPacket pointer. Cannot be NULL.

CAER_POLARITY_CONST_REVERSE_ITERATOR_ALL_START(POLARITY_PACKET)

Const-Reverse iterator over all polarity events in a packet. Returns the current index in the 'caerPolarityIteratorCounter' variable of type 'int32_t' and the current read-only event in the 'caerPolarityIteratorElement' variable of type caerPolarityEventConst.

POLARITY_PACKET: a valid PolarityEventPacket pointer. Cannot be NULL.

CAER_POLARITY_REVERSE_ITERATOR_ALL_END

Reverse iterator close statement.

CAER_POLARITY_REVERSE_ITERATOR_VALID_START(POLARITY_PACKET)

Reverse iterator over only the valid polarity events in a packet. Returns the current index in the 'caerPolarityIteratorCounter' variable of type 'int32_t' and the current event in the 'caerPolarityIteratorElement' variable of type caerPolarityEvent.

POLARITY_PACKET: a valid PolarityEventPacket pointer. Cannot be NULL.

CAER_POLARITY_CONST_REVERSE_ITERATOR_VALID_START(POLARITY_PACKET)

Const-Reverse iterator over only the valid polarity events in a packet. Returns the current index in the 'caerPolarityIteratorCounter' variable of type 'int32_t' and the current read-only event in the 'caerPolarityIteratorElement' variable of type caerPolarityEventConst.

POLARITY_PACKET: a valid PolarityEventPacket pointer. Cannot be NULL.

CAER_POLARITY_REVERSE_ITERATOR_VALID_END

Reverse iterator close statement.

Typedefs

typedef struct caer_polarity_event ***caerPolarityEvent**

Type for pointer to polarity event data structure.

typedef const struct caer_polarity_event ***caerPolarityEventConst**

typedef struct caer_polarity_event_packet ***caerPolarityEventPacket**

Type for pointer to polarity event packet data structure.

typedef const struct caer_polarity_event_packet ***caerPolarityEventPacketConst**

Functions

PACKED_STRUCT (struct caer_polarity_event { uint32_t data;int32_t timestamp;})

Polarity event data structure definition. This contains the actual X/Y addresses, the polarity, as well as the 32 bit event timestamp. The (0, 0) address is in the upper left corner of the screen, like in OpenCV/computer graphics. Signed integers are used for fields that are to be interpreted directly, for compatibility with languages that do not have unsigned integer types, such as Java.

PACKED_STRUCT (struct caer_polarity_event_packet { struct caer_event_packet_header packetHeader; struct caer_polarity_event events[];})

Polarity event packet data structure definition. EventPackets are always made up of the common packet header, followed by 'eventCapacity' events. Everything has to be in one contiguous memory block.

static inline *caerPolarityEventPacket* **caerPolarityEventPacketAllocate**(int32_t eventCapacity, int16_t eventSource, int32_t tsOverflow)

Allocate a new polarity events packet. Use free() to reclaim this memory.

Parameters

- **eventCapacity** – the maximum number of events this packet will hold.
- **eventSource** – the unique ID representing the source/generator of this packet.
- **tsOverflow** – the current timestamp overflow counter value for this packet.

Returns
a valid PolarityEventPacket handle or NULL on error.

static inline *caerPolarityEventPacket* **caerPolarityEventPacketFromPacketHeader**(*caerEventPacketHeader* header)

Transform a generic event packet header into a Polarity event packet. This takes care of proper casting and checks that the packet type really matches the intended conversion type.

Parameters
header – a valid event packet header pointer. Cannot be NULL.

Returns
a properly converted, typed event packet pointer.

static inline *caerPolarityEventPacketConst* **caerPolarityEventPacketFromPacketHeaderConst**(*caerEventPacketHeaderCo* header)

Transform a generic read-only event packet header into a read-only Polarity event packet. This takes care of proper casting and checks that the packet type really matches the intended conversion type.

Parameters
header – a valid read-only event packet header pointer. Cannot be NULL.

Returns
a properly converted, read-only typed event packet pointer.

static inline *caerPolarityEvent* **caerPolarityEventPacketGetEvent**(*caerPolarityEventPacket* packet, int32_t n)

Get the polarity event at the given index from the event packet.

Parameters

- **packet** – a valid PolarityEventPacket pointer. Cannot be NULL.

- **n** – the index of the returned event. Must be within [0,eventCapacity[bounds.

Returns
the requested polarity event. NULL on error.

static inline *caerPolarityEventConst* **caerPolarityEventPacketGetEventConst**(*caerPolarityEventPacketConst* packet, int32_t n)

Get the polarity event at the given index from the event packet. This is a read-only event, do not change its contents in any way!

Parameters

- **packet** – a valid PolarityEventPacket pointer. Cannot be NULL.
- **n** – the index of the returned event. Must be within [0,eventCapacity[bounds.

Returns
the requested read-only polarity event. NULL on error.

static inline int32_t **caerPolarityEventGetTimestamp**(*caerPolarityEventConst* event)

Get the 32bit event timestamp, in microseconds. Be aware that this wraps around! You can either ignore this fact, or handle the special 'TIMESTAMP_WRAP' event that is generated when this happens, or use the 64bit timestamp which never wraps around. See 'caerEventPacketHeaderGetEventTSOverflow()' documentation for more details on the 64bit timestamp.

Parameters
event – a valid PolarityEvent pointer. Cannot be NULL.

Returns
this event's 32bit microsecond timestamp.

static inline int64_t **caerPolarityEventGetTimestamp64**(*caerPolarityEventConst* event, *caerPolarityEventPacketConst* packet)

Get the 64bit event timestamp, in microseconds. See 'caerEventPacketHeaderGetEventTSOverflow()' documentation for more details on the 64bit timestamp.

Parameters

- **event** – a valid PolarityEvent pointer. Cannot be NULL.
- **packet** – the PolarityEventPacket pointer for the packet containing this event. Cannot be NULL.

Returns
this event's 64bit microsecond timestamp.

static inline void **caerPolarityEventSetTimestamp**(*caerPolarityEvent* event, int32_t timestamp)

Set the 32bit event timestamp, the value has to be in microseconds.

Parameters

- **event** – a valid PolarityEvent pointer. Cannot be NULL.
- **timestamp** – a positive 32bit microsecond timestamp.

static inline bool **caerPolarityEventIsValid**(*caerPolarityEventConst* event)

Check if this polarity event is valid.

Parameters
event – a valid PolarityEvent pointer. Cannot be NULL.

Returns
true if valid, false if not.

static inline void **caerPolarityEventValidate**(*caerPolarityEvent* event, *caerPolarityEventPacket* packet)

Validate the current event by setting its valid bit to true and increasing the event packet's event count and valid event count. Only works on events that are invalid. DO NOT CALL THIS AFTER HAVING PREVIOUSLY ALREADY INVALIDATED THIS EVENT, the total count will be incorrect.

Parameters

- **event** – a valid PolarityEvent pointer. Cannot be NULL.
- **packet** – the PolarityEventPacket pointer for the packet containing this event. Cannot be NULL.

static inline void **caerPolarityEventInvalidate**(*caerPolarityEvent* event, *caerPolarityEventPacket* packet)

Invalidate the current event by setting its valid bit to false and decreasing the number of valid events held in the packet. Only works with events that are already valid!

Parameters

- **event** – a valid PolarityEvent pointer. Cannot be NULL.
- **packet** – the PolarityEventPacket pointer for the packet containing this event. Cannot be NULL.

static inline bool **caerPolarityEventGetPolarity**(*caerPolarityEventConst* event)

Get the change event polarity. 1 is ON, 0 is OFF.

Parameters
event – a valid PolarityEvent pointer. Cannot be NULL.

Returns
event polarity value.

static inline void **caerPolarityEventSetPolarity**(*caerPolarityEvent* event, bool polarity)

Set the change event polarity. 1 is ON, 0 is OFF.

Parameters

- **event** – a valid PolarityEvent pointer. Cannot be NULL.
- **polarity** – event polarity value.

static inline uint16_t **caerPolarityEventGetY**(*caerPolarityEventConst* event)

Get the Y (row) address for a change event, in pixels. The (0, 0) address is in the upper left corner, like in OpenCV/computer graphics.

Parameters
event – a valid PolarityEvent pointer. Cannot be NULL.

Returns
the event Y address.

static inline void **caerPolarityEventSetY**(*caerPolarityEvent* event, uint16_t yAddress)

Set the Y (row) address for a change event, in pixels. The (0, 0) address is in the upper left corner, like in OpenCV/computer graphics.

Parameters

- **event** – a valid PolarityEvent pointer. Cannot be NULL.
- **yAddress** – the event Y address.

static inline uint16_t **caerPolarityEventGetX**(*caerPolarityEventConst* event)

Get the X (column) address for a change event, in pixels. The (0, 0) address is in the upper left corner, like in OpenCV/computer graphics.

Parameters
event – a valid PolarityEvent pointer. Cannot be NULL.

Returns
the event X address.

static inline void **caerPolarityEventSetX**(*caerPolarityEvent* event, uint16_t xAddress)

Set the X (column) address for a change event, in pixels. The (0, 0) address is in the upper left corner, like in OpenCV/computer graphics.

Parameters

- **event** – a valid PolarityEvent pointer. Cannot be NULL.
- **xAddress** – the event X address.

file **special.h**

*#include "*common.h*"* Special Events format definition and handling functions. This event type encodes special occurrences, such as timestamp related notifications or external input events.

Defines

SPECIAL_TYPE_SHIFT

Shift and mask values for the type and data portions of a special event. Up to 128 types, with 24 bits of data each, are possible. Bit 0 is the valid mark, see '*common.h*' for more details.

SPECIAL_TYPE_MASK

SPECIAL_DATA_SHIFT

SPECIAL_DATA_MASK

CAER_SPECIAL_ITERATOR_ALL_START(SPECIAL_PACKET)

Iterator over all special events in a packet. Returns the current index in the 'caerSpecialIteratorCounter' variable of type 'int32_t' and the current event in the 'caerSpecialIteratorElement' variable of type caerSpecialEvent.

SPECIAL_PACKET: a valid SpecialEventPacket pointer. Cannot be NULL.

CAER_SPECIAL_CONST_ITERATOR_ALL_START(SPECIAL_PACKET)

Const-Iterator over all special events in a packet. Returns the current index in the 'caerSpecialIteratorCounter' variable of type 'int32_t' and the current read-only event in the 'caerSpecialIteratorElement' variable of type caerSpecialEventConst.

SPECIAL_PACKET: a valid SpecialEventPacket pointer. Cannot be NULL.

CAER_SPECIAL_ITERATOR_ALL_END

Iterator close statement.

CAER_SPECIAL_ITERATOR_VALID_START(SPECIAL_PACKET)

Iterator over only the valid special events in a packet. Returns the current index in the 'caerSpecialIteratorCounter' variable of type 'int32_t' and the current event in the 'caerSpecialIteratorElement' variable of type caerSpecialEvent.

SPECIAL_PACKET: a valid SpecialEventPacket pointer. Cannot be NULL.

CAER_SPECIAL_CONST_ITERATOR_VALID_START(SPECIAL_PACKET)

Const-Iterator over only the valid special events in a packet. Returns the current index in the 'caerSpecialIteratorCounter' variable of type 'int32_t' and the current read-only event in the 'caerSpecialIteratorElement' variable of type caerSpecialEventConst.

SPECIAL_PACKET: a valid SpecialEventPacket pointer. Cannot be NULL.

CAER_SPECIAL_ITERATOR_VALID_END

Iterator close statement.

CAER_SPECIAL_REVERSE_ITERATOR_ALL_START(SPECIAL_PACKET)

Reverse iterator over all special events in a packet. Returns the current index in the 'caerSpecialIteratorCounter' variable of type 'int32_t' and the current event in the 'caerSpecialIteratorElement' variable of type caerSpecialEvent.

SPECIAL_PACKET: a valid SpecialEventPacket pointer. Cannot be NULL.

CAER_SPECIAL_CONST_REVERSE_ITERATOR_ALL_START(SPECIAL_PACKET)

Const-Reverse iterator over all special events in a packet. Returns the current index in the 'caerSpecialIteratorCounter' variable of type 'int32_t' and the current read-only event in the 'caerSpecialIteratorElement' variable of type caerSpecialEventConst.

SPECIAL_PACKET: a valid SpecialEventPacket pointer. Cannot be NULL.

CAER_SPECIAL_REVERSE_ITERATOR_ALL_END

Reverse iterator close statement.

CAER_SPECIAL_REVERSE_ITERATOR_VALID_START(SPECIAL_PACKET)

Reverse iterator over only the valid special events in a packet. Returns the current index in the 'caerSpecialIteratorCounter' variable of type 'int32_t' and the current event in the 'caerSpecialIteratorElement' variable of type caerSpecialEvent.

SPECIAL_PACKET: a valid SpecialEventPacket pointer. Cannot be NULL.

CAER_SPECIAL_CONST_REVERSE_ITERATOR_VALID_START(SPECIAL_PACKET)

Const-Reverse iterator over only the valid special events in a packet. Returns the current index in the 'caerSpecialIteratorCounter' variable of type 'int32_t' and the current read-only event in the 'caerSpecialIteratorElement' variable of type caerSpecialEventConst.

SPECIAL_PACKET: a valid SpecialEventPacket pointer. Cannot be NULL.

CAER_SPECIAL_REVERSE_ITERATOR_VALID_END

Reverse iterator close statement.

Typedefs

typedef struct caer_special_event *caerSpecialEvent

Type for pointer to special event data structure.

typedef const struct caer_special_event *caerSpecialEventConst

typedef struct caer_special_event_packet *caerSpecialEventPacket

Type for pointer to special event packet data structure.

typedef const struct caer_special_event_packet *caerSpecialEventPacketConst

Enums

enum caer_special_event_types

List of all special event type identifiers. Used to interpret the special event type field.

Values:

enumerator TIMESTAMP_WRAP

A 32 bit timestamp wrap occurred.

enumerator TIMESTAMP_RESET

A timestamp reset occurred.

enumerator EXTERNAL_INPUT_RISING_EDGE

A rising edge was detected (External Input module on device).

enumerator EXTERNAL_INPUT_FALLING_EDGE

A falling edge was detected (External Input module on device).

enumerator EXTERNAL_INPUT_PULSE

A pulse was detected (External Input module on device).

enumerator DVS_ROW_ONLY

A DVS row-only event was detected (a row address without any following column addresses).

enumerator EXTERNAL_INPUT1_RISING_EDGE

A rising edge was detected (External Input 1 module on device).

enumerator EXTERNAL_INPUT1_FALLING_EDGE

A falling edge was detected (External Input 1 module on device).

enumerator EXTERNAL_INPUT1_PULSE

A pulse was detected (External Input 1 module on device).

enumerator **EXTERNAL_INPUT2_RISING_EDGE**

A rising edge was detected (External Input 2 module on device).

enumerator **EXTERNAL_INPUT2_FALLING_EDGE**

A falling edge was detected (External Input 2 module on device).

enumerator **EXTERNAL_INPUT2_PULSE**

A pulse was detected (External Input 2 module on device).

enumerator **EXTERNAL_GENERATOR_RISING_EDGE**

A rising edge was generated (External Input Generator module on device).

enumerator **EXTERNAL_GENERATOR_FALLING_EDGE**

A falling edge was generated (External Input Generator module on device).

enumerator **APS_FRAME_START**

An APS frame capture has started (Frame Event will follow).

enumerator **APS_FRAME_END**

An APS frame capture has completed (Frame Event is alongside).

enumerator **APS_EXPOSURE_START**

An APS frame exposure has started (Frame Event will follow).

enumerator **APS_EXPOSURE_END**

An APS frame exposure has completed (Frame Event will follow).

enumerator **EVENT_READOUT_START**

Start of a new event readout from synchronous scanning sensors.

Functions

PACKED_STRUCT (struct caer_special_event { uint32_t data;int32_t timestamp;})

Special event data structure definition. This contains the actual data, as well as the 32 bit event timestamp. Signed integers are used for fields that are to be interpreted directly, for compatibility with languages that do not have unsigned integer types, such as Java.

PACKED_STRUCT (struct caer_special_event_packet { struct caer_event_packet_header packetHeader; struct caer_special_event events[];})

Special event packet data structure definition. EventPackets are always made up of the common packet header, followed by 'eventCapacity' events. Everything has to be in one contiguous memory block.

static inline *caerSpecialEventPacket* **caerSpecialEventPacketAllocate**(int32_t eventCapacity, int16_t eventSource, int32_t tsOverflow)

Allocate a new special events packet. Use free() to reclaim this memory.

Parameters

- **eventCapacity** – the maximum number of events this packet will hold.
- **eventSource** – the unique ID representing the source/generator of this packet.
- **tsOverflow** – the current timestamp overflow counter value for this packet.

Returns
a valid SpecialEventPacket handle or NULL on error.

static inline *caerSpecialEventPacket* **caerSpecialEventPacketFromPacketHeader**(*caerEventPacketHeader* header)

Transform a generic event packet header into a Special event packet. This takes care of proper casting and checks that the packet type really matches the intended conversion type.

Parameters
header – a valid event packet header pointer. Cannot be NULL.

Returns
a properly converted, typed event packet pointer.

static inline *caerSpecialEventPacketConst* **caerSpecialEventPacketFromPacketHeaderConst**(*caerEventPacketHeaderCons* header)

Transform a generic read-only event packet header into a read-only Special event packet. This takes care of proper casting and checks that the packet type really matches the intended conversion type.

Parameters
header – a valid read-only event packet header pointer. Cannot be NULL.

Returns
a properly converted, read-only typed event packet pointer.

static inline *caerSpecialEvent* **caerSpecialEventPacketGetEvent**(*caerSpecialEventPacket* packet, int32_t n)

Get the special event at the given index from the event packet.

Parameters

- **packet** – a valid SpecialEventPacket pointer. Cannot be NULL.
- **n** – the index of the returned event. Must be within [0,eventCapacity[bounds.

Returns
the requested special event. NULL on error.

static inline *caerSpecialEventConst* **caerSpecialEventPacketGetEventConst**(*caerSpecialEventPacketConst* packet, int32_t n)

Get the special event at the given index from the event packet. This is a read-only event, do not change its contents in any way!

Parameters

- **packet** – a valid SpecialEventPacket pointer. Cannot be NULL.
- **n** – the index of the returned event. Must be within [0,eventCapacity[bounds.

Returns
the requested read-only special event. NULL on error.

static inline int32_t **caerSpecialEventGetTimestamp**(*caerSpecialEventConst* event)

Get the 32bit event timestamp, in microseconds. Be aware that this wraps around! You can either ignore this fact, or handle the special 'TIMESTAMP_WRAP' event that is generated when this happens, or use the 64bit timestamp which never wraps around. See 'caerEventPacketHeaderGetEventTSOverflow()' documentation for more details on the 64bit timestamp.

Parameters
event – a valid SpecialEvent pointer. Cannot be NULL.

Returns
this event's 32bit microsecond timestamp.

static inline int64_t **caerSpecialEventGetTimestamp64**(*caerSpecialEventConst* event, *caerSpecialEventPacketConst* packet)

Get the 64bit event timestamp, in microseconds. See 'caerEventPacketHeaderGetEventTSOverflow()' documentation for more details on the 64bit timestamp.

Parameters

- **event** – a valid SpecialEvent pointer. Cannot be NULL.
- **packet** – the SpecialEventPacket pointer for the packet containing this event. Cannot be NULL.

Returns
this event's 64bit microsecond timestamp.

static inline void **caerSpecialEventSetTimestamp**(*caerSpecialEvent* event, int32_t timestamp)

Set the 32bit event timestamp, the value has to be in microseconds.

Parameters

- **event** – a valid SpecialEvent pointer. Cannot be NULL.
- **timestamp** – a positive 32bit microsecond timestamp.

static inline bool **caerSpecialEventIsValid**(*caerSpecialEventConst* event)

Check if this special event is valid.

Parameters
event – a valid SpecialEvent pointer. Cannot be NULL.

Returns
true if valid, false if not.

static inline void **caerSpecialEventValidate**(*caerSpecialEvent* event, *caerSpecialEventPacket* packet)

Validate the current event by setting its valid bit to true and increasing the event packet's event count and valid event count. Only works on events that are invalid. DO NOT CALL THIS AFTER HAVING PREVIOUSLY ALREADY INVALIDATED THIS EVENT, the total count will be incorrect.

Parameters

- **event** – a valid SpecialEvent pointer. Cannot be NULL.
- **packet** – the SpecialEventPacket pointer for the packet containing this event. Cannot be NULL.

static inline void **caerSpecialEventInvalidate**(*caerSpecialEvent* event, *caerSpecialEventPacket* packet)

Invalidate the current event by setting its valid bit to false and decreasing the number of valid events held in the packet. Only works with events that are already valid!

Parameters

- **event** – a valid SpecialEvent pointer. Cannot be NULL.
- **packet** – the SpecialEventPacket pointer for the packet containing this event. Cannot be NULL.

static inline uint8_t **caerSpecialEventGetType**(*caerSpecialEventConst* event)

Get the numerical special event type.

Parameters
event – a valid SpecialEvent pointer. Cannot be NULL.

Returns
the special event type (see 'enum caer_special_event_types').

static inline void **caerSpecialEventSetType**(*caerSpecialEvent* event, uint8_t type)

Set the numerical special event type.

Parameters

- **event** – a valid SpecialEvent pointer. Cannot be NULL.
- **type** – the special event type (see 'enum caer_special_event_types').

static inline uint32_t **caerSpecialEventGetData**(*caerSpecialEventConst* event)

Get the special event data. Its meaning depends on the type. Current types that make use of it are (see 'enum caer_special_event_types'):

- DVS_ROW_ONLY: encodes the address of the row from the row-only event.

Parameters
event – a valid SpecialEvent pointer. Cannot be NULL.

Returns
the special event data.

static inline void **caerSpecialEventSetData**(*caerSpecialEvent* event, uint32_t data)

Set the special event data. Its meaning depends on the type. Current types that make use of it are (see 'enum caer_special_event_types'):

- DVS_ROW_ONLY: encodes the address of the row from the row-only event.

Parameters

- **event** – a valid SpecialEvent pointer. Cannot be NULL.
- **data** – the special event data.

static inline *caerSpecialEvent* **caerSpecialEventPacketFindEventByType**(*caerSpecialEventPacket* packet, uint8_t type)

Get the first special event with the given event type in this event packet. This returns the first found event with that type ID, or NULL if we get to the end without finding any such event.

Parameters

- **packet** – a valid SpecialEventPacket pointer. Cannot be NULL.
- **type** – the special event type to search for.

Returns
the requested special event or NULL on error/not found.

static inline *caerSpecialEventConst* **caerSpecialEventPacketFindEventByTypeConst**(*caerSpecialEventPacketConst* packet, uint8_t type)

Get the first special event with the given event type in this event packet. This returns the first found event with that type ID, or NULL if we get to the end without finding any such event. The returned event is read-only!

Parameters

- **packet** – a valid SpecialEventPacket pointer. Cannot be NULL.
- **type** – the special event type to search for.

Returns
the requested read-only special event or NULL on error/not found.

static inline *caerSpecialEvent* **caerSpecialEventPacketFindValidEventByType**(*caerSpecialEventPacket* packet, uint8_t type)

Get the first valid special event with the given event type in this event packet. This returns the first found valid event with that type ID, or NULL if we get to the end without finding any such event.

Parameters

- **packet** – a valid SpecialEventPacket pointer. Cannot be NULL.
- **type** – the special event type to search for.

Returns
the requested valid special event or NULL on error/not found.

static inline *caerSpecialEventConst* **caerSpecialEventPacketFindValidEventByTypeConst**(*caerSpecialEventPacketConst* packet, uint8_t type)

Get the first valid special event with the given event type in this event packet. This returns the first found valid event with that type ID, or NULL if we get to the end without finding any such event. The returned event is read-only!

Parameters

- **packet** – a valid SpecialEventPacket pointer. Cannot be NULL.
- **type** – the special event type to search for.

Returns
the requested read-only valid special event or NULL on error/not found.

file **spike.h**

#include "common.h" Spike Events format definition and handling functions. This contains spikes generated by a neuron-array chip.

Defines

SPIKE_SOURCE_CORE_ID_SHIFT

Shift and mask values for spike information associated with a Spike event. 32 core IDs, 64 chip IDs and up to a million neuron IDs are supported. Bit 0 is the valid mark, see '*common.h*' for more details.

SPIKE_SOURCE_CORE_ID_MASK

SPIKE_CHIP_ID_SHIFT

SPIKE_CHIP_ID_MASK

SPIKE_NEURON_ID_SHIFT

SPIKE_NEURON_ID_MASK

CAER_SPIKE_ITERATOR_ALL_START(SPIKE_PACKET)

Iterator over all Spike events in a packet. Returns the current index in the 'caerSpikeIteratorCounter' variable of type 'int32_t' and the current event in the 'caerSpikeIteratorElement' variable of type caerSpikeEvent.

SPIKE_PACKET: a valid SpikeEventPacket pointer. Cannot be NULL.

CAER_SPIKE_CONST_ITERATOR_ALL_START(SPIKE_PACKET)

Const-Iterator over all Spike events in a packet. Returns the current index in the 'caerSpikeIteratorCounter' variable of type 'int32_t' and the current read-only event in the 'caerSpikeIteratorElement' variable of type caerSpikeEventConst.

SPIKE_PACKET: a valid SpikeEventPacket pointer. Cannot be NULL.

CAER_SPIKE_ITERATOR_ALL_END

Iterator close statement.

CAER_SPIKE_ITERATOR_VALID_START(SPIKE_PACKET)

Iterator over only the valid Spike events in a packet. Returns the current index in the 'caerSpikeIteratorCounter' variable of type 'int32_t' and the current event in the 'caerSpikeIteratorElement' variable of type caerSpikeEvent.

SPIKE_PACKET: a valid SpikeEventPacket pointer. Cannot be NULL.

CAER_SPIKE_CONST_ITERATOR_VALID_START(SPIKE_PACKET)

Const-Iterator over only the valid Spike events in a packet. Returns the current index in the 'caerSpikeIteratorCounter' variable of type 'int32_t' and the current read-only event in the 'caerSpikeIteratorElement' variable of type caerSpikeEventConst.

SPIKE_PACKET: a valid SpikeEventPacket pointer. Cannot be NULL.

CAER_SPIKE_ITERATOR_VALID_END

Iterator close statement.

CAER_SPIKE_REVERSE_ITERATOR_ALL_START(SPIKE_PACKET)

Reverse iterator over all spike events in a packet. Returns the current index in the 'caerSpikeIteratorCounter' variable of type 'int32_t' and the current event in the 'caerSpikeIteratorElement' variable of type caerSpikeEvent.

SPIKE_PACKET: a valid SpikeEventPacket pointer. Cannot be NULL.

CAER_SPIKE_CONST_REVERSE_ITERATOR_ALL_START(SPIKE_PACKET)

Const-Reverse iterator over all spike events in a packet. Returns the current index in the 'caerSpikeIteratorCounter' variable of type 'int32_t' and the current read-only event in the 'caerSpikeIteratorElement' variable of type caerSpikeEventConst.

SPIKE_PACKET: a valid SpikeEventPacket pointer. Cannot be NULL.

CAER_SPIKE_REVERSE_ITERATOR_ALL_END

Reverse iterator close statement.

CAER_SPIKE_REVERSE_ITERATOR_VALID_START(SPIKE_PACKET)

Reverse iterator over only the valid spike events in a packet. Returns the current index in the 'caerSpikeIteratorCounter' variable of type 'int32_t' and the current event in the 'caerSpikeIteratorElement' variable of type caerSpikeEvent.

SPIKE_PACKET: a valid SpikeEventPacket pointer. Cannot be NULL.

CAER_SPIKE_CONST_REVERSE_ITERATOR_VALID_START(SPIKE_PACKET)

Const-Reverse iterator over only the valid spike events in a packet. Returns the current index in the 'caerSpikeIteratorCounter' variable of type 'int32_t' and the current read-only event in the 'caerSpikeIteratorElement' variable of type caerSpikeEventConst.

SPIKE_PACKET: a valid SpikeEventPacket pointer. Cannot be NULL.

CAER_SPIKE_REVERSE_ITERATOR_VALID_END

Reverse iterator close statement.

Typedefs

typedef struct caer_spike_event ***caerSpikeEvent**

Type for pointer to Spike event data structure.

typedef const struct caer_spike_event ***caerSpikeEventConst**

typedef struct caer_spike_event_packet ***caerSpikeEventPacket**

Type for pointer to Spike event packet data structure.

typedef const struct caer_spike_event_packet ***caerSpikeEventPacketConst**

Functions

PACKED_STRUCT (struct caer_spike_event { uint32_t data;int32_t timestamp;})

Spike event data structure definition. This contains the core ID, the neuron ID and the timestamp of the received spike, together with the usual validity mark. Signed integers are used for fields that are to be interpreted directly, for compatibility with languages that do not have unsigned integer types, such as Java.

PACKED_STRUCT (struct caer_spike_event_packet { struct caer_event_packet_header packetHeader; struct caer_spike_event events[];})

Spike event packet data structure definition. EventPackets are always made up of the common packet header, followed by 'eventCapacity' events. Everything has to be in one contiguous memory block.

static inline *caerSpikeEventPacket* **caerSpikeEventPacketAllocate**(int32_t eventCapacity, int16_t eventSource, int32_t tsOverflow)

Allocate a new Spike events packet. Use free() to reclaim this memory.

Parameters

- **eventCapacity** – the maximum number of events this packet will hold.
- **eventSource** – the unique ID representing the source/generator of this packet.
- **tsOverflow** – the current timestamp overflow counter value for this packet.

Returns
a valid SpikeEventPacket handle or NULL on error.

static inline *caerSpikeEventPacket* **caerSpikeEventPacketFromPacketHeader**(*caerEventPacketHeader* header)

Transform a generic event packet header into a Spike event packet. This takes care of proper casting and checks that the packet type really matches the intended conversion type.

Parameters
header – a valid event packet header pointer. Cannot be NULL.

Returns
a properly converted, typed event packet pointer.

static inline *caerSpikeEventPacketConst* **caerSpikeEventPacketFromPacketHeaderConst**(*caerEventPacketHeaderConst* header)

Transform a generic read-only event packet header into a read-only Spike event packet. This takes care of proper casting and checks that the packet type really matches the intended conversion type.

Parameters
header – a valid read-only event packet header pointer. Cannot be NULL.

Returns
a properly converted, read-only typed event packet pointer.

static inline *caerSpikeEvent* **caerSpikeEventPacketGetEvent**(*caerSpikeEventPacket* packet, int32_t n)

Get the Spike event at the given index from the event packet.

Parameters

- **packet** – a valid SpikeEventPacket pointer. Cannot be NULL.
- **n** – the index of the returned event. Must be within [0,eventCapacity[bounds.

Returns
the requested Spike event. NULL on error.

static inline *caerSpikeEventConst* **caerSpikeEventPacketGetEventConst**(*caerSpikeEventPacketConst* packet, int32_t n)

Get the Spike event at the given index from the event packet. This is a read-only event, do not change its contents in any way!

Parameters

- **packet** – a valid SpikeEventPacket pointer. Cannot be NULL.
- **n** – the index of the returned event. Must be within [0,eventCapacity[bounds.

Returns
the requested read-only Spike event. NULL on error.

static inline int32_t **caerSpikeEventGetTimestamp**(*caerSpikeEventConst* event)

Get the 32bit event timestamp, in microseconds. Be aware that this wraps around! You can either ignore this fact, or handle the special 'TIMESTAMP_WRAP' event that is generated when this happens, or use the 64bit timestamp which never wraps around. See 'caerEventPacketHeaderGetEventTSOverflow()' documentation for more details on the 64bit timestamp.

Parameters
event – a valid SpikeEvent pointer. Cannot be NULL.

Returns
this event's 32bit microsecond timestamp.

static inline int64_t **caerSpikeEventGetTimestamp64**(*caerSpikeEventConst* event, *caerSpikeEventPacketConst* packet)

Get the 64bit event timestamp, in microseconds. See 'caerEventPacketHeaderGetEventTSOverflow()' documentation for more details on the 64bit timestamp.

Parameters

- **event** – a valid SpikeEvent pointer. Cannot be NULL.
- **packet** – the SpikeEventPacket pointer for the packet containing this event. Cannot be NULL.

Returns
this event's 64bit microsecond timestamp.

static inline void **caerSpikeEventSetTimestamp**(*caerSpikeEvent* event, int32_t timestamp)

Set the 32bit event timestamp, the value has to be in microseconds.

Parameters

- **event** – a valid SpikeEvent pointer. Cannot be NULL.
- **timestamp** – a positive 32bit microsecond timestamp.

static inline bool **caerSpikeEventIsValid**(*caerSpikeEventConst* event)

Check if this Spike event is valid.

Parameters
event – a valid SpikeEvent pointer. Cannot be NULL.

Returns
true if valid, false if not.

static inline void **caerSpikeEventValidate**(*caerSpikeEvent* event, *caerSpikeEventPacket* packet)

Validate the current event by setting its valid bit to true and increasing the event packet's event count and valid event count. Only works on events that are invalid. DO NOT CALL THIS AFTER HAVING PREVIOUSLY ALREADY INVALIDATED THIS EVENT, the total count will be incorrect.

Parameters

- **event** – a valid SpikeEvent pointer. Cannot be NULL.
- **packet** – the SpikeEventPacket pointer for the packet containing this event. Cannot be NULL.

static inline void **caerSpikeEventInvalidate**(*caerSpikeEvent* event, *caerSpikeEventPacket* packet)

Invalidate the current event by setting its valid bit to false and decreasing the number of valid events held in the packet. Only works with events that are already valid!

Parameters

- **event** – a valid SpikeEvent pointer. Cannot be NULL.
- **packet** – the SpikeEventPacket pointer for the packet containing this event. Cannot be NULL.

static inline uint8_t **caerSpikeEventGetSourceCoreID**(*caerSpikeEventConst* event)

Get the source core ID.

Parameters
event – a valid SpikeEvent pointer. Cannot be NULL.

Returns
the Spike's source core ID.

static inline void **caerSpikeEventSetSourceCoreID**(*caerSpikeEvent* event, uint8_t sourceCoreID)

Set the source core ID.

Parameters

- **event** – a valid SpikeEvent pointer. Cannot be NULL.
- **sourceCoreID** – the Spike's source core ID.

static inline uint8_t **caerSpikeEventGetChipID**(*caerSpikeEventConst* event)

Get the chip ID.

Parameters
event – a valid SpikeEvent pointer. Cannot be NULL.

Returns
the Spike's chip ID.

static inline void **caerSpikeEventSetChipID**(*caerSpikeEvent* event, uint8_t chipID)

Set the chip ID.

Parameters

- **event** – a valid SpikeEvent pointer. Cannot be NULL.
- **chipID** – the Spike's chip ID.

static inline uint32_t **caerSpikeEventGetNeuronID**(*caerSpikeEventConst* event)

Get the neuron ID.

Parameters
event – a valid SpikeEvent pointer. Cannot be NULL.

Returns
the Spike's neuron ID.

static inline void **caerSpikeEventSetNeuronID**(*caerSpikeEvent* event, uint32_t neuronID)

Set the neuron ID.

Parameters

- **event** – a valid SpikeEvent pointer. Cannot be NULL.
- **neuronID** – the Spike's neuron ID.

file **dvs_noise.h**

#include "../events/polarity.h" The DVS noise filter combines a HotPixel filter (high activity pixels), a Background-Activity filter (uncorrelated events), and a Refractory Period filter (limit event rate of a pixel). The HotPixel and Background-Activity filters reduce noise due to transistor mismatch, the Refractory Period filter can reduce the event rate and is efficient to implement together with the Background-Activity filter, requiring only one pixel memory map for both. Please note that the filter is not thread-safe, all function calls should happen on the same thread, unless you take care that they never overlap.

Defines

CAER_FILTER_DVS_HOTPIXEL_LEARN

DVS HotPixel Filter: Turn on learning to determine which pixels are hot, meaning abnormally active within a certain time period. In the absence of external stimuli, the only pixels behaving as such must be noise. Once learning is enabled, do not disable it until completed. To verify completion, query this parameter and wait for it to switch from 'true' back to 'false'.

CAER_FILTER_DVS_HOTPIXEL_TIME

DVS HotPixel Filter: Minimum time (in µs) to accumulate events for during learning.

CAER_FILTER_DVS_HOTPIXEL_COUNT

DVS HotPixel Filter: Minimum number of events, during the given learning time, for a pixel to be considered hot.

CAER_FILTER_DVS_HOTPIXEL_ENABLE

DVS HotPixel Filter: Enable the hot pixel filter, filtering out the last learned hot pixels.

CAER_FILTER_DVS_HOTPIXEL_STATISTICS

DVS HotPixel Filter: Number of events filtered out by the hot pixel filter.

CAER_FILTER_DVS_HOTPIXEL_STATISTICS_ON

DVS HotPixel Filter: Number of ON events filtered out by the hot pixel filter.

CAER_FILTER_DVS_HOTPIXEL_STATISTICS_OFF

DVS HotPixel Filter: Number of OFF events filtered out by the hot pixel filter.

CAER_FILTER_DVS_BACKGROUND_ACTIVITY_ENABLE

DVS Background-Activity Filter: enable the background-activity filter, which tries to remove events caused by transistor leakage, by rejecting uncorrelated events.

CAER_FILTER_DVS_BACKGROUND_ACTIVITY_TIME

DVS Background-Activity Filter: specify the time difference constant for the background-activity filter in microseconds. Events that do correlated within this time-frame are let through, while others are filtered out.

CAER_FILTER_DVS_BACKGROUND_ACTIVITY_STATISTICS

DVS Background-Activity Filter: number of events filtered out by the background-activity filter.

CAER_FILTER_DVS_BACKGROUND_ACTIVITY_STATISTICS_ON

DVS Background-Activity Filter: number of ON events filtered out by the background-activity filter.

CAER_FILTER_DVS_BACKGROUND_ACTIVITY_STATISTICS_OFF

DVS Background-Activity Filter: number of OFF events filtered out by the background-activity filter.

CAER_FILTER_DVS_REFRACTORY_PERIOD_ENABLE

DVS Refractory Period Filter: enable the refractory period filter, which limits the firing rate of pixels.

CAER_FILTER_DVS_REFRACTORY_PERIOD_TIME

DVS Refractory Period Filter: specify the time constant for the refractory period filter. Pixels will be inhibited from generating new events during this time after the last even has fired.

CAER_FILTER_DVS_REFRACTORY_PERIOD_STATISTICS

DVS Refractory Period Filter: number of events filtered out by the refractory period filter.

CAER_FILTER_DVS_REFRACTORY_PERIOD_STATISTICS_ON

DVS Refractory Period Filter: number of ON events filtered out by the refractory period filter.

CAER_FILTER_DVS_REFRACTORY_PERIOD_STATISTICS_OFF

DVS Refractory Period Filter: number of OFF events filtered out by the refractory period filter.

CAER_FILTER_DVS_LOG_LEVEL

DVS Noise Filter: set a custom log-level for an instance of the DVS Noise filter.

CAER_FILTER_DVS_RESET

DVS Noise Filter: reset this instance of the filter to its initial state, forgetting any learned hot pixels and clearing the timestamp map and the statistics. This does not change or reset the configuration.

CAER_FILTER_DVS_BACKGROUND_ACTIVITY_TWO_LEVELS

DVS Background-Activity Filter: repeat the background-activity check, that at least one neighbor pixel supports this pixel, on each pixel that supported the current pixel in turn, basically repeating the check for a second level of pixels.

CAER_FILTER_DVS_BACKGROUND_ACTIVITY_SUPPORT_MIN

DVS Background-Activity Filter: minimum number of pixels in the immediate neighborhood that must support the current pixel for it to be considered valid.

CAER_FILTER_DVS_BACKGROUND_ACTIVITY_SUPPORT_MAX

DVS Background-Activity Filter: maximum number of pixels in the immediate neighborhood that can support the current pixel for it to be considered valid.

CAER_FILTER_DVS_BACKGROUND_ACTIVITY_CHECK_POLARITY

DVS Background-Activity Filter: whether polarity is considered when searching the neighbors for supporting activity.

Typedefs

typedef struct *caer_filter_dvs_pixel* ***caerFilterDVSPixel**

Pointer to DVS pixel address structure.

typedef struct caer_filter_dvs_noise ***caerFilterDVSNoise**

Pointer to DVS noise filter structure (private).

Functions

LIBRARY_PUBLIC_VISIBILITY caerFilterDVSNoise caerFilterDVSNoiseInitialize (uint16_t sizeX, uint16_t sizeY)

Allocate memory and initialize the DVS noise filter. This filter combines a HotPixel filter (high activity pixels), a Background-Activity filter (uncorrelated events), and a Refractory Period filter (limit event rate of a pixel). The HotPixel and Background-Activity filters reduce noise due to transistor mismatch, the Refractory Period filter can reduce the event rate and is efficient to implement together with the Background-Activity filter, requiring only one pixel memory map for both. At initialization, all filters are disabled. You must configure and enable them using *caerFilterDVSNoiseConfigSet()*. You must specify the maximum resolution at initialization, as it is used to set up efficient lookup tables.

Parameters

- **sizeX** – maximum X axis resolution.
- **sizeY** – maximum Y axis resolution.

Returns
DVS noise filter instance, NULL on error.

LIBRARY_PUBLIC_VISIBILITY void caerFilterDVSNoiseDestroy (caerFilterDVSNoise noiseFilter)

Destroy a DVS noise filter instance and free its memory.

Parameters
noiseFilter – a valid DVS noise filter instance.

LIBRARY_PUBLIC_VISIBILITY void caerFilterDVSNoiseApply (caerFilterDVSNoise noiseFilter, caerPolarityEventPacket polarity)

Apply the DVS noise filter to the given polarity events packet. This will filter out events by marking them as invalid, depending on the given filter configuration.

Parameters

- **noiseFilter** – a valid DVS noise filter instance.
- **polarity** – a valid polarity event packet. If NULL, no operation is performed.

LIBRARY_PUBLIC_VISIBILITY void caerFilterDVSNoiseStatsApply (caerFilterDVSNoise noiseFilter, caerPolarityEventPacketConst polarity)

Apply the DVS noise filter to the given polarity events packet. This will only gather statistics on the noise, without changing the event packet at all!

Parameters

- **noiseFilter** – a valid DVS noise filter instance.
- **polarity** – a valid polarity event packet. If NULL, no operation is performed.

LIBRARY_PUBLIC_VISIBILITY bool caerFilterDVSNoiseConfigSet (caerFilterDVSNoise noiseFilter, uint8_t paramAddr, uint64_t param)

Set DVS noise filter configuration parameters.

Parameters

- **noiseFilter** – a valid DVS noise filter instance.
- **paramAddr** – a configuration parameter address, see defines CAER_FILTER_DVS_*.

- **param** – a configuration parameter value integer.

Returns
true if operation successful, false otherwise.

`LIBRARY_PUBLIC_VISIBILITY bool caerFilterDVSNoiseConfigGet (caerFilterDVSNoise noiseFilter, uint8_t paramAddr, uint64_t *param)`

Get DVS noise filter configuration parameters.

Parameters

- **noiseFilter** – a valid DVS noise filter instance.
- **paramAddr** – a configuration parameter address, see defines CAER_FILTER_DVS_*.
- **param** – a pointer to a configuration parameter value integer, in which to store the current value.

Returns
true if operation successful, false otherwise.

`LIBRARY_PUBLIC_VISIBILITY ssize_t caerFilterDVSNoiseGetHotPixels (caerFilterDVSNoise noiseFilter, caerFilterDVSPixel *hotPixels)`

Get an array of currently learned hot pixels, in order of activity (most active first, least active last). Useful for working with hardware-based pixel filtering (FPGA/CPLD).

Parameters

- **noiseFilter** – a valid DVS noise filter instance.
- **hotPixels** – array of DVS pixel addresses, sorted by activity (most active first). Memory will be allocated for it automatically. On error, the pointer is set to NULL. Remember to free() the memory once done!

Returns
number of hot pixels in array, 0 if no hot pixels were found; or -1 if an error occurred.

file **frame_utils.h**

*#include "*events/frame.h*"* Functions for frame enhancement and demosaicing. Basic variants that don't require any external dependencies, such as OpenCV. Use of the OpenCV variants is recommended for quality and performance, and can optionally be enabled at build-time.

Enums

enum **caer_frame_utils_demosaic_types**

Values:

enumerator **DEMOSAIC_STANDARD**

enumerator **DEMOSAIC_TO_GRAY**

enum **caer_frame_utils_contrast_types**

Values:

enumerator **CONTRAST_STANDARD**

enum **caer_frame_utils_pixel_color**

Values:

enumerator **PX_COLOR_R**

enumerator **PX_COLOR_B**

enumerator **PX_COLOR_G1**

enumerator **PX_COLOR_G2**

enumerator **PX_COLOR_W**

Functions

LIBRARY_PUBLIC_VISIBILITY void caerFrameUtilsDemosaic (caerFrameEventConst inputFrame, caerFrameEvent outputFrame, enum caer_frame_utils_demosaic_types demosaicType)

LIBRARY_PUBLIC_VISIBILITY void caerFrameUtilsContrast (caerFrameEventConst inputFrame, caerFrameEvent outputFrame, enum caer_frame_utils_contrast_types contrastType)

enum *caer_frame_utils_pixel_color* **caerFrameUtilsPixelColor**(enum *caer_frame_event_color_filter* colorFilter, int32_t x, int32_t y)

file **log.h**

#include <stdarg.h>#include <stdint.h>#include <stddef.h> Logging functions to print useful messages for the user.

Defines

ATTRIBUTE_FORMAT(N)

ATTRIBUTE_FORMAT_VA(N)

LIBRARY_PUBLIC_VISIBILITY

Typedefs

typedef void (*caerLogCallback)(const char *msg, size_t msgLength)

Logging callback, called on any caerLog() invocation. Arguments are the full log string resulting from the caerLog() calls, plus its size (excluding trailing NUL byte).

Enums

enum caer_log_level

Log levels for caerLog() logging function. Log messages only get printed if their log level is equal or above the global system log level, which can be set with caerLogLevelSet(). The default log level is CAER_LOG_ERROR. CAER_LOG_EMERGENCY is the most urgent log level and will always be printed, while CAER_LOG_DEBUG is the least urgent log level and will only be delivered if configured by the user.

Values:

enumerator CAER_LOG_EMERGENCY

enumerator CAER_LOG_ALERT

enumerator CAER_LOG_CRITICAL

enumerator CAER_LOG_ERROR

enumerator CAER_LOG_WARNING

enumerator CAER_LOG_NOTICE

enumerator CAER_LOG_INFO

enumerator CAER_LOG_DEBUG

Functions

__attribute__((visibility("default"))) void caerLogLevelSet(enum *caer_log_level* logLevel)

Set the system-wide log level. Log messages will only be printed if their level is equal or above this level.

Get the current system-wide log level. Log messages are only printed if their level is equal or above this level.

Set callback function to be used on each log message.

Get current callback function for log messages.

Set to which file descriptors log messages are sent. Up to two different file descriptors can be configured here. By default logging to STDERR only is enabled. If both file descriptors are identical, logging to it will only happen once, as if the second one was disabled.

Get the current output file descriptor 1.

Get the current output file descriptor 2.

Disable all logging for this thread only. Call again with different argument to re-enable.

Status of logging for this thread.

Main logging function. This function takes messages, formats them and sends them out to a file descriptor, respecting the system-wide log level setting and prepending the current time, the log level and a user-specified common string to the actual formatted output. The format is specified exactly as with the printf() family of functions. Please see their manual-page for more information.

Secondary logging function. This function takes messages, formats them and sends them out to a file descriptor, respecting the system-wide log level setting and prepending the current time, the log level and a user-specified common string to the actual formatted output. The format is specified exactly as with the printf() family of functions. The argument list is a va_list as returned by va_start(), following the vprintf() family of functions in its functionality. Please see their manual-page for more information.

Tertiary logging function. This function takes messages, formats them and sends them out via up to two file descriptors and a callback; allows a user-given system log level setting to also be specified, and then prepends the current time, the message log level and a user-specified common string to the actual formatted output. The format is specified exactly as with the printf() family of functions. The argument list is a va_list as returned by va_start(), following the vprintf() family of functions in its functionality. Please see their manual-page for more information.

Parameters

- **logLevel** – the system-wide log level.
- **callback** – the callback function. NULL to disable.
- **fd1** – first file descriptor to log to. A negative value will disable it.
- **fd2** – second file descriptor to log to. A negative value will disable it.
- **disableLogging** – true to disable logging for this thread, false to enable it again.
- **logLevel** – the message-specific log level.
- **subSystem** – a common, user-specified string to prepend before the message.
- **format** – the message format string (see printf()).
- **...** – the parameters to be formatted according to the format string (see printf()).
- **logLevel** – the message-specific log level.
- **subSystem** – a common, user-specified string to prepend before the message.

- **format** – the message format string (see printf()).
- **args** – the parameters to be formatted according to the format string (see printf()). This is an argument list as returned by va_start().
- **systemLogLevel** – the system-wide log level.
- **logLevel** – the message-specific log level.
- **subSystem** – a common, user-specified string to prepend before the message.
- **format** – the message format string (see printf()).
- **args** – the parameters to be formatted according to the format string (see printf()). This is an argument list as returned by va_start().

Returns
the current system-wide log level.

Returns
the current output file descriptor 1.

Returns
the current output file descriptor 2.

Returns
true if logging is disabled for this thread, false if it is enabled.

Variables

int **fd2**

const char ***subSystem**

const char const char * format

const char const char const char const char va_list args

file **network.h**

#include "libcaer.h" Useful functions for AEDAT 3.X network streams.

Defines

AEDAT3_NETWORK_HEADER_LENGTH

AEDAT3_NETWORK_MAGIC_NUMBER

AEDAT3_NETWORK_VERSION

AEDAT3_FILE_VERSION

AEDAT3_MAX_UDP_SIZE

Functions

PACKED_STRUCT (struct aedat3_network_header { int64_t magicNumber; int64_t sequenceNumber;int8_t versionNumber;int8_t formatNumber;int16_t sourceID;})

static inline struct aedat3_network_header **caerParseNetworkHeader**(const uint8_t *dataBuffer)

file **portable_endian.h**

#include <stdint.h>#include <string.h> Endianness conversion functions for a wide variety of systems, including Linux, FreeBSD, MacOS X and Windows.

Functions

static inline float **htobeflt**(float val)

static inline float **htoleflt**(float val)

static inline float **beflttoh**(float val)

static inline float **leflttoh**(float val)

file **ringbuffer.h**

#include <stdbool.h>#include <stdint.h>#include <stdlib.h>

Typedefs

typedef struct caer_ring_buffer ***caerRingBuffer**

Functions

caerRingBuffer **caerRingBufferInit**(size_t size)

void **caerRingBufferFree**(*caerRingBuffer* rBuf)

bool **caerRingBufferPut**(*caerRingBuffer* rBuf, void *elem)

bool **caerRingBufferFull**(*caerRingBuffer* rBuf)

void ***caerRingBufferGet**(*caerRingBuffer* rBuf)

void ***caerRingBufferLook**(*caerRingBuffer* rBuf)

bool **caerRingBufferEmpty**(*caerRingBuffer* rBuf)

file **davis.hpp**

#include "../events/frame.hpp"*#include* "../events/imu6.hpp"*#include* "../events/polarity.hpp"*#include* "../events/special.hpp"*#include* "../../libcaer/devices/davis.h"*#include* "usb.hpp"

file **device.hpp**

#include "../libcaer.hpp"#include "../events/packetContainer.hpp"#include "../events/utils.hpp"#include "../../libcaer/devices/device.h"#include <memory>#include <string>

file **device_discover.hpp**

#include "../../libcaer/devices/device_discover.h"#include "davis.hpp"#include "device.hpp"#include "dvs128.hpp"#include "dvs132s.hpp"#include "dvxplorer.hpp"#include "dynapse.hpp"#include "edvs.hpp"#include "samsung_evk.hpp"#include <memory>#include <vector>

file **dvs128.hpp**

#include "../events/polarity.hpp"#include "../events/special.hpp"#include "../../libcaer/devices/dvs128.h"#include "usb.hpp"

file **dvs132s.hpp**

#include "../events/polarity.hpp"#include "../events/special.hpp"#include "../../libcaer/devices/dvs132s.h"#include "usb.hpp"

file **dvxplorer.hpp**

#include "../events/imu6.hpp"#include "../events/polarity.hpp"#include "../events/special.hpp"#include "../../libcaer/devices/dvxplorer.h"#include "usb.hpp"

file **dynapse.hpp**

#include "../events/special.hpp"#include "../events/spike.hpp"#include "../../libcaer/devices/dynapse.h"#include "usb.hpp"

file **edvs.hpp**

#include "../events/polarity.hpp"#include "../events/special.hpp"#include "../../libcaer/devices/edvs.h"#include "serial.hpp"

file **samsung_evk.hpp**

#include "../events/polarity.hpp"#include "../events/special.hpp"#include "../../libcaer/devices/samsung_evk.h"#include "usb.hpp"

file **serial.hpp**

#include "../../libcaer/devices/serial.h"#include "device.hpp"

file **usb.hpp**

#include "../../libcaer/devices/usb.h"#include "device.hpp"

file **common.hpp**

#include "../libcaer.hpp"#include "../../libcaer/events/common.h"#include <cassert>#include <memory>#include <utility>

file **frame.hpp**

#include "../../libcaer/events/frame.h"#include "../../libcaer/frame_utils.h"#include "common.hpp"

Defines

LIBCAER_FRAMECPP_OPENCV_INSTALLED

file **imu6.hpp**

#include "../../libcaer/events/imu6.h"#include "common.hpp"

file **imu9.hpp**

#include "../../libcaer/events/imu9.h"#include "common.hpp"

file **packetContainer.hpp**

#include "../../libcaer/events/packetContainer.h"#include "common.hpp"#include "utils.hpp"#include <memory>#include <utility>#include <vector>

file **polarity.hpp**

#include "../../libcaer/events/polarity.h"#include "common.hpp"

file **special.hpp**

#include "../../libcaer/events/special.h"#include "common.hpp"

file **spike.hpp**

#include "../../libcaer/events/spike.h"#include "common.hpp"

file **utils.hpp**

#include "common.hpp"#include "frame.hpp"#include "imu6.hpp"#include "imu9.hpp"#include "polarity.hpp"#include "special.hpp"#include "spike.hpp"#include <memory>

file **dvs_noise.hpp**

#include "../events/polarity.hpp"#include "../../libcaer/filters/dvs_noise.h"#include <memory>#include <string>#include <vector>

file **libcaer.hpp**

#include "../libcaer/libcaer.h"#include <stdexcept>#include <type_traits>

file **network.hpp**

#include "../libcaer/network.h"#include "libcaer.hpp"

file **ringbuffer.hpp**

#include <atomic>#include <cstdint>#include <stdexcept>#include <vector>

Defines

CACHELINE_SIZE

dir /builds/inivation/dv/libcaer/include/libcaer/devices

dir /builds/inivation/dv/libcaer/include/libcaercpp/devices

dir /builds/inivation/dv/libcaer/include/libcaer/events

dir /builds/inivation/dv/libcaer/include/libcaercpp/events

dir /builds/inivation/dv/libcaer/include/libcaer/filters

dir /builds/inivation/dv/libcaer/include/libcaercpp/filters

dir /builds/inivation/dv/libcaer/include

dir /builds/inivation/dv/libcaer/include/libcaer

dir /builds/inivation/dv/libcaer/include/libcaercpp

Minimal C library to access, configure and get data from neuromorphic sensors and processors. Currently supported devices are the Dynamic Vision Sensor (DVS) and the DAVIS cameras.

INDEX

D

E

F

H

I

L

P

S

T

V