ChibiOS/HAL

5.0.0

Reference Manual

Sun Jun 4 2017 15:00:02

Contents

Chapter 1

ChibiOS/HAL

1.1 Copyright

1.2 Introduction

This document is the Reference Manual for the ChibiOS/HAL hardware abstraction layer.

1.3 Related Documents

- ChibiOS/HAL General Architecture

Chapter 2

Deprecated List

Global canReceive (canp, mailbox, crfp, timeout)

Global canTransmit (canp, mailbox, ctfp, timeout)

Global sdGetWouldBlock (SerialDriver ∗sdp)

Global sdPutWouldBlock (SerialDriver ∗sdp)

Chapter 3

Module Index

3.1 Modules

Here is a list of all modules:

Chapter 4

Hierarchical Index

4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 5

Data Structure Index

5.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 6

File Index

6.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 7

Module Documentation

7.1 ADC Driver

Generic ADC Driver.

7.1.1 Detailed Description

Generic ADC Driver.

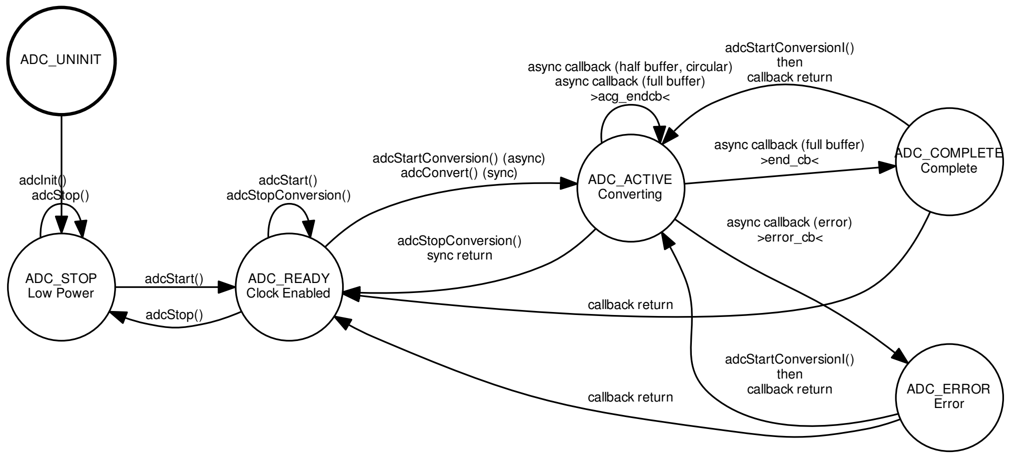
This module implements a generic ADC (Analog to Digital Converter) driver supporting a variety of buffer and conversion modes.

Precondition

In order to use the ADC driver the `HAL_USE_ADC` option must be enabled in `halconf.h`.

7.1.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



7.1.3 ADC Operations

The ADC driver is quite complex, an explanation of the terminology and of the operational details follows.

7.1.3.1 ADC Conversion Groups

The `ADCConversionGroup` is the objects that specifies a physical conversion operation. This structure contains some standard fields and several implementation-dependent fields.
The standard fields define the CG mode, the number of channels belonging to the CG and the optional callbacks.
The implementation-dependent fields specify the physical ADC operation mode, the analog channels belonging to the group and any other implementation-specific setting. Usually the extra fields just mirror the physical ADC registers, please refer to the vendor's MCU Reference Manual for details about the available settings. Details are also available into the documentation of the ADC low level drivers and in the various sample applications.

7.1.3.2 ADC Conversion Modes

The driver supports several conversion modes:

- **One Shot**, the driver performs a single group conversion then stops.
- **Linear Buffer**, the driver performs a series of group conversions then stops. This mode is like a one shot conversion repeated N times, the buffer pointer increases after each conversion. The buffer is organized as an S(CG)$*$N samples matrix, when S(CG) is the conversion group size (number of channels) and N is the buffer depth (number of repeated conversions).
- **Circular Buffer**, much like the linear mode but the operation does not stop when the buffer is filled, it is automatically restarted with the buffer pointer wrapping back to the buffer base.

7.1.3.3 ADC Callbacks

The driver is able to invoke callbacks during the conversion process. A callback is invoked when the operation has been completed or, in circular mode, when the buffer has been filled and the operation is restarted. In circular mode a callback is also invoked when the buffer is half filled.
The "half filled" and "filled" callbacks in circular mode allow to implement "streaming processing" of the sampled

data, while the driver is busy filling one half of the buffer the application can process the other half, this allows for continuous interleaved operations.

The driver is not thread safe for performance reasons, if you need to access the ADC bus from multiple threads then use the `adcAcquireBus()` and `adcReleaseBus()` APIs in order to gain exclusive access.

ADC configuration options

- #define ADC_USE_WAIT TRUE

  Enables synchronous APIs.
- #define ADC_USE_MUTUAL_EXCLUSION TRUE

  Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.

Low level driver helper macros

- #define _adc_reset_i(adcp) osalThreadResumeI(&(adcp)->thread, MSG_RESET)

  Resumes a thread waiting for a conversion completion.
- #define _adc_reset_s(adcp) osalThreadResumeS(&(adcp)->thread, MSG_RESET)

  Resumes a thread waiting for a conversion completion.
- #define _adc_wakeup_isr(adcp)

  Wakes up the waiting thread.
- #define _adc_timeout_isr(adcp)

  Wakes up the waiting thread with a timeout message.
- #define _adc_isr_half_code(adcp)

  Common ISR code, half buffer event.
- #define _adc_isr_full_code(adcp)

  Common ISR code, full buffer event.
- #define _adc_isr_error_code(adcp, err)

  Common ISR code, error event.

PLATFORM configuration options

- #define PLATFORM_ADC_USE_ADC1 FALSE

  ADC1 driver enable switch.

Typedefs

- typedef uint16_t adcsample_t

  ADC sample data type.
- typedef uint16_t adc_channels_num_t

  Channels number in a conversion group.
- typedef struct ADCDriver ADCDriver

  Type of a structure representing an ADC driver.
- typedef void(∗ adccallback_t) (ADCDriver ∗adcp, adcsample_t ∗buffer, size_t n)

  ADC notification callback type.
- typedef void(∗ adcerrorcallback_t) (ADCDriver ∗adcp, adcerror_t err)

  ADC error callback type.

Data Structures

- struct ADCConversionGroup

  Conversion group configuration structure.
- struct ADCConfig

  Driver configuration structure.
- struct ADCDriver

  Structure representing an ADC driver.

Functions

- void adcInit (void)

  ADC Driver initialization.
- void adcObjectInit (ADCDriver ∗adcp)

  Initializes the standard part of a `ADCDriver` structure.
- void adcStart (ADCDriver ∗adcp, const ADCConfig ∗config)

  Configures and activates the ADC peripheral.
- void adcStop (ADCDriver ∗adcp)

  Deactivates the ADC peripheral.
- void adcStartConversion (ADCDriver ∗adcp, const ADCConversionGroup ∗grpp, adcsample_t ∗samples, size_t depth)

  Starts an ADC conversion.
- void adcStartConversionI (ADCDriver ∗adcp, const ADCConversionGroup ∗grpp, adcsample_t ∗samples, size_t depth)

  Starts an ADC conversion.
- void adcStopConversion (ADCDriver ∗adcp)

  Stops an ongoing conversion.
- void adcStopConversionI (ADCDriver ∗adcp)

  Stops an ongoing conversion.
- msg_t adcConvert (ADCDriver ∗adcp, const ADCConversionGroup ∗grpp, adcsample_t ∗samples, size_t depth)

  Performs an ADC conversion.
- void adcAcquireBus (ADCDriver ∗adcp)

  Gains exclusive access to the ADC peripheral.
- void adcReleaseBus (ADCDriver ∗adcp)

  Releases exclusive access to the ADC peripheral.
- void adc_lld_init (void)

  Low level ADC driver initialization.
- void adc_lld_start (ADCDriver ∗adcp)

  Configures and activates the ADC peripheral.
- void adc_lld_stop (ADCDriver ∗adcp)

  Deactivates the ADC peripheral.
- void adc_lld_start_conversion (ADCDriver ∗adcp)

  Starts an ADC conversion.
- void adc_lld_stop_conversion (ADCDriver ∗adcp)

  Stops an ongoing conversion.

Enumerations

Variables

- ADCDriver ADCD1

  ADC1 driver identifier.

7.1.4 Macro Definition Documentation

7.1.4.1 #define ADC_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.1.4.2 #define ADC_USE_MUTUAL_EXCLUSION TRUE

Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.

Note

Disabling this option saves both code and data space.

7.1.4.3 #define _adc_reset_i(*adcp*) osalThreadResumeI(&(adcp)->thread, MSG_RESET)

Resumes a thread waiting for a conversion completion.

Parameters

`in`	*adcp*	pointer to the `ADCDriver` object

Function Class:

Not an API, this function is for internal use only.

7.1.4.4 #define _adc_reset_s(*adcp*) osalThreadResumeS(&(adcp)->thread, MSG_RESET)

Resumes a thread waiting for a conversion completion.

Parameters

`in`	*adcp*	pointer to the `ADCDriver` object

Function Class:

Not an API, this function is for internal use only.

7.1.4.5 #define _adc_wakeup_isr(*adcp*)

Value:

```
{                                              \
  osalSysLockFromISR();                                                   \
  osalThreadResumeI(&(adcp)->thread, MSG_OK);                             \
  osalSysUnlockFromISR();                                                 \
}
```

Wakes up the waiting thread.

Parameters

in	*adcp*	pointer to the ADCDriver object

Function Class:

Not an API, this function is for internal use only.

7.1.4.6 #define _adc_timeout_isr(*adcp*)

Value:

```
{                                                                    \
  osalSysLockFromISR();                                              \
  osalThreadResumeI(&(adcp)->thread, MSG_TIMEOUT);                   \
  osalSysUnlockFromISR();                                            \
}
```

Wakes up the waiting thread with a timeout message.

Parameters

in	*adcp*	pointer to the ADCDriver object

Function Class:

Not an API, this function is for internal use only.

7.1.4.7 #define _adc_isr_half_code(*adcp*)

Value:

```
{                                                                    \
  if ((adcp)->grpp->end_cb != NULL) {                                \
    (adcp)->grpp->end_cb(adcp, (adcp)->samples, (adcp)->depth / 2);  \
  }                                                                  \
}
```

Common ISR code, half buffer event.

This code handles the portable part of the ISR code:

- Callback invocation.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

in	*adcp*	pointer to the ADCDriver object

Function Class:

Not an API, this function is for internal use only.

7.1.4.8 #define _adc_isr_full_code(*adcp*)

Common ISR code, full buffer event.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

in	*adcp*	pointer to the ADCDriver object

Function Class:

Not an API, this function is for internal use only.

7.1.4.9 #define _adc_isr_error_code(*adcp*, *err*)

Value:

```
{                                                         \
  adc_lld_stop_conversion(adcp);                                                    \
  if ((adcp)->grpp->error_cb != NULL) {                                             \
    (adcp)->state = ADC_ERROR;                                                      \
    (adcp)->grpp->error_cb(adcp, err);                                              \
    if ((adcp)->state == ADC_ERROR)                                                 \
      (adcp)->state = ADC_READY;                                                    \
      (adcp)->grpp = NULL;                                                          \
  }                                                                                 \
  else {                                                                            \
    (adcp)->state = ADC_READY;                                                      \
    (adcp)->grpp = NULL;                                                            \
  }                                                                                 \
  _adc_timeout_isr(adcp);                                                           \
}
```

Common ISR code, error event.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread timeout signaling, if any.
- Driver state transitions.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

in	*adcp*	pointer to the ADCDriver object
in	*err*	platform dependent error code

Function Class:

Not an API, this function is for internal use only.

7.1.4.10 #define PLATFORM_ADC_USE_ADC1 FALSE

ADC1 driver enable switch.

If set to `TRUE` the support for ADC1 is included.

Note

The default is `FALSE`.

7.1.5 Typedef Documentation

7.1.5.1 typedef uint16_t **adcsample_t**

ADC sample data type.

7.1.5.2 typedef uint16_t **adc_channels_num_t**

Channels number in a conversion group.

7.1.5.3 typedef struct **ADCDriver ADCDriver**

Type of a structure representing an ADC driver.

7.1.5.4 typedef void(∗ adccallback_t) (**ADCDriver** ∗adcp, **adcsample_t** ∗buffer, size_t n)

ADC notification callback type.

Parameters

`in`	*adcp*	pointer to the `ADCDriver` object triggering the callback
`in`	*buffer*	pointer to the most recent samples data
`in`	*n*	number of buffer rows available starting from `buffer`

7.1.5.5 typedef void(∗ adcerrorcallback_t) (**ADCDriver** ∗adcp, **adcerror_t** err)

ADC error callback type.

Parameters

`in`	*adcp*	pointer to the `ADCDriver` object triggering the callback
`in`	*err*	ADC error code

7.1.6 Enumeration Type Documentation

7.1.6.1 enum adcstate_t

Driver state machine possible states.

Enumerator

ADC_UNINIT	Not initialized.
ADC_STOP	Stopped.
ADC_READY	Ready.
ADC_ACTIVE	Converting.
ADC_COMPLETE	Conversion complete.
ADC_ERROR	Conversion error.

7.1.6.2 enum adcerror_t

Possible ADC failure causes.

Note

Error codes are architecture dependent and should not relied upon.

Enumerator

ADC_ERR_DMAFAILURE	DMA operations failure.
ADC_ERR_OVERFLOW	ADC overflow condition.
ADC_ERR_AWD	Analog watchdog triggered.

7.1.7 Function Documentation

7.1.7.1 void adcInit (void)

ADC Driver initialization.

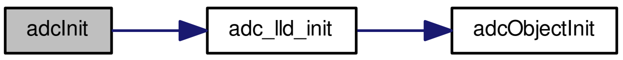
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



7.1.7.2 void adcObjectInit (**ADCDriver** ∗ *adcp*)

Initializes the standard part of a `ADCDriver` structure.

Parameters

out	*adcp*	pointer to the ADCDriver object

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.1.7.3 void adcStart (ADCDriver * *adcp*, const ADCConfig * *config*)

Configures and activates the ADC peripheral.

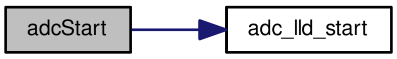
Parameters

in	*adcp*	pointer to the ADCDriver object
in	*config*	pointer to the ADCConfig object. Depending on the implementation the value can be NULL.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.1.7.4 void adcStop (ADCDriver * *adcp*)

Deactivates the ADC peripheral.

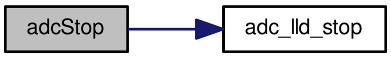
Parameters

in	*adcp*	pointer to the ADCDriver object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.1.7.5 void adcStartConversion (**ADCDriver** ∗ *adcp,* const **ADCConversionGroup** ∗ *grpp,* **adcsample_t** ∗ *samples,* size_t *depth*)

Starts an ADC conversion.

Starts an asynchronous conversion operation.

Note

The buffer is organized as a matrix of M∗N elements where M is the channels number configured into the conversion group and N is the buffer depth. The samples are sequentially written into the buffer with no gaps.

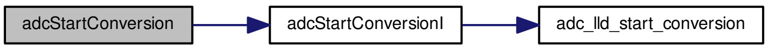
Parameters

`in`	*adcp*	pointer to the `ADCDriver` object
`in`	*grpp*	pointer to a `ADCConversionGroup` object
`out`	*samples*	pointer to the samples buffer
`in`	*depth*	buffer depth (matrix rows number). The buffer depth must be one or an even number.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.1.7.6 void adcStartConversionI (**ADCDriver** ∗ *adcp,* const **ADCConversionGroup** ∗ *grpp,* **adcsample_t** ∗ *samples,* size_t *depth*)

Starts an ADC conversion.

Starts an asynchronous conversion operation.

Postcondition

The callbacks associated to the conversion group will be invoked on buffer fill and error events.

Note

The buffer is organized as a matrix of M∗N elements where M is the channels number configured into the conversion group and N is the buffer depth. The samples are sequentially written into the buffer with no gaps.

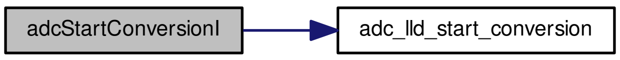
Parameters

`in`	*adcp*	pointer to the `ADCDriver` object
`in`	*grpp*	pointer to a `ADCConversionGroup` object
`out`	*samples*	pointer to the samples buffer
`in`	*depth*	buffer depth (matrix rows number). The buffer depth must be one or an even number.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



7.1.7.7 void adcStopConversion (**ADCDriver** ∗ *adcp*)

Stops an ongoing conversion.

This function stops the currently ongoing conversion and returns the driver in the `ADC_READY` state. If there was no conversion being processed then the function does nothing.

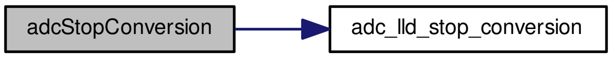
Parameters

`in`	*adcp*	pointer to the `ADCDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.1.7.8 void adcStopConversionI (**ADCDriver** ∗ *adcp*)

Stops an ongoing conversion.

This function stops the currently ongoing conversion and returns the driver in the `ADC_READY` state. If there was no conversion being processed then the function does nothing.

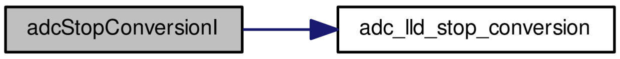
Parameters

`in`	*adcp*	pointer to the `ADCDriver` object

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



7.1.7.9 msg_t adcConvert (**ADCDriver** ∗ *adcp,* const **ADCConversionGroup** ∗ *grpp,* **adcsample_t** ∗ *samples,* size_t *depth*)

Performs an ADC conversion.

Performs a synchronous conversion operation.

Note

The buffer is organized as a matrix of M∗N elements where M is the channels number configured into the conversion group and N is the buffer depth. The samples are sequentially written into the buffer with no gaps.

Parameters

`in`	*adcp*	pointer to the `ADCDriver` object
`in`	*grpp*	pointer to a `ADCConversionGroup` object
`out`	*samples*	pointer to the samples buffer
`in`	*depth*	buffer depth (matrix rows number). The buffer depth must be one or an even number.

Returns

The operation result.

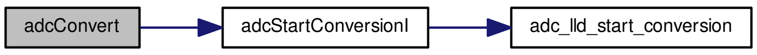
Return values

MSG_OK	Conversion finished.
MSG_RESET	The conversion has been stopped using `acdStopConversion()` or `acdStopConversionI()`, the result buffer may contain incorrect data.
MSG_TIMEOUT	The conversion has been stopped because an hardware error.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.1.7.10 void adcAcquireBus (ADCDriver * *adcp*)

Gains exclusive access to the ADC peripheral.

This function tries to gain ownership to the ADC bus, if the bus is already being used then the invoking thread is queued.

Precondition

In order to use this function the option `ADC_USE_MUTUAL_EXCLUSION` must be enabled.

Parameters

`in`	*adcp*	pointer to the `ADCDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.1.7.11 void adcReleaseBus (**ADCDriver** ∗ *adcp*)

Releases exclusive access to the ADC peripheral.

Precondition

In order to use this function the option `ADC_USE_MUTUAL_EXCLUSION` must be enabled.

Parameters

in	*adcp*	pointer to the `ADCDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

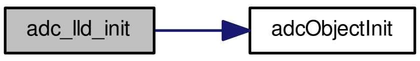
7.1.7.12 void adc_lld_init (void)

Low level ADC driver initialization.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.1.7.13 void adc_lld_start (**ADCDriver** ∗ *adcp*)

Configures and activates the ADC peripheral.

Parameters

in	*adcp*	pointer to the `ADCDriver` object

Function Class:

Not an API, this function is for internal use only.

7.1.7.14 void adc_lld_stop (**ADCDriver** ∗ *adcp*)

Deactivates the ADC peripheral.

Parameters

`in`	*adcp*	pointer to the `ADCDriver` object

Function Class:

Not an API, this function is for internal use only.

7.1.7.15 void adc_lld_start_conversion (**ADCDriver** ∗ *adcp*)

Starts an ADC conversion.

Parameters

`in`	*adcp*	pointer to the `ADCDriver` object

Function Class:

Not an API, this function is for internal use only.

7.1.7.16 void adc_lld_stop_conversion (**ADCDriver** ∗ *adcp*)

Stops an ongoing conversion.

Parameters

`in`	*adcp*	pointer to the `ADCDriver` object

Function Class:

Not an API, this function is for internal use only.

7.1.8 Variable Documentation

7.1.8.1 ADCDriver ADCD1

ADC1 driver identifier.

7.2 CAN Driver

Generic CAN Driver.

7.2.1 Detailed Description

Generic CAN Driver.

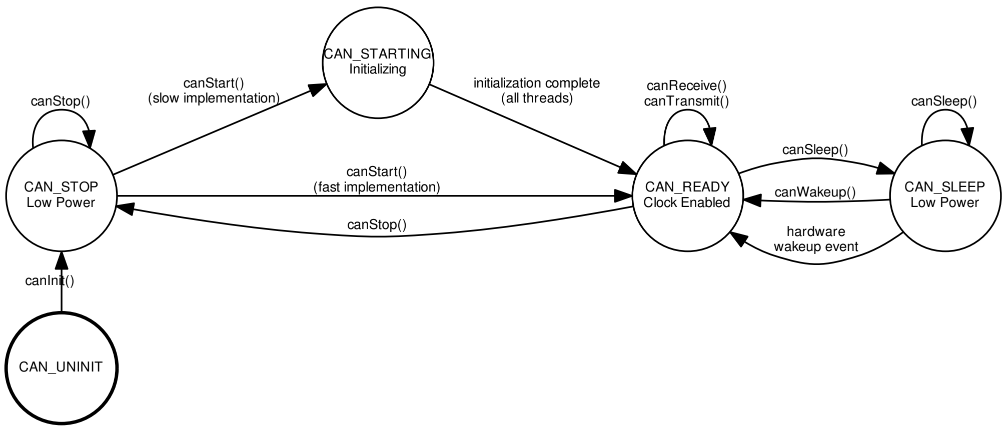
This module implements a generic CAN (Controller Area Network) driver allowing the exchange of information at frame level.

Precondition

In order to use the CAN driver the `HAL_USE_CAN` option must be enabled in `halconf.h`.

7.2.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



Macros

- #define CAN_ANY_MAILBOX 0

    Special mailbox identifier.
- #define CAN_TX_MAILBOXES 1

    Number of transmit mailboxes.
- #define CAN_RX_MAILBOXES 1

    Number of receive mailboxes.

CAN status flags

- #define CAN_LIMIT_WARNING 1U

Errors rate warning.

- #define CAN_LIMIT_ERROR 2U

  Errors rate error.
- #define CAN_BUS_OFF_ERROR 4U

  Bus off condition reached.
- #define CAN_FRAMING_ERROR 8U

  Framing error of some kind on the CAN bus.
- #define CAN_OVERFLOW_ERROR 16U

  Overflow in receive queue.

CAN configuration options

- #define CAN_USE_SLEEP_MODE TRUE

  Sleep mode related APIs inclusion switch.

Macro Functions

- #define CAN_MAILBOX_TO_MASK(mbx) (1U << ((mbx) - 1U))

  Converts a mailbox index to a bit mask.
- #define canTransmit(canp, mailbox, ctfp, timeout) canTransmitTimeout(canp, mailbox, ctfp, timeout)

  Legacy name for `canTransmitTimeout()`.
- #define canReceive(canp, mailbox, crfp, timeout) canReceiveTimeout(canp, mailbox, crfp, timeout)

  Legacy name for `canReceiveTimeout()`.

PLATFORM configuration options

- #define PLATFORM_CAN_USE_CAN1 FALSE

  CAN1 driver enable switch.

Typedefs

- typedef uint32_t canmbx_t

  Type of a transmission mailbox index.

Data Structures

- struct CANTxFrame

  CAN transmission frame.
- struct CANRxFrame

  CAN received frame.
- struct CANConfig

  Driver configuration structure.
- struct CANDriver

  Structure representing an CAN driver.

Functions

- void canInit (void)

  CAN Driver initialization.
- void canObjectInit (CANDriver ∗canp)

  Initializes the standard part of a `CANDriver` structure.
- void canStart (CANDriver ∗canp, const CANConfig ∗config)

  Configures and activates the CAN peripheral.
- void canStop (CANDriver ∗canp)

  Deactivates the CAN peripheral.
- bool canTryTransmitI (CANDriver ∗canp, canmbx_t mailbox, const CANTxFrame ∗ctfp)

  Can frame transmission attempt.
- bool canTryReceiveI (CANDriver ∗canp, canmbx_t mailbox, CANRxFrame ∗crfp)

  Can frame receive attempt.
- msg_t canTransmitTimeout (CANDriver ∗canp, canmbx_t mailbox, const CANTxFrame ∗ctfp, systime_t time-out)

  Can frame transmission.
- msg_t canReceiveTimeout (CANDriver ∗canp, canmbx_t mailbox, CANRxFrame ∗crfp, systime_t timeout)

  Can frame receive.
- void canSleep (CANDriver ∗canp)

  Enters the sleep mode.
- void canWakeup (CANDriver ∗canp)

  Enforces leaving the sleep mode.
- void can_lld_init (void)

  Low level CAN driver initialization.
- void can_lld_start (CANDriver ∗canp)

  Configures and activates the CAN peripheral.
- void can_lld_stop (CANDriver ∗canp)

  Deactivates the CAN peripheral.
- bool can_lld_is_tx_empty (CANDriver ∗canp, canmbx_t mailbox)

  Determines whether a frame can be transmitted.
- void can_lld_transmit (CANDriver ∗canp, canmbx_t mailbox, const CANTxFrame ∗ctfp)

  Inserts a frame into the transmit queue.
- bool can_lld_is_rx_nonempty (CANDriver ∗canp, canmbx_t mailbox)

  Determines whether a frame has been received.
- void can_lld_receive (CANDriver ∗canp, canmbx_t mailbox, CANRxFrame ∗crfp)

  Receives a frame from the input queue.
- void can_lld_sleep (CANDriver ∗canp)

  Enters the sleep mode.
- void can_lld_wakeup (CANDriver ∗canp)

  Enforces leaving the sleep mode.

Enumerations

Variables

- CANDriver CAND1

  CAN1 driver identifier.

7.2.3 Macro Definition Documentation

7.2.3.1 #define CAN_LIMIT_WARNING 1U

Errors rate warning.

7.2.3.2 #define CAN_LIMIT_ERROR 2U

Errors rate error.

7.2.3.3 #define CAN_BUS_OFF_ERROR 4U

Bus off condition reached.

7.2.3.4 #define CAN_FRAMING_ERROR 8U

Framing error of some kind on the CAN bus.

7.2.3.5 #define CAN_OVERFLOW_ERROR 16U

Overflow in receive queue.

7.2.3.6 #define CAN_ANY_MAILBOX 0

Special mailbox identifier.

7.2.3.7 #define CAN_USE_SLEEP_MODE TRUE

Sleep mode related APIs inclusion switch.

This option can only be enabled if the CAN implementation supports the sleep mode, see the macro `CAN_SUPP↩ ORTS_SLEEP` exported by the underlying implementation.

7.2.3.8 #define CAN_MAILBOX_TO_MASK(*mbx*) (1U << ((mbx) - 1U))

Converts a mailbox index to a bit mask.

7.2.3.9 #define canTransmit(*canp,* *mailbox,* *ctfp,* *timeout*) canTransmitTimeout(canp, mailbox, ctfp, timeout)

Legacy name for `canTransmitTimeout()`.

Deprecated

7.2.3.10 #define canReceive(*canp,* *mailbox,* *crfp,* *timeout*) canReceiveTimeout(canp, mailbox, crfp, timeout)

Legacy name for `canReceiveTimeout()`.

Deprecated

7.2.3.11 #define CAN_TX_MAILBOXES 1

Number of transmit mailboxes.

7.2.3.12 #define CAN_RX_MAILBOXES 1

Number of receive mailboxes.

7.2.3.13 #define PLATFORM_CAN_USE_CAN1 FALSE

CAN1 driver enable switch.

If set to `TRUE` the support for CAN1 is included.

Note

The default is `FALSE`.

7.2.4 Typedef Documentation

7.2.4.1 typedef uint32_t canmbx_t

Type of a transmission mailbox index.

7.2.5 Enumeration Type Documentation

7.2.5.1 enum canstate_t

Driver state machine possible states.

Enumerator

CAN_UNINIT	Not initialized.
CAN_STOP	Stopped.
CAN_STARTING	Starting.
CAN_READY	Ready.
CAN_SLEEP	Sleep state.

7.2.6 Function Documentation

7.2.6.1 void canInit (void)

CAN Driver initialization.

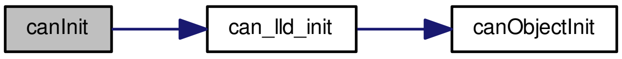
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



7.2.6.2 void canObjectInit (**CANDriver** * *canp*)

Initializes the standard part of a `CANDriver` structure.

Parameters

`out`	*canp*	pointer to the `CANDriver` object

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.2.6.3 void canStart (**CANDriver** * *canp*, const **CANConfig** * *config*)

Configures and activates the CAN peripheral.

Note

Activating the CAN bus can be a slow operation.
Unlike other drivers it is not possible to restart the CAN driver without first stopping it using canStop().

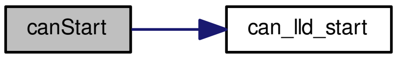
Parameters

`in`	*canp*	pointer to the `CANDriver` object
`in`	*config*	pointer to the `CANConfig` object. Depending on the implementation the value can be `NULL`.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.2.6.4 void canStop (CANDriver * *canp*)

Deactivates the CAN peripheral.

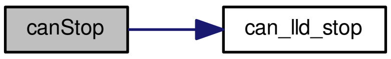
Parameters

`in`	*canp*	pointer to the `CANDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.2.6.5 bool canTryTransmitI (CANDriver * *canp*, canmbx_t *mailbox*, const CANTxFrame * *ctfp*)

Can frame transmission attempt.

The specified frame is queued for transmission, if the hardware queue is full then the function fails.

Parameters

`in`	*canp*	pointer to the `CANDriver` object
`in`	*mailbox*	mailbox number, `CAN_ANY_MAILBOX` for any mailbox
`in`	*ctfp*	pointer to the CAN frame to be transmitted

Returns

The operation result.

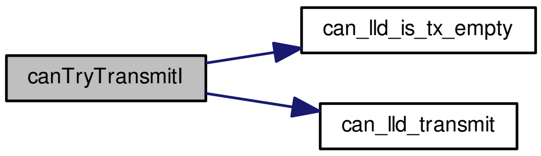
Return values

false	Frame transmitted.
true	Mailbox full.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



7.2.6.6 bool canTryReceiveI (**CANDriver** ∗ *canp*, **canmbx_t** *mailbox*, **CANRxFrame** ∗ *crfp*)

Can frame receive attempt.

The function tries to fetch a frame from a mailbox.

Parameters

`in`	*canp*	pointer to the `CANDriver` object
`in`	*mailbox*	mailbox number, `CAN_ANY_MAILBOX` for any mailbox
`out`	*crfp*	pointer to the buffer where the CAN frame is copied

Returns

The operation result.

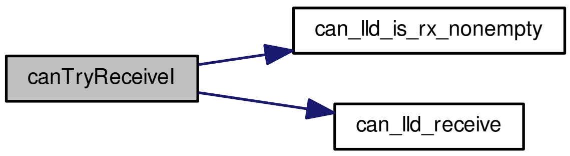
Return values

false	Frame fetched.
true	Mailbox empty.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



7.2.6.7 msg_t canTransmitTimeout (**CANDriver** ∗ *canp,* **canmbx_t** *mailbox,* const **CANTxFrame** ∗ *ctfp,* systime_t *timeout*)

Can frame transmission.

The specified frame is queued for transmission, if the hardware queue is full then the invoking thread is queued.

Note

Trying to transmit while in sleep mode simply enqueues the thread.

Parameters

`in`	*canp*	pointer to the `CANDriver` object
`in`	*mailbox*	mailbox number, `CAN_ANY_MAILBOX` for any mailbox
`in`	*ctfp*	pointer to the CAN frame to be transmitted
`in`	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The operation result.

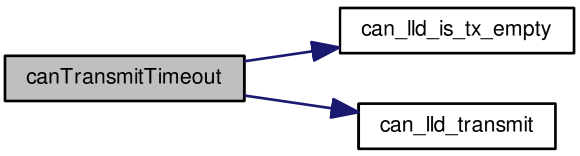
Return values

MSG_OK	the frame has been queued for transmission.
MSG_TIMEOUT	The operation has timed out.
MSG_RESET	The driver has been stopped while waiting.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.2.6.8 msg_t canReceiveTimeout (**CANDriver** ∗ *canp,* **canmbx_t** *mailbox,* **CANRxFrame** ∗ *crfp,* systime_t *timeout*)

Can frame receive.

The function waits until a frame is received.

Note

Trying to receive while in sleep mode simply enqueues the thread.

Parameters

`in`	*canp*	pointer to the `CANDriver` object
`in`	*mailbox*	mailbox number, `CAN_ANY_MAILBOX` for any mailbox
`out`	*crfp*	pointer to the buffer where the CAN frame is copied
`in`	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout (useful in an event driven scenario where a thread never blocks for I/O). • *TIME_INFINITE* no timeout.

Returns

The operation result.

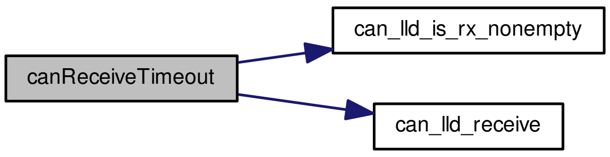
Return values

MSG_OK	a frame has been received and placed in the buffer.
MSG_TIMEOUT	The operation has timed out.
MSG_RESET	The driver has been stopped while waiting.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.2.6.9 void canSleep (CANDriver * *canp*)

Enters the sleep mode.

This function puts the CAN driver in sleep mode and broadcasts the `sleep_event` event source.

Precondition

In order to use this function the option `CAN_USE_SLEEP_MODE` must be enabled and the `CAN_SUPPOR↩ TS_SLEEP` mode must be supported by the low level driver.

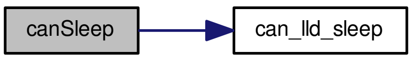
Parameters

in	*canp*	pointer to the `CANDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.2.6.10 void canWakeup (CANDriver * *canp*)

Enforces leaving the sleep mode.

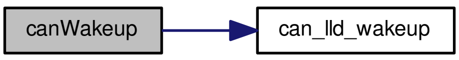
Note

The sleep mode is supposed to be usually exited automatically by an hardware event.

Parameters

in	*canp*	pointer to the `CANDriver` object

Here is the call graph for this function:



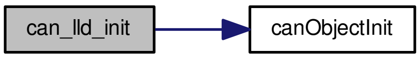
7.2.6.11 void can_lld_init (void)

Low level CAN driver initialization.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.2.6.12 void can_lld_start (CANDriver ∗ *canp*)

Configures and activates the CAN peripheral.

Parameters

`in`	*canp*	pointer to the `CANDriver` object

Function Class:

Not an API, this function is for internal use only.

7.2.6.13 void can_lld_stop (CANDriver ∗ *canp*)

Deactivates the CAN peripheral.

Parameters

`in`	*canp*	pointer to the `CANDriver` object

Function Class:

Not an API, this function is for internal use only.

7.2.6.14 bool can_lld_is_tx_empty (CANDriver ∗ *canp*, canmbx_t *mailbox*)

Determines whether a frame can be transmitted.

Parameters

`in`	*canp*	pointer to the `CANDriver` object
`in`	*mailbox*	mailbox number, `CAN_ANY_MAILBOX` for any mailbox

Returns

The queue space availability.

Return values

FALSE	no space in the transmit queue.
TRUE	transmit slot available.

Function Class:

Not an API, this function is for internal use only.

7.2.6.15 void can_lld_transmit (**CANDriver** ∗ *canp,* **canmbx_t** *mailbox,* const **CANTxFrame** ∗ *ctfp*)

Inserts a frame into the transmit queue.

Parameters

`in`	*canp*	pointer to the `CANDriver` object
`in`	*ctfp*	pointer to the CAN frame to be transmitted
`in`	*mailbox*	mailbox number, `CAN_ANY_MAILBOX` for any mailbox

Function Class:

Not an API, this function is for internal use only.

7.2.6.16 bool can_lld_is_rx_nonempty (**CANDriver** ∗ *canp,* **canmbx_t** *mailbox*)

Determines whether a frame has been received.

Parameters

`in`	*canp*	pointer to the `CANDriver` object
`in`	*mailbox*	mailbox number, `CAN_ANY_MAILBOX` for any mailbox

Returns

The queue space availability.

Return values

FALSE	no space in the transmit queue.
TRUE	transmit slot available.

Function Class:

Not an API, this function is for internal use only.

7.2.6.17 void can_lld_receive (CANDriver ∗ *canp*, canmbx_t *mailbox*, CANRxFrame ∗ *crfp*)

Receives a frame from the input queue.

Parameters

in	*canp*	pointer to the `CANDriver` object
in	*mailbox*	mailbox number, `CAN_ANY_MAILBOX` for any mailbox
out	*crfp*	pointer to the buffer where the CAN frame is copied

Function Class:

Not an API, this function is for internal use only.

7.2.6.18 void can_lld_sleep (CANDriver ∗ *canp*)

Enters the sleep mode.

Parameters

in	*canp*	pointer to the `CANDriver` object

Function Class:

Not an API, this function is for internal use only.

7.2.6.19 void can_lld_wakeup (CANDriver ∗ *canp*)

Enforces leaving the sleep mode.

Parameters

in	*canp*	pointer to the `CANDriver` object

Function Class:

Not an API, this function is for internal use only.

7.2.7 Variable Documentation

7.2.7.1 CANDriver CAND1

CAN1 driver identifier.

7.3 DAC Driver

Generic DAC Driver.

7.3.1 Detailed Description

Generic DAC Driver.

This module implements a generic DAC (Digital to Analog Converter) driver.

Precondition

In order to use the MAC driver the `HAL_USE_DAC` option must be enabled in `halconf.h`.

Macros

- #define DAC_MAX_CHANNELS 2

  Maximum number of DAC channels per unit.

DAC configuration options

- #define DAC_USE_WAIT TRUE

  Enables synchronous APIs.
- #define DAC_USE_MUTUAL_EXCLUSION TRUE

  Enables the `dacAcquireBus()` and `dacReleaseBus()` APIs.

Low level driver helper macros

- #define _dac_wait_s(dacp) osalThreadSuspendS(&(dacp)->thread)

  Waits for operation completion.
- #define _dac_reset_i(dacp) osalThreadResumeI(&(dacp)->thread, MSG_RESET)

  Resumes a thread waiting for a conversion completion.
- #define _dac_reset_s(dacp) osalThreadResumeS(&(dacp)->thread, MSG_RESET)

  Resumes a thread waiting for a conversion completion.
- #define _dac_wakeup_isr(dacp)

  Wakes up the waiting thread.
- #define _dac_timeout_isr(dacp)

  Wakes up the waiting thread with a timeout message.
- #define _dac_isr_half_code(dacp)

  Common ISR code, half buffer event.
- #define _dac_isr_full_code(dacp)

  Common ISR code, full buffer event.
- #define _dac_isr_error_code(dacp, err)

  Common ISR code, error event.

Configuration options

- #define PLATFORM_DAC_USE_DAC1 FALSE

  DAC1 CH1 driver enable switch.

Typedefs

- typedef uint32_t dacchannel_t

  Type of a DAC channel index.
- typedef struct DACDriver DACDriver

  Type of a structure representing an DAC driver.
- typedef uint16_t dacsample_t

  Type representing a DAC sample.
- typedef void(∗ daccallback_t) (DACDriver ∗dacp, dacsample_t ∗buffer, size_t n)

  DAC notification callback type.
- typedef void(∗ dacerrorcallback_t) (DACDriver ∗dacp, dacerror_t err)

  ADC error callback type.

Data Structures

- struct DACConversionGroup

  DAC Conversion group structure.
- struct DACConfig

  Driver configuration structure.
- struct DACDriver

  Structure representing a DAC driver.

Functions

- void dacInit (void)

  DAC Driver initialization.
- void dacObjectInit (DACDriver ∗dacp)

  Initializes the standard part of a `DACDriver` structure.
- void dacStart (DACDriver ∗dacp, const DACConfig ∗config)

  Configures and activates the DAC peripheral.
- void dacStop (DACDriver ∗dacp)

  Deactivates the DAC peripheral.
- void dacPutChannelX (DACDriver ∗dacp, dacchannel_t channel, dacsample_t sample)

  Outputs a value directly on a DAC channel.
- void dacStartConversion (DACDriver ∗dacp, const DACConversionGroup ∗grpp, dacsample_t ∗samples, size_t depth)

  Starts a DAC conversion.
- void dacStartConversionI (DACDriver ∗dacp, const DACConversionGroup ∗grpp, dacsample_t ∗samples, size_t depth)

  Starts a DAC conversion.
- void dacStopConversion (DACDriver ∗dacp)

  Stops an ongoing conversion.
- void dacStopConversionI (DACDriver ∗dacp)

  Stops an ongoing conversion.
- msg_t dacConvert (DACDriver ∗dacp, const DACConversionGroup ∗grpp, dacsample_t ∗samples, size_t depth)

  Performs a DAC conversion.
- void dacAcquireBus (DACDriver ∗dacp)

  Gains exclusive access to the DAC bus.
- void dacReleaseBus (DACDriver ∗dacp)

  Releases exclusive access to the DAC bus.

- void dac_lld_init (void)

  Low level DAC driver initialization.
- void dac_lld_start (DACDriver ∗dacp)

  Configures and activates the DAC peripheral.
- void dac_lld_stop (DACDriver ∗dacp)

  Deactivates the DAC peripheral.
- void dac_lld_put_channel (DACDriver ∗dacp, dacchannel_t channel, dacsample_t sample)

  Outputs a value directly on a DAC channel.
- void dac_lld_start_conversion (DACDriver ∗dacp)

  Starts a DAC conversion.
- void dac_lld_stop_conversion (DACDriver ∗dacp)

  Stops an ongoing conversion.

Enumerations

Variables

- DACDriver DACD1

  DAC1 driver identifier.

7.3.2 Macro Definition Documentation

7.3.2.1 #define DAC_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.3.2.2 #define DAC_USE_MUTUAL_EXCLUSION TRUE

Enables the `dacAcquireBus()` and `dacReleaseBus()` APIs.

Note

Disabling this option saves both code and data space.

7.3.2.3 #define _dac_wait_s(*dacp*) osalThreadSuspendS(&(dacp)->thread)

Waits for operation completion.

This function waits for the driver to complete the current operation.

Precondition

An operation must be running while the function is invoked.

Note

No more than one thread can wait on a DAC driver using this function.

Parameters

in	*dacp*	pointer to the DACDriver object

Function Class:

Not an API, this function is for internal use only.

7.3.2.4 #define _dac_reset_i(*dacp*) osalThreadResumeI(&(dacp)->thread, MSG_RESET)

Resumes a thread waiting for a conversion completion.

Parameters

in	*dacp*	pointer to the DACDriver object

Function Class:

Not an API, this function is for internal use only.

7.3.2.5 #define _dac_reset_s(*dacp*) osalThreadResumeS(&(dacp)->thread, MSG_RESET)

Resumes a thread waiting for a conversion completion.

Parameters

in	*dacp*	pointer to the DACDriver object

Function Class:

Not an API, this function is for internal use only.

7.3.2.6 #define _dac_wakeup_isr(*dacp*)

Value:

```
{                                                    \
  osalSysLockFromISR();                                                    \
  osalThreadResumeI(&(dacp)->thread, MSG_OK);                              \
  osalSysUnlockFromISR();                                                  \
}
```

Wakes up the waiting thread.

Parameters

in	*dacp*	pointer to the DACDriver object

Function Class:

Not an API, this function is for internal use only.

7.3.2.7 #define _dac_timeout_isr(*dacp*)

Value:

```
{                                               \
  osalSysLockFromISR();                                                \
  osalThreadResumeI(&(dacp)->thread, MSG_TIMEOUT);                     \
  osalSysUnlockFromISR();                                              \
}
```

Wakes up the waiting thread with a timeout message.

Parameters

in	*dacp*	pointer to the DACDriver object

Function Class:

Not an API, this function is for internal use only.

7.3.2.8 #define _dac_isr_half_code(*dacp*)

Value:

```
{                                             \
  if ((dacp)->grpp->end_cb != NULL) {                                  \
    (dacp)->grpp->end_cb(dacp, (dacp)->samples, (dacp)->depth / 2);    \
  }                                                                    \
}
```

Common ISR code, half buffer event.

This code handles the portable part of the ISR code:

- Callback invocation.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

in	*dacp*	pointer to the DACDriver object

Function Class:

Not an API, this function is for internal use only.

7.3.2.9 #define _dac_isr_full_code(*dacp*)

Value:

```
{                                             \
  if ((dacp)->grpp->end_cb != NULL) {                                  \
    if ((dacp)->depth > 1) {                                           \
      /* Invokes the callback passing the 2nd half of the buffer.*/    \
      size_t half = (dacp)->depth / 2;                                 \
      size_t half_index = half * (dacp)->grpp->num_channels;           \
      (dacp)->grpp->end_cb(dacp, (dacp)->samples + half_index, half);  \
    }                                                                  \
    else {                                                             \
      /* Invokes the callback passing the whole buffer.*/              \
      (dacp)->grpp->end_cb(dacp, (dacp)->samples, (dacp)->depth);      \
    }                                                                  \
  }                                                                    \
}
```

Common ISR code, full buffer event.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

in	*dacp*	pointer to the `DACDriver` object

Function Class:

Not an API, this function is for internal use only.

7.3.2.10 #define _dac_isr_error_code(*dacp, err*)

Value:

```
{                                                  \
  dac_lld_stop_conversion(dacp);                                       \
  if ((dacp)->grpp->error_cb != NULL) {                                \
    (dacp)->state = DAC_ERROR;                                         \
    (dacp)->grpp->error_cb(dacp, err);                                 \
    if ((dacp)->state == DAC_ERROR)                                    \
      (dacp)->state = DAC_READY;                                       \
  }                                                                    \
  (dacp)->grpp = NULL;                                                 \
  _dac_timeout_isr(dacp);                                              \
}
```

Common ISR code, error event.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread timeout signaling, if any.
- Driver state transitions.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

in	*dacp*	pointer to the `DACDriver` object
in	*err*	platform dependent error code

Function Class:

Not an API, this function is for internal use only.

7.3.2.11 #define DAC_MAX_CHANNELS 2

Maximum number of DAC channels per unit.

7.3.2.12 #define PLATFORM_DAC_USE_DAC1 FALSE

DAC1 CH1 driver enable switch.

If set to `TRUE` the support for DAC1 channel 1 is included.

Note

The default is `FALSE`.

7.3.3 Typedef Documentation

7.3.3.1 typedef uint32_t **dacchannel_t**

Type of a DAC channel index.

7.3.3.2 typedef struct **DACDriver DACDriver**

Type of a structure representing an DAC driver.

7.3.3.3 typedef uint16_t **dacsample_t**

Type representing a DAC sample.

7.3.3.4 typedef void(∗ daccallback_t) (**DACDriver** ∗dacp, **dacsample_t** ∗buffer, size_t n)

DAC notification callback type.

Parameters

in	*dacp*	pointer to the `DACDriver` object triggering the
in	*buffer*	pointer to the next semi-buffer to be filled
in	*n*	number of buffer rows available starting from `buffer` callback

7.3.3.5 typedef void(∗ dacerrorcallback_t) (**DACDriver** ∗dacp, **dacerror_t** err)

ADC error callback type.

Parameters

`in`	*dacp*	pointer to the `DACDriver` object triggering the callback
`in`	*err*	ADC error code

7.3.4 Enumeration Type Documentation

7.3.4.1 enum dacstate_t

Driver state machine possible states.

Enumerator

DAC_UNINIT Not initialized.

DAC_STOP Stopped.

DAC_READY Ready.

DAC_ACTIVE Exchanging data.

DAC_COMPLETE Asynchronous operation complete.

DAC_ERROR Error.

7.3.4.2 enum dacerror_t

Possible DAC failure causes.

Note

Error codes are architecture dependent and should not relied upon.

Enumerator

DAC_ERR_DMAFAILURE DMA operations failure.

DAC_ERR_UNDERFLOW DAC overflow condition.

7.3.5 Function Documentation

7.3.5.1 void dacInit (void)

DAC Driver initialization.

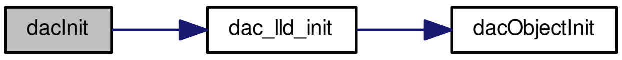
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



7.3.5.2 void dacObjectInit (**DACDriver** ∗ *dacp*)

Initializes the standard part of a `DACDriver` structure.

Parameters

`out`	*dacp*	pointer to the `DACDriver` object

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.3.5.3 void dacStart (**DACDriver** ∗ *dacp*, const **DACConfig** ∗ *config*)

Configures and activates the DAC peripheral.

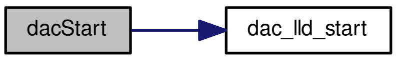
Parameters

`in`	*dacp*	pointer to the `DACDriver` object
`in`	*config*	pointer to the `DACConfig` object, it can be `NULL` if the low level driver implementation supports a default configuration

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.3.5.4 void dacStop (**DACDriver** ∗ *dacp*)

Deactivates the DAC peripheral.

Note

Deactivating the peripheral also enforces a release of the slave select line.

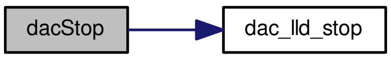
Parameters

`in`	*dacp*	pointer to the `DACDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.3.5.5 void dacPutChannelX (**DACDriver** ∗ *dacp,* **dacchannel_t** *channel,* **dacsample_t** *sample*)

Outputs a value directly on a DAC channel.

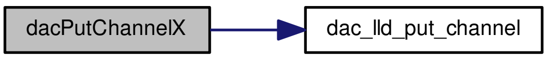
Parameters

`in`	*dacp*	pointer to the `DACDriver` object
`in`	*channel*	DAC channel number
`in`	*sample*	value to be output

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

Here is the call graph for this function:



7.3.5.6 void dacStartConversion (**DACDriver** ∗ *dacp,* const **DACConversionGroup** ∗ *grpp,* **dacsample_t** ∗ *samples,* size_t *depth*)

Starts a DAC conversion.

Starts an asynchronous conversion operation.

Note

The buffer is organized as a matrix of M∗N elements where M is the channels number configured into the conversion group and N is the buffer depth. The samples are sequentially written into the buffer with no gaps.

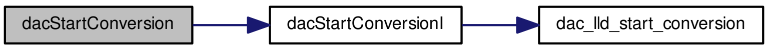
Parameters

`in`	*dacp*	pointer to the `DACDriver` object
`in`	*grpp*	pointer to a `DACConversionGroup` object
`in`	*samples*	pointer to the samples buffer
`in`	*depth*	buffer depth (matrix rows number). The buffer depth must be one or an even number.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.3.5.7 void dacStartConversionI (**DACDriver** ∗ *dacp,* const **DACConversionGroup** ∗ *grpp,* **dacsample_t** ∗ *samples,* size_t *depth*)

Starts a DAC conversion.

Starts an asynchronous conversion operation.

Postcondition

The callbacks associated to the conversion group will be invoked on buffer fill and error events.

Note

The buffer is organized as a matrix of M∗N elements where M is the channels number configured into the conversion group and N is the buffer depth. The samples are sequentially written into the buffer with no gaps.

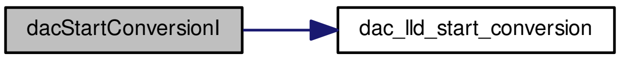
Parameters

`in`	*dacp*	pointer to the `DACDriver` object
`in`	*grpp*	pointer to a `DACConversionGroup` object
`in`	*samples*	pointer to the samples buffer
`in`	*depth*	buffer depth (matrix rows number). The buffer depth must be one or an even number.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



7.3.5.8 void dacStopConversion (**DACDriver** * *dacp*)

Stops an ongoing conversion.

This function stops the currently ongoing conversion and returns the driver in the `DAC_READY` state. If there was no conversion being processed then the function does nothing.

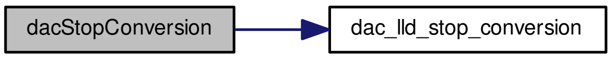
Parameters

`in`	*dacp*	pointer to the `DACDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.3.5.9 void dacStopConversionI (**DACDriver** * *dacp*)

Stops an ongoing conversion.

This function stops the currently ongoing conversion and returns the driver in the `DAC_READY` state. If there was no conversion being processed then the function does nothing.

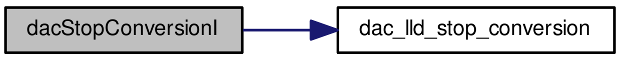
Parameters

`in`	*dacp*	pointer to the `DACDriver` object

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



7.3.5.10 msg_t dacConvert (**DACDriver** ∗ *dacp,* const **DACConversionGroup** ∗ *grpp,* **dacsample_t** ∗ *samples,* size_t *depth*)

Performs a DAC conversion.

Performs a synchronous conversion operation.

Note

The buffer is organized as a matrix of M∗N elements where M is the channels number configured into the conversion group and N is the buffer depth. The samples are sequentially written into the buffer with no gaps.

Parameters

`in`	*dacp*	pointer to the `DACDriver` object
`in`	*grpp*	pointer to a `DACConversionGroup` object
`out`	*samples*	pointer to the samples buffer
`in`	*depth*	buffer depth (matrix rows number). The buffer depth must be one or an even number.

Returns

The operation result.

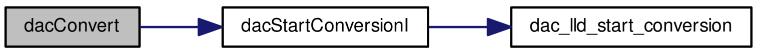
Return values

MSG_OK	Conversion finished.
MSG_RESET	The conversion has been stopped using `acdStopConversion()` or `acdStopConversionI()`, the result buffer may contain incorrect data.
MSG_TIMEOUT	The conversion has been stopped because an hardware error.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.3.5.11 void dacAcquireBus (DACDriver ∗ *dacp*)

Gains exclusive access to the DAC bus.

This function tries to gain ownership to the DAC bus, if the bus is already being used then the invoking thread is queued.

Precondition

In order to use this function the option `DAC_USE_MUTUAL_EXCLUSION` must be enabled.

Parameters

in	*dacp*	pointer to the `DACDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.3.5.12 void dacReleaseBus (DACDriver ∗ *dacp*)

Releases exclusive access to the DAC bus.

Precondition

In order to use this function the option `DAC_USE_MUTUAL_EXCLUSION` must be enabled.

Parameters

in	*dacp*	pointer to the `DACDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

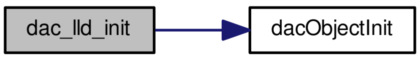
7.3.5.13 void dac_lld_init (void)

Low level DAC driver initialization.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.3.5.14 void dac_lld_start (**DACDriver** ∗ *dacp*)

Configures and activates the DAC peripheral.

Parameters

in	*dacp*	pointer to the `DACDriver` object

Function Class:

Not an API, this function is for internal use only.

7.3.5.15 void dac_lld_stop (**DACDriver** ∗ *dacp*)

Deactivates the DAC peripheral.

Parameters

in	*dacp*	pointer to the `DACDriver` object

Function Class:

Not an API, this function is for internal use only.

7.3.5.16 void dac_lld_put_channel (**DACDriver** ∗ *dacp*, **dacchannel_t** *channel*, **dacsample_t** *sample*)

Outputs a value directly on a DAC channel.

Parameters

in	*dacp*	pointer to the `DACDriver` object
in	*channel*	DAC channel number
in	*sample*	value to be output

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.3.5.17 void dac_lld_start_conversion (**DACDriver** ∗ *dacp*)

Starts a DAC conversion.

Starts an asynchronous conversion operation.

Note

In `DAC_DHRM_8BIT_RIGHT` mode the parameters passed to the callback are wrong because two samples are packed in a single dacsample_t element. This will not be corrected, do not rely on those parameters.
In `DAC_DHRM_8BIT_RIGHT_DUAL` mode two samples are treated as a single 16 bits sample and packed into a single dacsample_t element. The num_channels must be set to one in the group conversion configuration structure.

Parameters

`in`	*dacp*	pointer to the `DACDriver` object

Function Class:

Not an API, this function is for internal use only.

7.3.5.18 void dac_lld_stop_conversion (**DACDriver** ∗ *dacp*)

Stops an ongoing conversion.

This function stops the currently ongoing conversion and returns the driver in the `DAC_READY` state. If there was no conversion being processed then the function does nothing.

Parameters

`in`	*dacp*	pointer to the `DACDriver` object

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.3.6 Variable Documentation

7.3.6.1 DACDriver DACD1

DAC1 driver identifier.

7.4 EXT Driver

Generic EXT Driver.

7.4.1 Detailed Description

Generic EXT Driver.

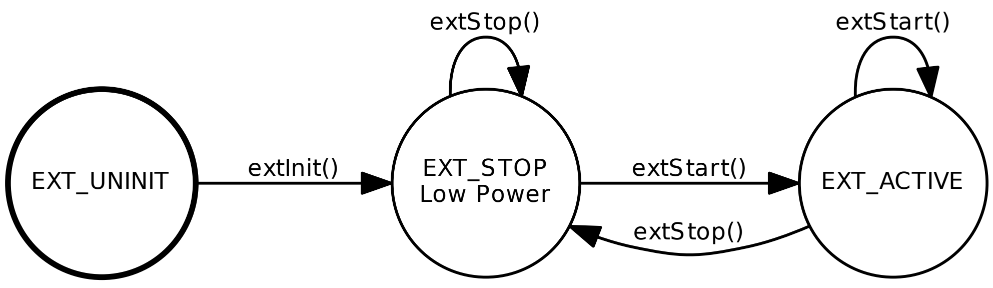
This module implements a generic EXT (EXTernal) driver.

Precondition

In order to use the EXT driver the `HAL_USE_EXT` option must be enabled in `halconf.h`.

7.4.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



7.4.3 EXT Operations.

This driver abstracts generic external interrupt sources, a callback is invoked when a programmable transition is detected on one of the configured channels. Several channel modes are possible.

- **EXT_CH_MODE_DISABLED**, channel not used.
- **EXT_CH_MODE_RISING_EDGE**, callback on a rising edge.
- **EXT_CH_MODE_FALLING_EDGE**, callback on a falling edge.
- **EXT_CH_MODE_BOTH_EDGES**, callback on a both edges.

Macros

- #define EXT_MAX_CHANNELS 20

  Available number of EXT channels.

EXT channel modes

- #define EXT_CH_MODE_EDGES_MASK 3U

  Mask of edges field.
- #define EXT_CH_MODE_DISABLED 0U

  Channel disabled.
- #define EXT_CH_MODE_RISING_EDGE 1U

  Rising edge callback.
- #define EXT_CH_MODE_FALLING_EDGE 2U

  Falling edge callback.
- #define EXT_CH_MODE_BOTH_EDGES 3U

  Both edges callback.
- #define EXT_CH_MODE_LOW_LEVEL 5U

  low level callback.
- #define EXT_CH_MODE_AUTOSTART 4U

  Channel started automatically on driver start.

Macro Functions

- #define extChannelEnableI(extp, channel) ext_lld_channel_enable(extp, channel)

  Enables an EXT channel.
- #define extChannelDisableI(extp, channel) ext_lld_channel_disable(extp, channel)

  Disables an EXT channel.
- #define extSetChannelMode(extp, channel, extcp)

  Changes the operation mode of a channel.

PLATFORM configuration options

- #define PLATFORM_EXT_USE_EXT1 FALSE

  EXT driver enable switch.

Typedefs

- typedef struct EXTDriver EXTDriver

  Type of a structure representing a EXT driver.
- typedef uint32_t expchannel_t

  EXT channel identifier.
- typedef void(∗ extcallback_t) (EXTDriver ∗extp, expchannel_t channel)

  Type of an EXT generic notification callback.

Data Structures

- struct EXTChannelConfig

  Channel configuration structure.
- struct EXTConfig

  Driver configuration structure.
- struct EXTDriver

  Structure representing an EXT driver.

Functions

- void extInit (void)

  EXT Driver initialization.
- void extObjectInit (EXTDriver ∗extp)

  Initializes the standard part of a `EXTDriver` structure.
- void extStart (EXTDriver ∗extp, const EXTConfig ∗config)

  Configures and activates the EXT peripheral.
- void extStop (EXTDriver ∗extp)

  Deactivates the EXT peripheral.
- void extChannelEnable (EXTDriver ∗extp, expchannel_t channel)

  Enables an EXT channel.
- void extChannelDisable (EXTDriver ∗extp, expchannel_t channel)

  Disables an EXT channel.
- void extSetChannelModeI (EXTDriver ∗extp, expchannel_t channel, const EXTChannelConfig ∗extcp)

  Changes the operation mode of a channel.
- void ext_lld_init (void)

  Low level EXT driver initialization.
- void ext_lld_start (EXTDriver ∗extp)

  Configures and activates the EXT peripheral.
- void ext_lld_stop (EXTDriver ∗extp)

  Deactivates the EXT peripheral.
- void ext_lld_channel_enable (EXTDriver ∗extp, expchannel_t channel)

  Enables an EXT channel.
- void ext_lld_channel_disable (EXTDriver ∗extp, expchannel_t channel)

  Disables an EXT channel.

Enumerations

Variables

- EXTDriver EXTD1

  EXT1 driver identifier.

7.4.4 Macro Definition Documentation

7.4.4.1 #define EXT_CH_MODE_EDGES_MASK 3U

Mask of edges field.

7.4.4.2 #define EXT_CH_MODE_DISABLED 0U

Channel disabled.

7.4.4.3 #define EXT_CH_MODE_RISING_EDGE 1U

Rising edge callback.

7.4.4.4 #define EXT_CH_MODE_FALLING_EDGE 2U

Falling edge callback.

7.4.4.5 #define EXT_CH_MODE_BOTH_EDGES 3U

Both edges callback.

7.4.4.6 #define EXT_CH_MODE_LOW_LEVEL 5U

low level callback.

7.4.4.7 #define EXT_CH_MODE_AUTOSTART 4U

Channel started automatically on driver start.

7.4.4.8 #define extChannelEnableI(*extp, channel*) ext_lld_channel_enable(extp, channel)

Enables an EXT channel.

Parameters

in	*extp*	pointer to the EXTDriver object
in	*channel*	channel to be enabled

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.4.4.9 #define extChannelDisableI(*extp, channel*) ext_lld_channel_disable(extp, channel)

Disables an EXT channel.

Parameters

in	*extp*	pointer to the EXTDriver object
in	*channel*	channel to be disabled

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.4.4.10 #define extSetChannelMode(*extp, channel, extcp*)

Value:

```
{                                \
  osalSysLock();                                                          \
  extSetChannelModeI(extp, channel, extcp);                               \
  osalSysUnlock();                                                        \
}
```

Changes the operation mode of a channel.

Note

This function attempts to write over the current configuration structure that must have been not declared constant. This violates the `const` qualifier in `extStart()` but it is intentional. This function cannot be used if the configuration structure is declared `const`.

Parameters

in	*extp*	pointer to the `EXTDriver` object
in	*channel*	channel to be changed
in	*extcp*	new configuration for the channel

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.4.4.11 #define EXT_MAX_CHANNELS 20

Available number of EXT channels.

7.4.4.12 #define PLATFORM_EXT_USE_EXT1 FALSE

EXT driver enable switch.

If set to `TRUE` the support for EXT1 is included.

Note

The default is `FALSE`.

7.4.5 Typedef Documentation

7.4.5.1 typedef struct EXTDriver EXTDriver

Type of a structure representing a EXT driver.

7.4.5.2 typedef uint32_t expchannel_t

EXT channel identifier.

7.4.5.3 typedef void(∗ extcallback_t) (EXTDriver ∗extp, expchannel_t channel)

Type of an EXT generic notification callback.

Parameters

in	*extp*	pointer to the `EXPDriver` object triggering the callback

7.4.6 Enumeration Type Documentation

7.4.6.1 enum **extstate_t**

Driver state machine possible states.

Enumerator

EXT_UNINIT	Not initialized.
EXT_STOP	Stopped.
EXT_ACTIVE	Active.

7.4.7 Function Documentation

7.4.7.1 void extInit (void)

EXT Driver initialization.

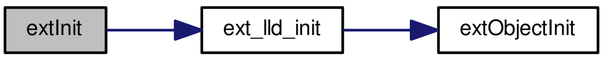
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



7.4.7.2 void extObjectInit (**EXTDriver** ∗ *extp*)

Initializes the standard part of a `EXTDriver` structure.

Parameters

`out`	*extp*	pointer to the `EXTDriver` object

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.4.7.3 void extStart (**EXTDriver** ∗ *extp,* const **EXTConfig** ∗ *config*)

Configures and activates the EXT peripheral.

Postcondition

After activation all EXT channels are in the disabled state, use `extChannelEnable()` in order to activate them.

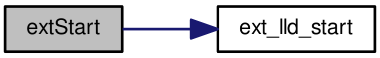
Parameters

`in`	*extp*	pointer to the `EXTDriver` object
`in`	*config*	pointer to the `EXTConfig` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.4.7.4 void extStop (**EXTDriver** ∗ *extp*)

Deactivates the EXT peripheral.

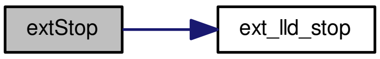
Parameters

`in`	*extp*	pointer to the `EXTDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.4.7.5 void extChannelEnable (**EXTDriver** ∗ *extp,* **expchannel_t** *channel*)

Enables an EXT channel.

Precondition

The channel must not be in `EXT_CH_MODE_DISABLED` mode.

Parameters

`in`	*extp*	pointer to the `EXTDriver` object
`in`	*channel*	channel to be enabled

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.4.7.6 void extChannelDisable (**EXTDriver** ∗ *extp,* **expchannel_t** *channel*)

Disables an EXT channel.

Precondition

The channel must not be in `EXT_CH_MODE_DISABLED` mode.

Parameters

`in`	*extp*	pointer to the `EXTDriver` object
`in`	*channel*	channel to be disabled

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.4.7.7 void extSetChannelModeI (**EXTDriver** ∗ *extp,* **expchannel_t** *channel,* const **EXTChannelConfig** ∗ *extcp*)

Changes the operation mode of a channel.

Note

This function attempts to write over the current configuration structure that must have been not declared constant. This violates the `const` qualifier in `extStart()` but it is intentional.
This function cannot be used if the configuration structure is declared `const`.
The effect of this function on constant configuration structures is not defined.

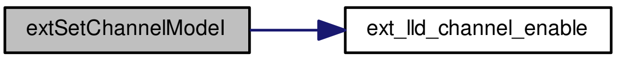
Parameters

`in`	*extp*	pointer to the `EXTDriver` object
`in`	*channel*	channel to be changed
`in`	*extcp*	new configuration for the channel

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



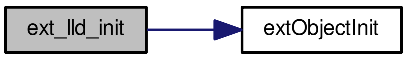
7.4.7.8 void ext_lld_init (void)

Low level EXT driver initialization.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.4.7.9 void ext_lld_start (EXTDriver * *extp*)

Configures and activates the EXT peripheral.

Parameters

in	*extp*	pointer to the `EXTDriver` object

Function Class:

Not an API, this function is for internal use only.

7.4.7.10 void ext_lld_stop (EXTDriver * *extp*)

Deactivates the EXT peripheral.

Parameters

in	*extp*	pointer to the `EXTDriver` object

Function Class:

Not an API, this function is for internal use only.

7.4.7.11 void ext_lld_channel_enable (EXTDriver ∗ *extp*, expchannel_t *channel*)

Enables an EXT channel.

Parameters

in	*extp*	pointer to the `EXTDriver` object
in	*channel*	channel to be enabled

Function Class:

Not an API, this function is for internal use only.

7.4.7.12 void ext_lld_channel_disable (EXTDriver ∗ *extp*, expchannel_t *channel*)

Disables an EXT channel.

Parameters

in	*extp*	pointer to the `EXTDriver` object
in	*channel*	channel to be disabled

Function Class:

Not an API, this function is for internal use only.

7.4.8 Variable Documentation

7.4.8.1 EXTDriver EXTD1

EXT1 driver identifier.

7.5 Abstract NOR Flash Class

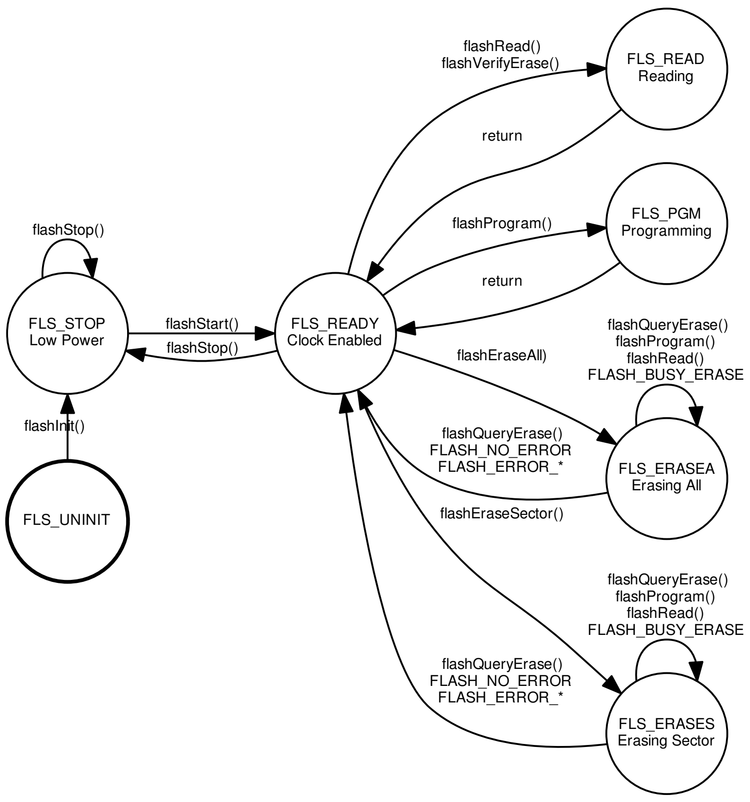
Generic NOR Flash interface.

Generic NOR Flash interface.

This module implements a generic class for NOR Flash devices.

7.5.1 Driver State Machine

The flash driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



7.5.2 Flash Operations.

This driver abstracts a generic PWM timer composed of:

- A clock prescaler.
- A main up counter.
- A comparator register that resets the main counter to zero when the limit is reached. An optional callback can be generated when this happens.
- An array of `PWM_CHANNELS` PWM channels, each channel has an output, a comparator and is able to invoke an optional callback when a comparator match with the main counter happens.

A PWM channel output can be in two different states:

- **IDLE**, when the channel is disabled or after a match occurred.
- **ACTIVE**, when the channel is enabled and a match didn't occur yet in the current PWM cycle.

Note that the two states can be associated to both logical zero or one in the `PWMChannelConfig` structure.

7.6 GPT Driver

Generic GPT Driver.

7.6.1 Detailed Description

Generic GPT Driver.

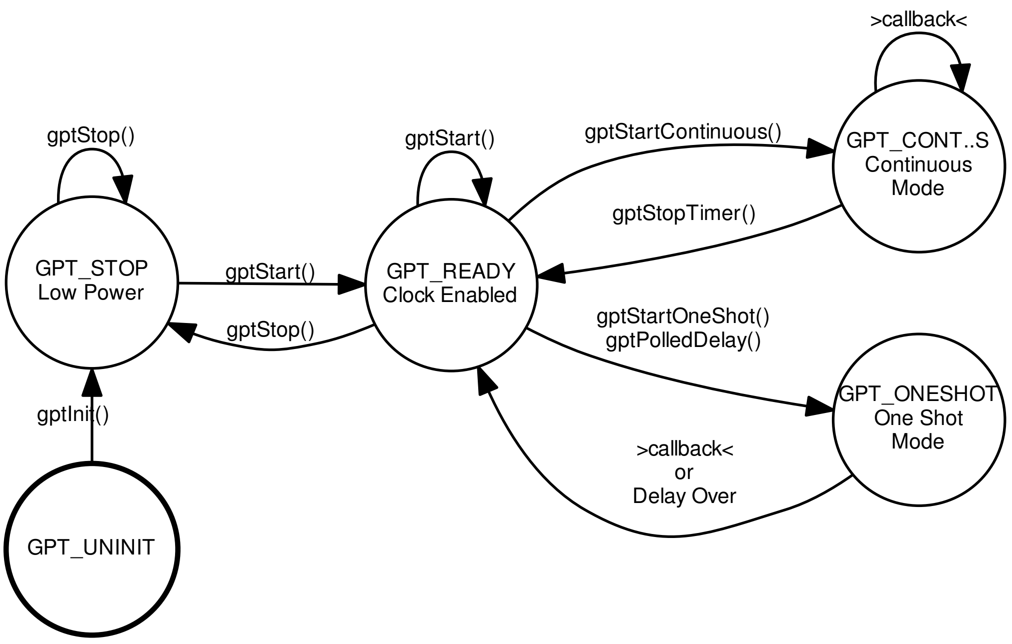
This module implements a generic GPT (General Purpose Timer) driver. The timer can be programmed in order to trigger callbacks after a specified time period or continuously with a specified interval.

Precondition

In order to use the GPT driver the `HAL_USE_GPT` option must be enabled in `halconf.h`.

7.6.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



7.6.3 GPT Operations.

This driver abstracts a generic timer composed of:

- A clock prescaler.
- A main up counter.

- A comparator register that resets the main counter to zero when the limit is reached. A callback is invoked when this happens.

The timer can operate in three different modes:

- **Continuous Mode**, a periodic callback is invoked until the driver is explicitly stopped.
- **One Shot Mode**, a callback is invoked after the programmed period and then the timer automatically stops.
- **Delay Mode**, the timer is used for inserting a brief delay into the execution flow, no callback is invoked in this mode.

Macros

- #define gptChangeIntervalI(gptp, interval)

  Changes the interval of GPT peripheral.
- #define gptGetIntervalX(gptp) gpt_lld_get_interval(gptp)

  Returns the interval of GPT peripheral.
- #define gptGetCounterX(gptp) gpt_lld_get_counter(gptp)

  Returns the counter value of GPT peripheral.
- #define gpt_lld_change_interval(gptp, interval)

  Changes the interval of GPT peripheral.

PLATFORM configuration options

- #define PLATFORM_GPT_USE_GPT1 FALSE

  GPTD1 driver enable switch.

Typedefs

- typedef struct GPTDriver GPTDriver

  Type of a structure representing a GPT driver.
- typedef void(∗ gptcallback_t) (GPTDriver ∗gptp)

  GPT notification callback type.
- typedef uint32_t gptfreq_t

  GPT frequency type.
- typedef uint16_t gptcnt_t

  GPT counter type.

Data Structures

- struct GPTConfig

  Driver configuration structure.
- struct GPTDriver

  Structure representing a GPT driver.

Functions

- void gptInit (void)

  GPT Driver initialization.
- void gptObjectInit (GPTDriver ∗gptp)

  Initializes the standard part of a `GPTDriver` structure.
- void gptStart (GPTDriver ∗gptp, const GPTConfig ∗config)

  Configures and activates the GPT peripheral.
- void gptStop (GPTDriver ∗gptp)

  Deactivates the GPT peripheral.
- void gptChangeInterval (GPTDriver ∗gptp, gptcnt_t interval)

  Changes the interval of GPT peripheral.
- void gptStartContinuous (GPTDriver ∗gptp, gptcnt_t interval)

  Starts the timer in continuous mode.
- void gptStartContinuousI (GPTDriver ∗gptp, gptcnt_t interval)

  Starts the timer in continuous mode.
- void gptStartOneShot (GPTDriver ∗gptp, gptcnt_t interval)

  Starts the timer in one shot mode.
- void gptStartOneShotI (GPTDriver ∗gptp, gptcnt_t interval)

  Starts the timer in one shot mode.
- void gptStopTimer (GPTDriver ∗gptp)

  Stops the timer.
- void gptStopTimerI (GPTDriver ∗gptp)

  Stops the timer.
- void gptPolledDelay (GPTDriver ∗gptp, gptcnt_t interval)

  Starts the timer in one shot mode and waits for completion.
- void gpt_lld_init (void)

  Low level GPT driver initialization.
- void gpt_lld_start (GPTDriver ∗gptp)

  Configures and activates the GPT peripheral.
- void gpt_lld_stop (GPTDriver ∗gptp)

  Deactivates the GPT peripheral.
- void gpt_lld_start_timer (GPTDriver ∗gptp, gptcnt_t interval)

  Starts the timer in continuous mode.
- void gpt_lld_stop_timer (GPTDriver ∗gptp)

  Stops the timer.
- void gpt_lld_polled_delay (GPTDriver ∗gptp, gptcnt_t interval)

  Starts the timer in one shot mode and waits for completion.

Enumerations

Variables

- GPTDriver GPTD1

  GPTD1 driver identifier.

7.6.4 Macro Definition Documentation

7.6.4.1 #define gptChangeIntervalI(*gptp,* *interval*)

Value:

```
{                                    \
  gpt_lld_change_interval(gptp, interval);                                   \
}
```

Changes the interval of GPT peripheral.

This function changes the interval of a running GPT unit.

Precondition

The GPT unit must be running in continuous mode.

Postcondition

The GPT unit interval is changed to the new value.

Parameters

in	*gptp*	pointer to a `GPTDriver` object
in	*interval*	new cycle time in timer ticks

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.6.4.2 #define gptGetIntervalX(*gptp*) gpt_lld_get_interval(gptp)

Returns the interval of GPT peripheral.

Precondition

The GPT unit must be running in continuous mode.

Parameters

in	*gptp*	pointer to a `GPTDriver` object

Returns

The current interval.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

7.6.4.3 #define gptGetCounterX(*gptp*) gpt_lld_get_counter(gptp)

Returns the counter value of GPT peripheral.

Precondition

The GPT unit must be running in continuous mode.

Note

The nature of the counter is not defined, it may count upward or downward, it could be continuously running or not.

Parameters

in	*gptp*	pointer to a `GPTDriver` object

Returns

The current counter value.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

7.6.4.4 #define PLATFORM_GPT_USE_GPT1 FALSE

GPTD1 driver enable switch.

If set to `TRUE` the support for GPTD1 is included.

Note

The default is `FALSE`.

7.6.4.5 #define gpt_lld_change_interval(*gptp*, *interval*)

Value:

```
{                                      \
  (void)gptp;                                                                   \
  (void)interval;                                                               \
}
```

Changes the interval of GPT peripheral.

This function changes the interval of a running GPT unit.

Precondition

The GPT unit must have been activated using `gptStart()`.
The GPT unit must have been running in continuous mode using `gptStartContinuous()`.

Postcondition

The GPT unit interval is changed to the new value.

Note

The function has effect at the next cycle start.

Parameters

in	*gptp*	pointer to a `GPTDriver` object
in	*interval*	new cycle time in timer ticks

Function Class:

Not an API, this function is for internal use only.

7.6.5 Typedef Documentation

7.6.5.1 typedef struct GPTDriver GPTDriver

Type of a structure representing a GPT driver.

7.6.5.2 typedef void(∗ gptcallback_t) (GPTDriver ∗gptp)

GPT notification callback type.

Parameters

in	*gptp*	pointer to a `GPTDriver` object

7.6.5.3 typedef uint32_t gptfreq_t

GPT frequency type.

7.6.5.4 typedef uint16_t gptcnt_t

GPT counter type.

7.6.6 Enumeration Type Documentation

7.6.6.1 enum gptstate_t

Driver state machine possible states.

Enumerator

GPT_UNINIT	Not initialized.
GPT_STOP	Stopped.
GPT_READY	Ready.
GPT_CONTINUOUS	Active in continuous mode.
GPT_ONESHOT	Active in one shot mode.

7.6.7 Function Documentation

7.6.7.1 void gptInit (void)

GPT Driver initialization.

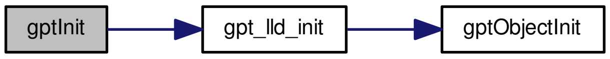
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



7.6.7.2 void gptObjectInit (GPTDriver ∗ *gptp*)

Initializes the standard part of a `GPTDriver` structure.

Parameters

`out`	*gptp*	pointer to the `GPTDriver` object

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.6.7.3 void gptStart (GPTDriver ∗ *gptp*, const GPTConfig ∗ *config*)

Configures and activates the GPT peripheral.

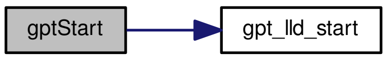
Parameters

`in`	*gptp*	pointer to the `GPTDriver` object
`in`	*config*	pointer to the `GPTConfig` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.6.7.4 void gptStop (GPTDriver * *gptp*)

Deactivates the GPT peripheral.

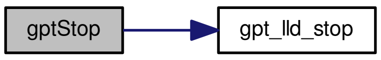
Parameters

in	*gptp*	pointer to the GPTDriver object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.6.7.5 void gptChangeInterval (GPTDriver * *gptp*, gptcnt_t *interval*)

Changes the interval of GPT peripheral.

This function changes the interval of a running GPT unit.

Precondition

The GPT unit must be running in continuous mode.

Postcondition

The GPT unit interval is changed to the new value.

Parameters

in	*gptp*	pointer to a GPTDriver object
in	*interval*	new cycle time in timer ticks

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.6.7.6 void gptStartContinuous (GPTDriver * *gptp*, gptcnt_t *interval*)

Starts the timer in continuous mode.

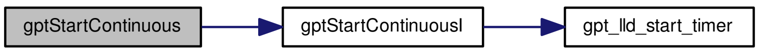
Parameters

in	*gptp*	pointer to the `GPTDriver` object
in	*interval*	period in ticks

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.6.7.7 void gptStartContinuousI (GPTDriver * *gptp*, gptcnt_t *interval*)

Starts the timer in continuous mode.

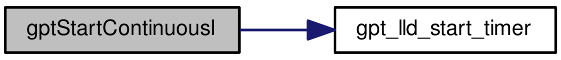
Parameters

in	*gptp*	pointer to the `GPTDriver` object
in	*interval*	period in ticks

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



7.6.7.8 void gptStartOneShot (GPTDriver ∗ *gptp*, gptcnt_t *interval*)

Starts the timer in one shot mode.

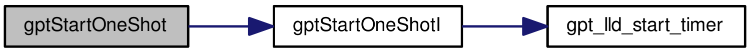
Parameters

in	*gptp*	pointer to the `GPTDriver` object
in	*interval*	time interval in ticks

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.6.7.9 void gptStartOneShotI (GPTDriver ∗ *gptp*, gptcnt_t *interval*)

Starts the timer in one shot mode.

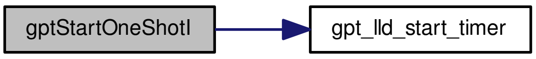
Parameters

in	*gptp*	pointer to the `GPTDriver` object
in	*interval*	time interval in ticks

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.6.7.10 void gptStopTimer (GPTDriver ∗ *gptp*)

Stops the timer.

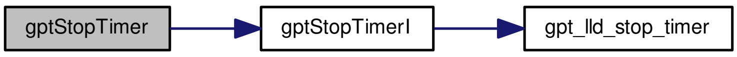
Parameters

in	*gptp*	pointer to the GPTDriver object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.6.7.11 void gptStopTimerI (GPTDriver ∗ *gptp*)

Stops the timer.

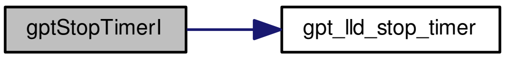
Parameters

in	*gptp*	pointer to the GPTDriver object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.6.7.12 void gptPolledDelay (GPTDriver ∗ *gptp,* gptcnt_t *interval*)

Starts the timer in one shot mode and waits for completion.

This function specifically polls the timer waiting for completion in order to not have extra delays caused by interrupt servicing, this function is only recommended for short delays.

Note

The configured callback is not invoked when using this function.

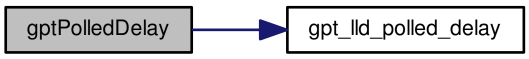
Parameters

in	*gptp*	pointer to the GPTDriver object
in	*interval*	time interval in ticks

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



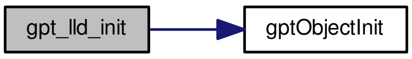
7.6.7.13 void gpt_lld_init (void)

Low level GPT driver initialization.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.6.7.14 void gpt_lld_start (GPTDriver ∗ *gptp*)

Configures and activates the GPT peripheral.

Parameters

in	*gptp*	pointer to the GPTDriver object

Function Class:

Not an API, this function is for internal use only.

7.6.7.15 void gpt_lld_stop (**GPTDriver** ∗ *gptp*)

Deactivates the GPT peripheral.

Parameters

`in`	*gptp*	pointer to the `GPTDriver` object

Function Class:

Not an API, this function is for internal use only.

7.6.7.16 void gpt_lld_start_timer (**GPTDriver** ∗ *gptp*, **gptcnt_t** *interval*)

Starts the timer in continuous mode.

Parameters

`in`	*gptp*	pointer to the `GPTDriver` object
`in`	*interval*	period in ticks

Function Class:

Not an API, this function is for internal use only.

7.6.7.17 void gpt_lld_stop_timer (**GPTDriver** ∗ *gptp*)

Stops the timer.

Parameters

`in`	*gptp*	pointer to the `GPTDriver` object

Function Class:

Not an API, this function is for internal use only.

7.6.7.18 void gpt_lld_polled_delay (**GPTDriver** ∗ *gptp*, **gptcnt_t** *interval*)

Starts the timer in one shot mode and waits for completion.

This function specifically polls the timer waiting for completion in order to not have extra delays caused by interrupt servicing, this function is only recommended for short delays.

Parameters

`in`	*gptp*	pointer to the `GPTDriver` object
`in`	*interval*	time interval in ticks

Function Class:

Not an API, this function is for internal use only.

7.6.8 Variable Documentation

7.6.8.1 GPTDriver GPTD1

GPTD1 driver identifier.

7.7 HAL Driver

Hardware Abstraction Layer.

7.7.1 Detailed Description

Hardware Abstraction Layer.

The HAL (Hardware Abstraction Layer) driver performs the system initialization and includes the platform support code shared by the other drivers. This driver does contain any API function except for a general initialization function `halInit()` that must be invoked before any HAL service can be used, usually the HAL initialization should be performed immediately before the kernel initialization.
Some HAL driver implementations also offer a custom early clock setup function that can be invoked before the C runtime initialization in order to accelerate the startup time.

Macros

- #define _CHIBIOS_HAL_

  ChibiOS/HAL identification macro.
- #define CH_HAL_STABLE 1

  Stable release flag.

ChibiOS/HAL version identification

- #define HAL_VERSION "5.0.0"

  HAL version string.
- #define CH_HAL_MAJOR 5

  HAL version major number.
- #define CH_HAL_MINOR 0

  HAL version minor number.
- #define CH_HAL_PATCH 0

  HAL version patch number.

Return codes

- #define **HAL_SUCCESS** false
- #define **HAL_FAILED** true

Platform identification macros

- #define **PLATFORM_NAME** "templates"

Functions

- void halInit (void)

  HAL initialization.
- void hal_lld_init (void)

  Low level HAL driver initialization.

7.7.2 Macro Definition Documentation

7.7.2.1 #define _CHIBIOS_HAL_

ChibiOS/HAL identification macro.

7.7.2.2 #define CH_HAL_STABLE 1

Stable release flag.

7.7.2.3 #define HAL_VERSION "5.0.0"

HAL version string.

7.7.2.4 #define CH_HAL_MAJOR 5

HAL version major number.

7.7.2.5 #define CH_HAL_MINOR 0

HAL version minor number.

7.7.2.6 #define CH_HAL_PATCH 0

HAL version patch number.

7.7.3 Function Documentation

7.7.3.1 void halInit (void)

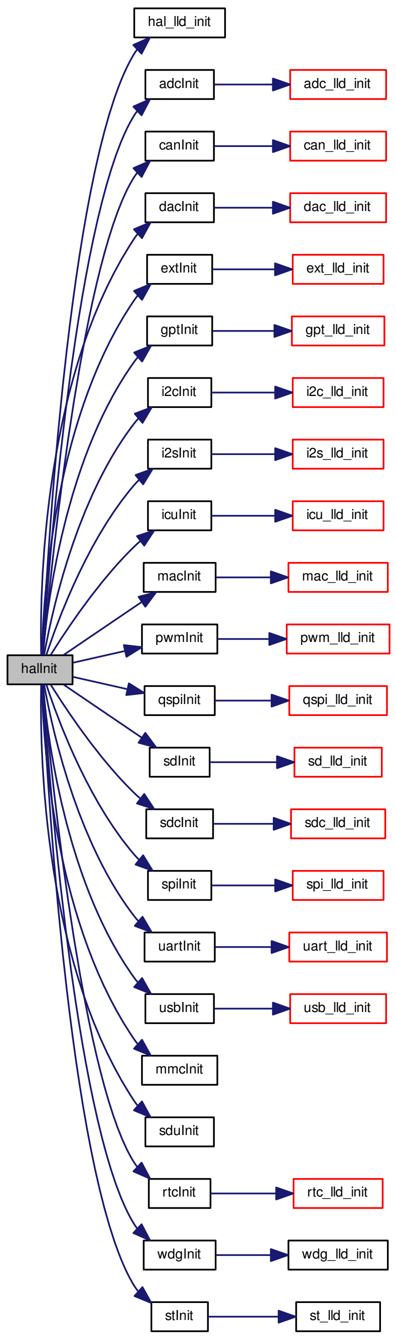
HAL initialization.

This function invokes the low level initialization code then initializes all the drivers enabled in the HAL. Finally the board-specific initialization is performed by invoking `boardInit()` (usually defined in `board.c`).

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



7.7.3.2 void hal_lld_init (void)

Low level HAL driver initialization.

Function Class:

Not an API, this function is for internal use only.

7.8 I/O Buffers Queues

7.8.1 Detailed Description

Buffers Queues are used when there is the need to exchange fixed-length data buffers between ISRs and threads. On the ISR side data can be exchanged only using buffers, on the thread side data can be exchanged both using buffers and/or using an emulation of regular byte queues. There are several kind of buffers queues:

- **Input queue**, unidirectional queue where the writer is the ISR side and the reader is the thread side.
- **Output queue**, unidirectional queue where the writer is the ISR side and the reader is the thread side.
- **Full duplex queue**, bidirectional queue. Full duplex queues are implemented by pairing an input queue and an output queue together.

Macros

- #define BQ_BUFFER_SIZE(n, size) (((size_t)(size) + sizeof (size_t)) ∗ (size_t)(n))

  Computes the size of a buffers queue buffer size.

Macro Functions

- #define bqSizeX(bqp) ((bqp)->bn)

  Returns the queue's number of buffers.
- #define bqSpaceI(bqp) ((bqp)->bcounter)

  Return the ready buffers number.
- #define bqGetLinkX(bqp) ((bqp)->link)

  Returns the queue application-defined link.
- #define bqIsSuspendedX(bqp) ((bqp)->suspended)

  Return the suspended state of the queue.
- #define bqSuspendI(bqp)

  Puts the queue in suspended state.
- #define bqResumeX(bqp)

  Resumes normal queue operations.
- #define ibqIsEmptyI(ibqp) ((bool)(bqSpaceI(ibqp) == 0U))

  Evaluates to `TRUE` if the specified input buffers queue is empty.
- #define ibqIsFullI(ibqp)

  Evaluates to `TRUE` if the specified input buffers queue is full.
- #define obqIsEmptyI(obqp)

  Evaluates to `true` if the specified output buffers queue is empty.
- #define obqIsFullI(obqp) ((bool)(bqSpaceI(obqp) == 0U))

  Evaluates to `true` if the specified output buffers queue is full.

Typedefs

- typedef struct io_buffers_queue io_buffers_queue_t

  Type of a generic queue of buffers.
- typedef void(∗ bqnotify_t) (io_buffers_queue_t ∗bqp)

  Double buffer notification callback type.
- typedef io_buffers_queue_t input_buffers_queue_t

  Type of an input buffers queue.
- typedef io_buffers_queue_t output_buffers_queue_t

  Type of an output buffers queue.

Data Structures

- struct io_buffers_queue

  Structure of a generic buffers queue.

Functions

- void ibqObjectInit (input_buffers_queue_t ∗ibqp, bool suspended, uint8_t ∗bp, size_t size, size_t n, bqnotify←_t infy, void ∗link)

  Initializes an input buffers queue object.
- void ibqResetI (input_buffers_queue_t ∗ibqp)

  Resets an input buffers queue.
- uint8_t ∗ ibqGetEmptyBufferI (input_buffers_queue_t ∗ibqp)

  Gets the next empty buffer from the queue.
- void ibqPostFullBufferI (input_buffers_queue_t ∗ibqp, size_t size)

  Posts a new filled buffer to the queue.
- msg_t ibqGetFullBufferTimeout (input_buffers_queue_t ∗ibqp, systime_t timeout)

  Gets the next filled buffer from the queue.
- msg_t ibqGetFullBufferTimeoutS (input_buffers_queue_t ∗ibqp, systime_t timeout)

  Gets the next filled buffer from the queue.
- void ibqReleaseEmptyBuffer (input_buffers_queue_t ∗ibqp)

  Releases the buffer back in the queue.
- void ibqReleaseEmptyBufferS (input_buffers_queue_t ∗ibqp)

  Releases the buffer back in the queue.
- msg_t ibqGetTimeout (input_buffers_queue_t ∗ibqp, systime_t timeout)

  Input queue read with timeout.
- size_t ibqReadTimeout (input_buffers_queue_t ∗ibqp, uint8_t ∗bp, size_t n, systime_t timeout)

  Input queue read with timeout.
- void obqObjectInit (output_buffers_queue_t ∗obqp, bool suspended, uint8_t ∗bp, size_t size, size_t n, bqnotify_t onfy, void ∗link)

  Initializes an output buffers queue object.
- void obqResetI (output_buffers_queue_t ∗obqp)

  Resets an output buffers queue.
- uint8_t ∗ obqGetFullBufferI (output_buffers_queue_t ∗obqp, size_t ∗sizep)

  Gets the next filled buffer from the queue.
- void obqReleaseEmptyBufferI (output_buffers_queue_t ∗obqp)

  Releases the next filled buffer back in the queue.
- msg_t obqGetEmptyBufferTimeout (output_buffers_queue_t ∗obqp, systime_t timeout)

  Gets the next empty buffer from the queue.
- msg_t obqGetEmptyBufferTimeoutS (output_buffers_queue_t ∗obqp, systime_t timeout)

  Gets the next empty buffer from the queue.
- void obqPostFullBuffer (output_buffers_queue_t ∗obqp, size_t size)

  Posts a new filled buffer to the queue.
- void obqPostFullBufferS (output_buffers_queue_t ∗obqp, size_t size)

  Posts a new filled buffer to the queue.
- msg_t obqPutTimeout (output_buffers_queue_t ∗obqp, uint8_t b, systime_t timeout)

  Output queue write with timeout.
- size_t obqWriteTimeout (output_buffers_queue_t ∗obqp, const uint8_t ∗bp, size_t n, systime_t timeout)

  Output queue write with timeout.
- bool obqTryFlushI (output_buffers_queue_t ∗obqp)

  Flushes the current, partially filled, buffer to the queue.
- void obqFlush (output_buffers_queue_t ∗obqp)

  Flushes the current, partially filled, buffer to the queue.

7.8.2 Macro Definition Documentation

7.8.2.1 #define BQ_BUFFER_SIZE(*n, size*) (((size_t)(size) + sizeof (size_t)) ∗ (size_t)(n))

Computes the size of a buffers queue buffer size.

Parameters

`in`	*n*	number of buffers in the queue
`in`	*size*	size of the buffers

7.8.2.2 #define bqSizeX(*bqp*) ((bqp)->bn)

Returns the queue's number of buffers.

Parameters

`in`	*bqp*	pointer to an `io_buffers_queue_t` structure

Returns

The number of buffers.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

7.8.2.3 #define bqSpaceI(*bqp*) ((bqp)->bcounter)

Return the ready buffers number.

Returns the number of filled buffers if used on an input queue or the number of empty buffers if used on an output queue.

Parameters

`in`	*bqp*	pointer to an `io_buffers_queue_t` structure

Returns

The number of ready buffers.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.8.2.4 #define bqGetLinkX(*bqp*) ((bqp)->link)

Returns the queue application-defined link.

Parameters

in	*bqp*	pointer to an `io_buffers_queue_t` structure

Returns

The application-defined link.

Function Class:

Special function, this function has special requirements see the notes.

7.8.2.5 #define bqIsSuspendedX(*bqp*) ((bqp)->suspended)

Return the suspended state of the queue.

Parameters

in	*bqp*	pointer to an `io_buffers_queue_t` structure

Returns

The suspended state.

Return values

false	if blocking access to the queue is enabled.
true	if blocking access to the queue is suspended.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

7.8.2.6 #define bqSuspendI(*bqp*)

Value:

```
{                                                    \
  (bqp)->suspended = true;                                                   \
  osalThreadDequeueAllI(&(bqp)->waiting, MSG_RESET);                         \
}
```

Puts the queue in suspended state.

When the queue is put in suspended state all waiting threads are woken with message `MSG_RESET` and subsequent attempt at waiting on the queue will result in an immediate return with `MSG_RESET` message.

Note

The content of the queue is not altered, queues can be accessed is suspended state until a blocking operation is met then a `MSG_RESET` occurs.

Parameters

in	*bqp*	pointer to an `io_buffers_queue_t` structure

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.8.2.7 #define bqResumeX(*bqp*)

Value:

```
{                                                               \
  (bqp)->suspended = false;                                                        \
}
```

Resumes normal queue operations.

Parameters

in	*bqp*	pointer to an `io_buffers_queue_t` structure

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

7.8.2.8 #define ibqIsEmptyI(*ibqp*) ((bool)(bqSpaceI(ibqp) == 0U))

Evaluates to `TRUE` if the specified input buffers queue is empty.

Parameters

in	*ibqp*	pointer to an `input_buffers_queue_t` structure

Returns

The queue status.

Return values

false	if the queue is not empty.
true	if the queue is empty.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.8.2.9 #define ibqIsFullI(*ibqp*)

Value:

```
/*lint -save -e9007 [13.5] No side effects, a pointer is passed.*/        \
  ((bool)(((ibqp)->bwrptr == (ibqp)->brdptr) && ((ibqp)->bcounter != 0U)))  \
  /*lint -restore*/
```

Evaluates to `TRUE` if the specified input buffers queue is full.

Parameters

`in`	*ibqp*	pointer to an `input_buffers_queue_t` structure

Returns

The queue status.

Return values

false	if the queue is not full.
true	if the queue is full.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.8.2.10 #define obqIsEmptyI(*obqp*)

Value:

```
/*lint -save -e9007 [13.5] No side effects, a pointer is passed.*/        \
  ((bool)(((obqp)->bwrptr == (obqp)->brdptr) && ((obqp)->bcounter != 0U)))  \
  /*lint -restore*/
```

Evaluates to `true` if the specified output buffers queue is empty.

Parameters

`in`	*obqp*	pointer to an `output_buffers_queue_t` structure

Returns

The queue status.

Return values

false	if the queue is not empty.
true	if the queue is empty.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.8.2.11 #define obqIsFullI(*obqp*) ((bool)(bqSpaceI(obqp) == 0U))

Evaluates to `true` if the specified output buffers queue is full.

Parameters

`in`	*obqp*	pointer to an `output_buffers_queue_t` structure

Returns

The queue status.

Return values

false	if the queue is not full.
true	if the queue is full.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.8.3 Typedef Documentation

7.8.3.1 typedef struct io_buffers_queue io_buffers_queue_t

Type of a generic queue of buffers.

7.8.3.2 typedef void(∗ bqnotify_t) (io_buffers_queue_t ∗bqp)

Double buffer notification callback type.

Parameters

`in`	*iodbp*	the buffers queue pointer

7.8.3.3 typedef io_buffers_queue_t input_buffers_queue_t

Type of an input buffers queue.

7.8.3.4 typedef io_buffers_queue_t output_buffers_queue_t

Type of an output buffers queue.

7.8.4 Function Documentation

7.8.4.1 void ibqObjectInit (**input_buffers_queue_t** ∗ *ibqp*, bool *suspended*, uint8_t ∗ *bp*, size_t *size*, size_t *n*, **bqnotify_t** *infy*, void ∗ *link*)

Initializes an input buffers queue object.

Parameters

`out`	*ibqp*	pointer to the `input_buffers_queue_t` object
`in`	*suspended*	initial state of the queue
`in`	*bp*	pointer to a memory area allocated for buffers
`in`	*size*	buffers size
`in`	*n*	number of buffers
`in`	*infy*	callback called when a buffer is returned to the queue
`in`	*link*	application defined pointer

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.8.4.2 void ibqResetI (**input_buffers_queue_t** ∗ *ibqp*)

Resets an input buffers queue.

All the data in the input buffers queue is erased and lost, any waiting thread is resumed with status `MSG_RESET`.

Note

A reset operation can be used by a low level driver in order to obtain immediate attention from the high level layers.

Parameters

`in`	*ibqp*	pointer to the `input_buffers_queue_t` object

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.8.4.3 uint8_t ∗ ibqGetEmptyBufferI (**input_buffers_queue_t** ∗ *ibqp*)

Gets the next empty buffer from the queue.

Note

The function always returns the same buffer if called repeatedly.

Parameters

`in`	*ibqp*	pointer to the `input_buffers_queue_t` object

Returns

A pointer to the next buffer to be filled.

Return values

NULL	if the queue is full.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.8.4.4 void ibqPostFullBufferI (input_buffers_queue_t * *ibqp*, size_t *size*)

Posts a new filled buffer to the queue.

Parameters

`in`	*ibqp*	pointer to the `input_buffers_queue_t` object
`in`	*size*	used size of the buffer, cannot be zero

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.8.4.5 msg_t ibqGetFullBufferTimeout (input_buffers_queue_t * *ibqp*, systime_t *timeout*)

Gets the next filled buffer from the queue.

Note

The function always acquires the same buffer if called repeatedly.

Postcondition

After calling the function the fields `ptr` and `top` are set at beginning and end of the buffer data or `NULL` if the queue is empty.

Parameters

`in`	*ibqp*	pointer to the `input_buffers_queue_t` object
`in`	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The operation status.

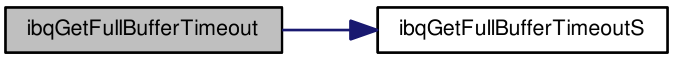
Return values

MSG_OK	if a buffer has been acquired.
MSG_TIMEOUT	if the specified time expired.
MSG_RESET	if the queue has been reset or has been put in suspended state.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.8.4.6 msg_t ibqGetFullBufferTimeoutS (**input_buffers_queue_t** ∗ *ibqp,* systime_t *timeout*)

Gets the next filled buffer from the queue.

Note

The function always acquires the same buffer if called repeatedly.

Postcondition

After calling the function the fields `ptr` and `top` are set at beginning and end of the buffer data or `NULL` if the queue is empty.

Parameters

`in`	*ibqp*	pointer to the `input_buffers_queue_t` object
`in`	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The operation status.

Return values

MSG_OK	if a buffer has been acquired.
MSG_TIMEOUT	if the specified time expired.
MSG_RESET	if the queue has been reset or has been put in suspended state.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

7.8.4.7 void ibqReleaseEmptyBuffer (**input_buffers_queue_t** ∗ *ibqp*)

Releases the buffer back in the queue.

Note

The object callback is called after releasing the buffer.

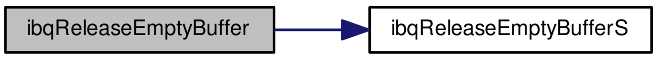
Parameters

`in`	*ibqp*	pointer to the `input_buffers_queue_t` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.8.4.8 void ibqReleaseEmptyBufferS (**input_buffers_queue_t** ∗ *ibqp*)

Releases the buffer back in the queue.

Note

The object callback is called after releasing the buffer.

Parameters

`in`	*ibqp*	pointer to the `input_buffers_queue_t` object

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

7.8.4.9 msg_t ibqGetTimeout (**input_buffers_queue_t** ∗ *ibqp,* systime_t *timeout*)

Input queue read with timeout.

This function reads a byte value from an input queue. If the queue is empty then the calling thread is suspended until a new buffer arrives in the queue or a timeout occurs.

Parameters

`in`	*ibqp*	pointer to the `input_buffers_queue_t` object
`in`	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

A byte value from the queue.

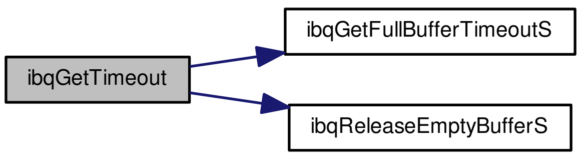
Return values

MSG_TIMEOUT	if the specified time expired.
MSG_RESET	if the queue has been reset or has been put in suspended state.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.8.4.10 size_t ibqReadTimeout (**input_buffers_queue_t** ∗ *ibqp,* uint8_t ∗ *bp,* size_t *n,* systime_t *timeout*)

Input queue read with timeout.

The function reads data from an input queue into a buffer. The operation completes when the specified amount of data has been transferred or after the specified timeout or if the queue has been reset.

Parameters

`in`	*ibqp*	pointer to the `input_buffers_queue_t` object
`out`	*bp*	pointer to the data buffer
`in`	*n*	the maximum amount of data to be transferred, the value 0 is reserved
`in`	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The number of bytes effectively transferred.

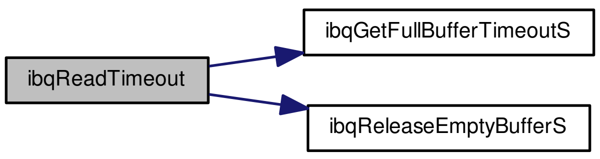
Return values

0	if a timeout occurred.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.8.4.11 void obqObjectInit (**output_buffers_queue_t** ∗ *obqp,* bool *suspended,* uint8_t ∗ *bp,* size_t *size,* size_t *n,* **bqnotify_t** *onfy,* void ∗ *link*)

Initializes an output buffers queue object.

Parameters

`out`	*obqp*	pointer to the `output_buffers_queue_t` object
`in`	*suspended*	initial state of the queue
`in`	*bp*	pointer to a memory area allocated for buffers
`in`	*size*	buffers size
`in`	*n*	number of buffers
`in`	*onfy*	callback called when a buffer is posted in the queue
`in`	*link*	application defined pointer

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.8.4.12 void obqResetI (output_buffers_queue_t ∗ *obqp*)

Resets an output buffers queue.

All the data in the output buffers queue is erased and lost, any waiting thread is resumed with status `MSG_RESET`.

Note

A reset operation can be used by a low level driver in order to obtain immediate attention from the high level layers.

Parameters

`in`	*obqp*	pointer to the `output_buffers_queue_t` object

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.8.4.13 uint8_t ∗ obqGetFullBufferI (output_buffers_queue_t ∗ *obqp,* size_t ∗ *sizep*)

Gets the next filled buffer from the queue.

Note

The function always returns the same buffer if called repeatedly.

Parameters

`in`	*obqp*	pointer to the `output_buffers_queue_t` object
`out`	*sizep*	pointer to the filled buffer size

Returns

A pointer to the filled buffer.

Return values

NULL	if the queue is empty.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.8.4.14 void obqReleaseEmptyBufferI (**output_buffers_queue_t** ∗ *obqp*)

Releases the next filled buffer back in the queue.

Parameters

`in`	*obqp*	pointer to the `output_buffers_queue_t` object

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.8.4.15 msg_t obqGetEmptyBufferTimeout (**output_buffers_queue_t** ∗ *obqp*, systime_t *timeout*)

Gets the next empty buffer from the queue.

Note

The function always acquires the same buffer if called repeatedly.

Postcondition

After calling the function the fields `ptr` and `top` are set at beginning and end of the buffer data or `NULL` if the queue is empty.

Parameters

`in`	*obqp*	pointer to the `output_buffers_queue_t` object
`in`	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The operation status.

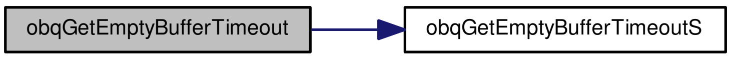
Return values

MSG_OK	if a buffer has been acquired.
MSG_TIMEOUT	if the specified time expired.
MSG_RESET	if the queue has been reset or has been put in suspended state.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.8.4.16 msg_t obqGetEmptyBufferTimeoutS (**output_buffers_queue_t** ∗ ***obqp***, **systime_t** ***timeout***)

Gets the next empty buffer from the queue.

Note

The function always acquires the same buffer if called repeatedly.

Postcondition

After calling the function the fields `ptr` and `top` are set at beginning and end of the buffer data or `NULL` if the queue is empty.

Parameters

`in`	*obqp*	pointer to the `output_buffers_queue_t` object
`in`	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The operation status.

Return values

MSG_OK	if a buffer has been acquired.
MSG_TIMEOUT	if the specified time expired.
MSG_RESET	if the queue has been reset or has been put in suspended state.

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

7.8.4.17 void obqPostFullBuffer (**output_buffers_queue_t** ∗ ***obqp***, **size_t** ***size***)

Posts a new filled buffer to the queue.

Note

The object callback is called after releasing the buffer.

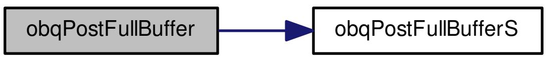
Parameters

`in`	*obqp*	pointer to the `output_buffers_queue_t` object
`in`	*size*	used size of the buffer, cannot be zero

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.8.4.18 void obqPostFullBufferS (**output_buffers_queue_t** ∗ *obqp,* size_t *size*)

Posts a new filled buffer to the queue.

Note

The object callback is called after releasing the buffer.

Parameters

`in`	*obqp*	pointer to the `output_buffers_queue_t` object
`in`	*size*	used size of the buffer, cannot be zero

Function Class:

This is an **S-Class** API, this function can be invoked from within a system lock zone by threads only.

7.8.4.19 msg_t obqPutTimeout (**output_buffers_queue_t** ∗ *obqp,* uint8_t *b,* systime_t *timeout*)

Output queue write with timeout.

This function writes a byte value to an output queue. If the queue is full then the calling thread is suspended until a new buffer is freed in the queue or a timeout occurs.

Parameters

`in`	*obqp*	pointer to the `output_buffers_queue_t` object
`in`	*b*	byte value to be transferred

Parameters

in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

A byte value from the queue.

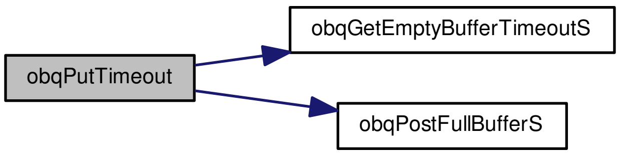
Return values

MSG_TIMEOUT	if the specified time expired.
MSG_RESET	if the queue has been reset or has been put in suspended state.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.8.4.20 size_t obqWriteTimeout (output_buffers_queue_t ∗ *obqp*, const uint8_t ∗ *bp*, size_t *n*, systime_t *timeout*)

Output queue write with timeout.

The function writes data from a buffer to an output queue. The operation completes when the specified amount of data has been transferred or after the specified timeout or if the queue has been reset.

Parameters

in	*obqp*	pointer to the `output_buffers_queue_t` object
in	*bp*	pointer to the data buffer
in	*n*	the maximum amount of data to be transferred, the value 0 is reserved
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The number of bytes effectively transferred.

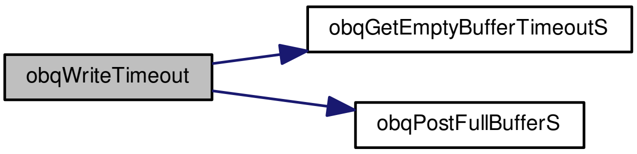
Return values

0	if a timeout occurred.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.8.4.21 bool obqTryFlushI (**output_buffers_queue_t** ∗ *obqp*)

Flushes the current, partially filled, buffer to the queue.

Note

The notification callback is not invoked because the function is meant to be called from ISR context. An operation status is returned instead.

Parameters

`in`	*obqp*	pointer to the `output_buffers_queue_t` object

Returns

The operation status.

Return values

false	if no new filled buffer has been posted to the queue.
true	if a new filled buffer has been posted to the queue.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.8.4.22 void obqFlush (output_buffers_queue_t * *obqp*)

Flushes the current, partially filled, buffer to the queue.

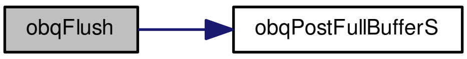
Parameters

`in`	*obqp*	pointer to the `output_buffers_queue_t` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.9 Abstract I/O Channel

7.9.1 Detailed Description

This module defines an abstract interface for I/O channels by extending the `BaseSequentialStream` interface. Note that no code is present, I/O channels are just abstract interface like structures, you should look at the systems as to a set of abstract C++ classes (even if written in C). Specific device drivers can use/extend the interface and implement them.
This system has the advantage to make the access to channels independent from the implementation logic.

Macros

- #define _base_channel_methods

  `BaseChannel` specific methods.
- #define _base_channel_data _base_sequential_stream_data

  `BaseChannel` specific data.
- #define _base_asynchronous_channel_methods _base_channel_methods \

  `BaseAsynchronousChannel` specific methods.
- #define _base_asynchronous_channel_data

  `BaseAsynchronousChannel` specific data.

Macro Functions (BaseChannel)

- #define chnPutTimeout(ip, b, time) ((ip)->vmt->putt(ip, b, time))

  Channel blocking byte write with timeout.
- #define chnGetTimeout(ip, time) ((ip)->vmt->gett(ip, time))

  Channel blocking byte read with timeout.
- #define chnWrite(ip, bp, n) streamWrite(ip, bp, n)

  Channel blocking write.
- #define chnWriteTimeout(ip, bp, n, time) ((ip)->vmt->writet(ip, bp, n, time))

  Channel blocking write with timeout.
- #define chnRead(ip, bp, n) streamRead(ip, bp, n)

  Channel blocking read.
- #define chnReadTimeout(ip, bp, n, time) ((ip)->vmt->readt(ip, bp, n, time))

  Channel blocking read with timeout.

I/O status flags added to the event listener

- #define CHN_NO_ERROR (eventflags_t)0

  No pending conditions.
- #define CHN_CONNECTED (eventflags_t)1

  Connection happened.
- #define CHN_DISCONNECTED (eventflags_t)2

  Disconnection happened.
- #define CHN_INPUT_AVAILABLE (eventflags_t)4

  Data available in the input queue.
- #define CHN_OUTPUT_EMPTY (eventflags_t)8

  Output queue empty.
- #define CHN_TRANSMISSION_END (eventflags_t)16

  Transmission end.

Macro Functions (BaseAsynchronousChannel)

- #define chnGetEventSource(ip) (&((ip)->event))

  Returns the I/O condition event source.
- #define chnAddFlagsI(ip, flags)

  Adds status flags to the listeners's flags mask.

Data Structures

- struct BaseChannelVMT

  `BaseChannel` *virtual methods table.*
- struct BaseChannel

  Base channel class.
- struct BaseAsynchronousChannelVMT

  `BaseAsynchronousChannel` *virtual methods table.*
- struct BaseAsynchronousChannel

  Base asynchronous channel class.

7.9.2 Macro Definition Documentation

7.9.2.1 #define _base_channel_methods

Value:

```
_base_sequential_stream_methods                                          \
  /* Channel put method with timeout specification.*/                     \
  msg_t (*putt)(void *instance, uint8_t b, systime_t time);               \
  /* Channel get method with timeout specification.*/                     \
  msg_t (*gett)(void *instance, systime_t time);                          \
  /* Channel write method with timeout specification.*/                   \
  size_t (*writet)(void *instance, const uint8_t *bp,                     \
                   size_t n, systime_t time);                             \
  /* Channel read method with timeout specification.*/                    \
  size_t (*readt)(void *instance, uint8_t *bp, size_t n, systime_t time);
```

`BaseChannel` specific methods.

7.9.2.2 #define _base_channel_data _base_sequential_stream_data

`BaseChannel` specific data.

Note

It is empty because `BaseChannel` is only an interface without implementation.

7.9.2.3 #define chnPutTimeout(*ip*, *b*, *time*) ((ip)->vmt->putt(ip, b, time))

Channel blocking byte write with timeout.

This function writes a byte value to a channel. If the channel is not ready to accept data then the calling thread is suspended.

Parameters

in	*ip*	pointer to a `BaseChannel` or derived class
in	*b*	the byte value to be written to the channel

Parameters

in	*time*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The operation status.

Return values

STM_OK	if the operation succeeded.
STM_TIMEOUT	if the specified time expired.
STM_RESET	if the channel associated queue (if any) was reset.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.9.2.4 #define chnGetTimeout(*ip*, *time*) ((ip)->vmt->gett(ip, time))

Channel blocking byte read with timeout.

This function reads a byte value from a channel. If the data is not available then the calling thread is suspended.

Parameters

in	*ip*	pointer to a `BaseChannel` or derived class
in	*time*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

A byte value from the queue.

Return values

STM_TIMEOUT	if the specified time expired.
STM_RESET	if the channel associated queue (if any) has been reset.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.9.2.5 #define chnWrite(*ip*, *bp*, *n*) streamWrite(ip, bp, n)

Channel blocking write.

The function writes data from a buffer to a channel. If the channel is not ready to accept data then the calling thread is suspended.

Parameters

`in`	*ip*	pointer to a `BaseChannel` or derived class
`out`	*bp*	pointer to the data buffer
`in`	*n*	the maximum amount of data to be transferred

Returns

The number of bytes transferred.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.9.2.6 #define chnWriteTimeout(*ip*, *bp*, *n*, *time*) ((ip)->vmt->writet(ip, bp, n, time))

Channel blocking write with timeout.

The function writes data from a buffer to a channel. If the channel is not ready to accept data then the calling thread is suspended.

Parameters

`in`	*ip*	pointer to a `BaseChannel` or derived class
`out`	*bp*	pointer to the data buffer
`in`	*n*	the maximum amount of data to be transferred
`in`	*time*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The number of bytes transferred.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.9.2.7 #define chnRead(*ip*, *bp*, *n*) streamRead(ip, bp, n)

Channel blocking read.

The function reads data from a channel into a buffer. If the data is not available then the calling thread is suspended.

Parameters

`in`	*ip*	pointer to a `BaseChannel` or derived class

Parameters

`in`	*bp*	pointer to the data buffer
`in`	*n*	the maximum amount of data to be transferred

Returns

The number of bytes transferred.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.9.2.8 #define chnReadTimeout(*ip, bp, n, time*) ((ip)->vmt->readt(ip, bp, n, time))

Channel blocking read with timeout.

The function reads data from a channel into a buffer. If the data is not available then the calling thread is suspended.

Parameters

`in`	*ip*	pointer to a `BaseChannel` or derived class
`in`	*bp*	pointer to the data buffer
`in`	*n*	the maximum amount of data to be transferred
`in`	*time*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The number of bytes transferred.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.9.2.9 #define CHN_NO_ERROR (eventflags_t)0

No pending conditions.

7.9.2.10 #define CHN_CONNECTED (eventflags_t)1

Connection happened.

7.9.2.11 #define CHN_DISCONNECTED (eventflags_t)2

Disconnection happened.

7.9.2.12 #define CHN_INPUT_AVAILABLE (eventflags_t)4

Data available in the input queue.

7.9.2.13 #define CHN_OUTPUT_EMPTY (eventflags_t)8

Output queue empty.

7.9.2.14 #define CHN_TRANSMISSION_END (eventflags_t)16

Transmission end.

7.9.2.15 #define _base_asynchronous_channel_methods _base_channel_methods \

BaseAsynchronousChannel specific methods.

7.9.2.16 #define _base_asynchronous_channel_data

Value:

```
_base_channel_data                                                        \
  /* I/O condition event source.*/                                          \
  event_source_t          event;
```

BaseAsynchronousChannel specific data.

7.9.2.17 #define chnGetEventSource(*ip*) (&((ip)->event))

Returns the I/O condition event source.

The event source is broadcasted when an I/O condition happens.

Parameters

in	*ip*	pointer to a BaseAsynchronousChannel or derived class

Returns

A pointer to an EventSource object.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.9.2.18 #define chnAddFlagsI(*ip*, *flags*)

Value:

```
{                                          \
  osalEventBroadcastFlagsI(&(ip)->event, flags);                          \
}
```

Adds status flags to the listeners's flags mask.

This function is usually called from the I/O ISRs in order to notify I/O conditions such as data events, errors, signal changes etc.

Parameters

in	*ip*	pointer to a BaseAsynchronousChannel or derived class
in	*flags*	condition flags to be added to the listener flags mask

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.10 Abstract Files

7.10.1 Detailed Description

This module define an abstract interface for generic data files by extending the `BaseSequentialStream` interface. Note that no code is present, data files are just abstract interface-like structures, you should look at the systems as to a set of abstract C++ classes (even if written in C). This system has the advantage to make the access to streams independent from the implementation logic.
The data files interface can be used as base class for high level object types such as an API for a File System implementation.

Macros

- #define _file_stream_methods

  FileStream specific methods.
- #define _file_stream_data _base_sequential_stream_data

  `FileStream` specific data.

Files return codes

- #define FILE_OK STM_OK

  No error return code.
- #define FILE_ERROR STM_TIMEOUT

  Error code from the file stream methods.
- #define FILE_EOF STM_RESET

  End-of-file condition for file get/put methods.

Macro Functions (FileStream)

- #define fileStreamWrite(ip, bp, n) streamWrite(ip, bp, n)

  File stream write.
- #define fileStreamRead(ip, bp, n) streamRead(ip, bp, n)

  File stream read.
- #define fileStreamPut(ip, b) streamPut(ip, b)

  File stream blocking byte write.
- #define fileStreamGet(ip) streamGet(ip)

  File stream blocking byte read.
- #define fileStreamClose(ip) ((ip)->vmt->close(ip))

  File Stream close.
- #define fileStreamGetError(ip) ((ip)->vmt->geterror(ip))

  Returns an implementation dependent error code.
- #define fileStreamGetSize(ip) ((ip)->vmt->getsize(ip))

  Returns the current file size.
- #define fileStreamGetPosition(ip) ((ip)->vmt->getposition(ip))

  Returns the current file pointer position.
- #define fileStreamSeek(ip, offset) ((ip)->vmt->lseek(ip, offset))

  Moves the file current pointer to an absolute position.

Typedefs

- typedef uint32_t fileoffset_t

  File offset type.

Data Structures

- struct FileStreamVMT

  FileStream virtual methods table.
- struct FileStream

  Base file stream class.

7.10.2 Macro Definition Documentation

7.10.2.1 #define FILE_OK STM_OK

No error return code.

7.10.2.2 #define FILE_ERROR STM_TIMEOUT

Error code from the file stream methods.

7.10.2.3 #define FILE_EOF STM_RESET

End-of-file condition for file get/put methods.

7.10.2.4 #define _file_stream_methods

Value:

```
_base_sequential_stream_methods                                      \
  /* File close method.*/                                             \
  msg_t (*close)(void *instance);                                     \
  /* Get last error code method.*/                                    \
  msg_t (*geterror)(void *instance);                                  \
  /* File get size method.*/                                          \
  msg_t (*getsize)(void *instance);                                   \
  /* File get current position method.*/                              \
  msg_t (*getposition)(void *instance);                               \
  /* File seek method.*/                                              \
  msg_t (*lseek)(void *instance, fileoffset_t offset);
```

FileStream specific methods.

7.10.2.5 #define _file_stream_data _base_sequential_stream_data

FileStream specific data.

Note

It is empty because FileStream is only an interface without implementation.

7.10.2.6 #define fileStreamWrite(*ip,* *bp,* *n*) streamWrite(ip, bp, n)

File stream write.

The function writes data from a buffer to a file stream.

Parameters

in	*ip*	pointer to a FileStream or derived class
in	*bp*	pointer to the data buffer
in	*n*	the maximum amount of data to be transferred

Returns

The number of bytes transferred. The return value can be less than the specified number of bytes if an end-of-file condition has been met.

Return values

FILE_ERROR	operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.10.2.7 #define fileStreamRead(*ip*, *bp*, *n*) **streamRead**(ip, bp, n)

File stream read.

The function reads data from a file stream into a buffer.

Parameters

`in`	*ip*	pointer to a `FileStream` or derived class
`out`	*bp*	pointer to the data buffer
`in`	*n*	the maximum amount of data to be transferred

Returns

The number of bytes transferred. The return value can be less than the specified number of bytes if an end-of-file condition has been met.

Return values

FILE_ERROR	operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.10.2.8 #define fileStreamPut(*ip*, *b*) **streamPut**(ip, b)

File stream blocking byte write.

This function writes a byte value to a channel. If the channel is not ready to accept data then the calling thread is suspended.

Parameters

`in`	*ip*	pointer to a `FileStream` or derived class
`in`	*b*	the byte value to be written to the channel

Returns

The operation status.

Return values

FILE_OK	if the operation succeeded.
FILE_ERROR	operation failed.
FILE_EOF	if an end-of-file condition has been met.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.10.2.9 #define fileStreamGet(*ip*) streamGet(ip)

File stream blocking byte read.

This function reads a byte value from a channel. If the data is not available then the calling thread is suspended.

Parameters

`in`	*ip*	pointer to a `FileStream` or derived class

Returns

A byte value from the queue.

Return values

FILE_ERROR	operation failed.
FILE_EOF	if an end-of-file condition has been met.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.10.2.10 #define fileStreamClose(*ip*) ((ip)->vmt->close(ip))

File Stream close.

The function closes a file stream.

Parameters

`in`	*ip*	pointer to a `FileStream` or derived class

Returns

The operation status.

Return values

FILE_OK	no error.
FILE_ERROR	operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.10.2.11 #define fileStreamGetError(*ip*) ((ip)->vmt->geterror(ip))

Returns an implementation dependent error code.

Precondition

The previously called function must have returned `FILE_ERROR`.

Parameters

`in`	*ip*	pointer to a `FileStream` or derived class

Returns

Implementation dependent error code.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.10.2.12 #define fileStreamGetSize(*ip*) ((ip)->vmt->getsize(ip))

Returns the current file size.

Parameters

`in`	*ip*	pointer to a `FileStream` or derived class

Returns

The file size.

Return values

FILE_ERROR	operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.10.2.13 #define fileStreamGetPosition(*ip*) ((ip)->vmt->getposition(ip))

Returns the current file pointer position.

Parameters

`in`	*ip*	pointer to a `FileStream` or derived class

Returns

The current position inside the file.

Return values

FILE_ERROR	operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.10.2.14 #define fileStreamSeek(*ip,* *offset*) ((ip)->vmt->lseek(ip, offset))

Moves the file current pointer to an absolute position.

Parameters

`in`	*ip*	pointer to a `FileStream` or derived class
`in`	*offset*	new absolute position

Returns

The operation status.

Return values

FILE_OK	no error.
FILE_ERROR	operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.10.3 Typedef Documentation

7.10.3.1 typedef uint32_t **fileoffset_t**

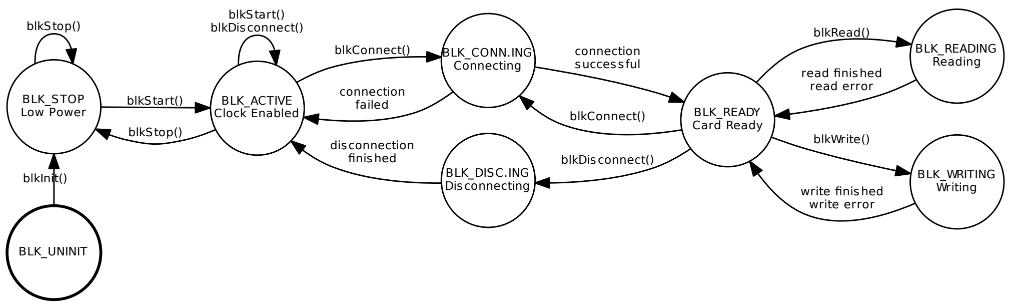
File offset type.

7.11 Abstract I/O Block Device

7.11.1 Detailed Description

7.11.2 Driver State Machine

The drivers implementing this interface shall implement the following state machine internally. Not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



This module defines an abstract interface for accessing generic block devices.
Note that no code is present, just abstract interfaces-like structures, you should look at the system as to a set of abstract C++ classes (even if written in C). This system has then advantage to make the access to block devices independent from the implementation logic.

Macros

- #define _base_block_device_methods

  `BaseBlockDevice` specific methods.
- #define _base_block_device_data

  `BaseBlockDevice` specific data.

Macro Functions (BaseBlockDevice)

- #define blkGetDriverState(ip) ((ip)->state)

  Returns the driver state.
- #define blkIsTransferring(ip)

  Determines if the device is transferring data.
- #define blkIsInserted(ip) ((ip)->vmt->is_inserted(ip))

  Returns the media insertion status.
- #define blkIsWriteProtected(ip) ((ip)->vmt->is_protected(ip))

  Returns the media write protection status.
- #define blkConnect(ip) ((ip)->vmt->connect(ip))

  Performs the initialization procedure on the block device.
- #define blkDisconnect(ip) ((ip)->vmt->disconnect(ip))

Terminates operations on the block device.
- #define blkRead(ip, startblk, buf, n) ((ip)->vmt->read(ip, startblk, buf, n))

  Reads one or more blocks.
- #define blkWrite(ip, startblk, buf, n) ((ip)->vmt->write(ip, startblk, buf, n))

  Writes one or more blocks.
- #define blkSync(ip) ((ip)->vmt->sync(ip))

  Ensures write synchronization.
- #define blkGetInfo(ip, bdip) ((ip)->vmt->get_info(ip, bdip))

  Returns a media information structure.

Data Structures

- struct BlockDeviceInfo

  Block device info.
- struct BaseBlockDeviceVMT

  `BaseBlockDevice` virtual methods table.
- struct BaseBlockDevice

  Base block device class.

Enumerations

7.11.3 Macro Definition Documentation

7.11.3.1 #define _base_block_device_methods

Value:

```
/* Removable media detection.*/                                          \
  bool (*is_inserted)(void *instance);                                    \
  /* Removable write protection detection.*/                              \
  bool (*is_protected)(void *instance);                                   \
  /* Connection to the block device.*/                                    \
  bool (*connect)(void *instance);                                        \
  /* Disconnection from the block device.*/                               \
  bool (*disconnect)(void *instance);                                     \
  /* Reads one or more blocks.*/                                          \
  bool (*read)(void *instance, uint32_t startblk,                         \
               uint8_t *buffer, uint32_t n);                              \
  /* Writes one or more blocks.*/                                         \
  bool (*write)(void *instance, uint32_t startblk,                        \
                const uint8_t *buffer, uint32_t n);                       \
  /* Write operations synchronization.*/                                  \
  bool (*sync)(void *instance);                                           \
  /* Obtains info about the media.*/                                      \
  bool (*get_info)(void *instance, BlockDeviceInfo *bdip);
```

`BaseBlockDevice` specific methods.

7.11.3.2 #define _base_block_device_data

Value:

```
/* Driver state.*/                                                       \
  blkstate_t             state;
```

`BaseBlockDevice` specific data.

7.11.3.3 #define blkGetDriverState(*ip*) ((ip)->state)

Returns the driver state.

Note

Can be called in ISR context.

Parameters

in	*ip*	pointer to a `BaseBlockDevice` or derived class

Returns

The driver state.

Function Class:

Special function, this function has special requirements see the notes.

7.11.3.4 #define blkIsTransferring(*ip*)

Value:

```
((((ip)->state) == BLK_CONNECTING) ||          \
                              (((ip)->state) == BLK_DISCONNECTING) ||       \
                              (((ip)->state) == BLK_READING) ||             \
                              (((ip)->state) == BLK_WRITING))
```

Determines if the device is transferring data.

Note

Can be called in ISR context.

Parameters

in	*ip*	pointer to a `BaseBlockDevice` or derived class

Returns

The driver state.

Return values

FALSE	the device is not transferring data.
TRUE	the device not transferring data.

Function Class:

Special function, this function has special requirements see the notes.

7.11.3.5 #define blkIsInserted(*ip*) ((ip)->vmt->is_inserted(ip))

Returns the media insertion status.

Note

On some implementations this function can only be called if the device is not transferring data. The function `blkIsTransferring()` should be used before calling this function.

Parameters

`in`	*ip*	pointer to a `BaseBlockDevice` or derived class

Returns

The media state.

Return values

FALSE	media not inserted.
TRUE	media inserted.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.11.3.6 #define blkIsWriteProtected(*ip*) ((ip)->vmt->is_protected(ip))

Returns the media write protection status.

Parameters

`in`	*ip*	pointer to a `BaseBlockDevice` or derived class

Returns

The media state.

Return values

FALSE	writable media.
TRUE	non writable media.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.11.3.7 #define blkConnect(*ip*) ((ip)->vmt->connect(ip))

Performs the initialization procedure on the block device.

This function should be performed before I/O operations can be attempted on the block device and after insertion has been confirmed using `blkIsInserted()`.

Parameters

`in`	*ip*	pointer to a `BaseBlockDevice` or derived class

Returns

The operation status.

Return values

HAL_SUCCESS	operation succeeded.
HAL_FAILED	operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.11.3.8 #define blkDisconnect(*ip*) ((ip)->vmt->disconnect(ip))

Terminates operations on the block device.

This operation safely terminates operations on the block device.

Parameters

`in`	*ip*	pointer to a `BaseBlockDevice` or derived class

Returns

The operation status.

Return values

HAL_SUCCESS	operation succeeded.
HAL_FAILED	operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.11.3.9 #define blkRead(*ip, startblk, buf, n*) ((ip)->vmt->read(ip, startblk, buf, n))

Reads one or more blocks.

Parameters

`in`	*ip*	pointer to a `BaseBlockDevice` or derived class
`in`	*startblk*	first block to read

Parameters

out	*buf*	pointer to the read buffer
in	*n*	number of blocks to read

Returns

The operation status.

Return values

HAL_SUCCESS	operation succeeded.
HAL_FAILED	operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.11.3.10 #define blkWrite(*ip,* *startblk,* *buf,* *n*) ((ip)->vmt->write(ip, startblk, buf, n))

Writes one or more blocks.

Parameters

in	*ip*	pointer to a `BaseBlockDevice` or derived class
in	*startblk*	first block to write
out	*buf*	pointer to the write buffer
in	*n*	number of blocks to write

Returns

The operation status.

Return values

HAL_SUCCESS	operation succeeded.
HAL_FAILED	operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.11.3.11 #define blkSync(*ip*) ((ip)->vmt->sync(ip))

Ensures write synchronization.

Parameters

in	*ip*	pointer to a `BaseBlockDevice` or derived class

Returns

The operation status.

Return values

HAL_SUCCESS	operation succeeded.
HAL_FAILED	operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.11.3.12 #define blkGetInfo(*ip,* *bdip*) ((ip)->vmt->get_info(ip, bdip))

Returns a media information structure.

Parameters

`in`	*ip*	pointer to a `BaseBlockDevice` or derived class
`out`	*bdip*	pointer to a `BlockDeviceInfo` structure

Returns

The operation status.

Return values

HAL_SUCCESS	operation succeeded.
HAL_FAILED	operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.11.4 Enumeration Type Documentation

7.11.4.1 enum blkstate_t

Driver state machine possible states.

Enumerator

BLK_UNINIT	Not initialized.
BLK_STOP	Stopped.
BLK_ACTIVE	Interface active.
BLK_CONNECTING	Connection in progress.
BLK_DISCONNECTING	Disconnection in progress.
BLK_READY	Device ready.
BLK_READING	Read operation in progress.
BLK_WRITING	Write operation in progress.
BLK_SYNCING	Sync. operation in progress.

7.12 I/O Bytes Queues

7.12.1 Detailed Description

Queues are mostly used in serial-like device drivers. Serial device drivers are usually designed to have a lower side (lower driver, it is usually an interrupt service routine) and an upper side (upper driver, accessed by the application threads).
There are several kind of queues:

- **Input queue**, unidirectional queue where the writer is the lower side and the reader is the upper side.
- **Output queue**, unidirectional queue where the writer is the upper side and the reader is the lower side.
- **Full duplex queue**, bidirectional queue. Full duplex queues are implemented by pairing an input queue and an output queue together.

Queue functions returned status value

- #define Q_OK MSG_OK

  Operation successful.
- #define Q_TIMEOUT MSG_TIMEOUT

  Timeout condition.
- #define Q_RESET MSG_RESET

  Queue has been reset.
- #define Q_EMPTY MSG_TIMEOUT

  Queue empty.
- #define Q_FULL MSG_TIMEOUT

  Queue full,.

Macro Functions

- #define qSizeX(qp)

  Returns the queue's buffer size.
- #define qSpaceI(qp) ((qp)->q_counter)

  Queue space.
- #define qGetLink(qp) ((qp)->q_link)

  Returns the queue application-defined link.
- #define iqGetFullI(iqp) qSpaceI(iqp)

  Returns the filled space into an input queue.
- #define iqGetEmptyI(iqp) (qSizeX(iqp) - qSpaceI(iqp))

  Returns the empty space into an input queue.
- #define iqIsEmptyI(iqp) ((bool)(qSpaceI(iqp) == 0U))

  Evaluates to `true` if the specified input queue is empty.
- #define iqIsFullI(iqp)

  Evaluates to `true` if the specified input queue is full.
- #define iqGet(iqp) iqGetTimeout(iqp, TIME_INFINITE)

  Input queue read.
- #define oqGetFullI(oqp) (qSizeX(oqp) - qSpaceI(oqp))

  Returns the filled space into an output queue.
- #define oqGetEmptyI(oqp) qSpaceI(oqp)

  Returns the empty space into an output queue.

- #define oqIsEmptyI(oqp)

  Evaluates to `true` if the specified output queue is empty.
- #define oqIsFullI(oqp) ((bool)(qSpaceI(oqp) == 0U))

  Evaluates to `true` if the specified output queue is full.
- #define oqPut(oqp, b) oqPutTimeout(oqp, b, TIME_INFINITE)

  Output queue write.

Typedefs

- typedef struct io_queue io_queue_t

  Type of a generic I/O queue structure.
- typedef void(∗ qnotify_t) (io_queue_t ∗qp)

  Queue notification callback type.
- typedef io_queue_t input_queue_t

  Type of an input queue structure.
- typedef io_queue_t output_queue_t

  Type of an output queue structure.

Data Structures

- struct io_queue

  Generic I/O queue structure.

Functions

- void iqObjectInit (input_queue_t ∗iqp, uint8_t ∗bp, size_t size, qnotify_t infy, void ∗link)

  Initializes an input queue.
- void iqResetI (input_queue_t ∗iqp)

  Resets an input queue.
- msg_t iqPutI (input_queue_t ∗iqp, uint8_t b)

  Input queue write.
- msg_t iqGetTimeout (input_queue_t ∗iqp, systime_t timeout)

  Input queue read with timeout.
- size_t iqReadTimeout (input_queue_t ∗iqp, uint8_t ∗bp, size_t n, systime_t timeout)

  Input queue read with timeout.
- void oqObjectInit (output_queue_t ∗oqp, uint8_t ∗bp, size_t size, qnotify_t onfy, void ∗link)

  Initializes an output queue.
- void oqResetI (output_queue_t ∗oqp)

  Resets an output queue.
- msg_t oqPutTimeout (output_queue_t ∗oqp, uint8_t b, systime_t timeout)

  Output queue write with timeout.
- msg_t oqGetI (output_queue_t ∗oqp)

  Output queue read.
- size_t oqWriteTimeout (output_queue_t ∗oqp, const uint8_t ∗bp, size_t n, systime_t timeout)

  Output queue write with timeout.

7.12.2 Macro Definition Documentation

7.12.2.1 #define Q_OK MSG_OK

Operation successful.

7.12.2.2 #define Q_TIMEOUT MSG_TIMEOUT

Timeout condition.

7.12.2.3 #define Q_RESET MSG_RESET

Queue has been reset.

7.12.2.4 #define Q_EMPTY MSG_TIMEOUT

Queue empty.

7.12.2.5 #define Q_FULL MSG_TIMEOUT

Queue full,.

7.12.2.6 #define qSizeX(*qp*)

Value:

```
/*lint -save -e9033 [10.8] The cast is safe.*/                              \
  ((size_t)((qp)->q_top - (qp)->q_buffer))                                    \
  /*lint -restore*/
```

Returns the queue's buffer size.

Parameters

in	*qp*	pointer to a `io_queue_t` structure

Returns

The buffer size.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

7.12.2.7 #define qSpaceI(*qp*) ((qp)->q_counter)

Queue space.

Returns the used space if used on an input queue or the empty space if used on an output queue.

Parameters

in	*qp*	pointer to a `io_queue_t` structure

Returns

The buffer space.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.12.2.8 #define qGetLink(*qp*) ((qp)->q_link)

Returns the queue application-defined link.

Note

This function can be called in any context.

Parameters

in	*qp*	pointer to a `io_queue_t` structure

Returns

The application-defined link.

Function Class:

Special function, this function has special requirements see the notes.

7.12.2.9 #define iqGetFullI(*iqp*) qSpaceI(iqp)

Returns the filled space into an input queue.

Parameters

in	*iqp*	pointer to an `input_queue_t` structure

Returns

The number of full bytes in the queue.

Return values

0	if the queue is empty.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.12.2.10 #define iqGetEmptyI(*iqp*) (qSizeX(iqp) - qSpaceI(iqp))

Returns the empty space into an input queue.

Parameters

in	*iqp*	pointer to an `input_queue_t` structure

Returns

The number of empty bytes in the queue.

Return values

0	if the queue is full.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.12.2.11 #define iqIsEmptyI(*iqp*) ((bool)(qSpaceI(iqp) == 0U))

Evaluates to `true` if the specified input queue is empty.

Parameters

in	*iqp*	pointer to an `input_queue_t` structure

Returns

The queue status.

Return values

false	if the queue is not empty.
true	if the queue is empty.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.12.2.12 #define iqIsFullI(*iqp*)

Value:

```
/*lint -save -e9007 [13.5] No side effects, a pointer is passed.*/         \
  ((bool)(((iqp)->q_wrptr == (iqp)->q_rdptr) && ((iqp)->q_counter != 0U)))  \
  /*lint -restore*/
```

Evaluates to `true` if the specified input queue is full.

Parameters

`in`	*iqp*	pointer to an `input_queue_t` structure

Returns

The queue status.

Return values

false	if the queue is not full.
true	if the queue is full.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.12.2.13 #define iqGet(*iqp*) **iqGetTimeout**(iqp, TIME_INFINITE)

Input queue read.

This function reads a byte value from an input queue. If the queue is empty then the calling thread is suspended until a byte arrives in the queue.

Parameters

`in`	*iqp*	pointer to an `input_queue_t` structure

Returns

A byte value from the queue.

Return values

MSG_RESET	if the queue has been reset.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.12.2.14 #define oqGetFullI(*oqp*) (**qSizeX**(oqp) - **qSpaceI**(oqp))

Returns the filled space into an output queue.

Parameters

`in`	*oqp*	pointer to an `output_queue_t` structure

Returns

The number of full bytes in the queue.

Return values

0	if the queue is empty.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.12.2.15 #define oqGetEmptyI(*oqp*) qSpaceI(oqp)

Returns the empty space into an output queue.

Parameters

in	*oqp*	pointer to an `output_queue_t` structure

Returns

The number of empty bytes in the queue.

Return values

0	if the queue is full.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.12.2.16 #define oqIsEmptyI(*oqp*)

Value:

```
/*lint -save -e9007 [13.5] No side effects, a pointer is passed.*/         \
  ((bool)(((oqp)->q_wrptr == (oqp)->q_rdptr) && ((oqp)->q_counter != 0U)))  \
  /*lint -restore*/
```

Evaluates to `true` if the specified output queue is empty.

Parameters

in	*oqp*	pointer to an `output_queue_t` structure

Returns

The queue status.

Return values

false	if the queue is not empty.
true	if the queue is empty.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.12.2.17 #define oqIsFullI(*oqp*) ((bool)(**qSpaceI**(oqp) == 0U))

Evaluates to `true` if the specified output queue is full.

Parameters

`in`	*oqp*	pointer to an `output_queue_t` structure

Returns

The queue status.

Return values

false	if the queue is not full.
true	if the queue is full.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.12.2.18 #define oqPut(*oqp, b*) **oqPutTimeout**(oqp, b, TIME_INFINITE)

Output queue write.

This function writes a byte value to an output queue. If the queue is full then the calling thread is suspended until there is space in the queue.

Parameters

`in`	*oqp*	pointer to an `output_queue_t` structure
`in`	*b*	the byte value to be written in the queue

Returns

The operation status.

Return values

MSG_OK	if the operation succeeded.

Return values

MSG_RESET	if the queue has been reset.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.12.3 Typedef Documentation

7.12.3.1 typedef struct io_queue io_queue_t

Type of a generic I/O queue structure.

7.12.3.2 typedef void(∗ qnotify_t) (io_queue_t ∗qp)

Queue notification callback type.

Parameters

in	*qp*	the queue pointer

7.12.3.3 typedef io_queue_t input_queue_t

Type of an input queue structure.

This structure represents a generic asymmetrical input queue. Writing to the queue is non-blocking and can be performed from interrupt handlers or from within a kernel lock zone. Reading the queue can be a blocking operation and is supposed to be performed by a system thread.

7.12.3.4 typedef io_queue_t output_queue_t

Type of an output queue structure.

This structure represents a generic asymmetrical output queue. Reading from the queue is non-blocking and can be performed from interrupt handlers or from within a kernel lock zone. Writing the queue can be a blocking operation and is supposed to be performed by a system thread.

7.12.4 Function Documentation

7.12.4.1 void iqObjectInit (input_queue_t ∗ *iqp,* uint8_t ∗ *bp,* size_t *size,* qnotify_t *infy,* void ∗ *link*)

Initializes an input queue.

A Semaphore is internally initialized and works as a counter of the bytes contained in the queue.

Note

The callback is invoked from within the S-Locked system state.

Parameters

out	*iqp*	pointer to an `input_queue_t` structure

Parameters

in	*bp*	pointer to a memory area allocated as queue buffer
in	*size*	size of the queue buffer
in	*infy*	pointer to a callback function that is invoked when data is read from the queue. The value can be `NULL`.
in	*link*	application defined pointer

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.12.4.2 void iqResetI (input_queue_t ∗ *iqp*)

Resets an input queue.

All the data in the input queue is erased and lost, any waiting thread is resumed with status `MSG_RESET`.

Note

A reset operation can be used by a low level driver in order to obtain immediate attention from the high level layers.

Parameters

in	*iqp*	pointer to an `input_queue_t` structure

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.12.4.3 msg_t iqPutI (input_queue_t ∗ *iqp*, uint8_t *b*)

Input queue write.

A byte value is written into the low end of an input queue.

Parameters

in	*iqp*	pointer to an `input_queue_t` structure
in	*b*	the byte value to be written in the queue

Returns

The operation status.

Return values

MSG_OK	if the operation has been completed with success.
MSG_TIMEOUT	if the queue is full and the operation cannot be completed.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.12.4.4 msg_t iqGetTimeout (input_queue_t ∗ *iqp*, systime_t *timeout*)

Input queue read with timeout.

This function reads a byte value from an input queue. If the queue is empty then the calling thread is suspended until a byte arrives in the queue or a timeout occurs.

Note

The callback is invoked after removing a character from the queue.

Parameters

`in`	*iqp*	pointer to an `input_queue_t` structure
`in`	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

A byte value from the queue.

Return values

MSG_TIMEOUT	if the specified time expired.
MSG_RESET	if the queue has been reset.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.12.4.5 size_t iqReadTimeout (input_queue_t ∗ *iqp*, uint8_t ∗ *bp*, size_t *n*, systime_t *timeout*)

Input queue read with timeout.

The function reads data from an input queue into a buffer. The operation completes when the specified amount of data has been transferred or after the specified timeout or if the queue has been reset.

Note

The function is not atomic, if you need atomicity it is suggested to use a semaphore or a mutex for mutual exclusion.
The callback is invoked after removing each character from the queue.

Parameters

`in`	*iqp*	pointer to an `input_queue_t` structure

Parameters

out	*bp*	pointer to the data buffer
in	*n*	the maximum amount of data to be transferred, the value 0 is reserved
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The number of bytes effectively transferred.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.12.4.6 void oqObjectInit (output_queue_t ∗ *oqp,* uint8_t ∗ *bp,* size_t *size,* qnotify_t *onfy,* void ∗ *link*)

Initializes an output queue.

A Semaphore is internally initialized and works as a counter of the free bytes in the queue.

Note

The callback is invoked from within the S-Locked system state.

Parameters

out	*oqp*	pointer to an `output_queue_t` structure
in	*bp*	pointer to a memory area allocated as queue buffer
in	*size*	size of the queue buffer
in	*onfy*	pointer to a callback function that is invoked when data is written to the queue. The value can be `NULL`.
in	*link*	application defined pointer

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.12.4.7 void oqResetI (output_queue_t ∗ *oqp*)

Resets an output queue.

All the data in the output queue is erased and lost, any waiting thread is resumed with status `MSG_RESET`.

Note

A reset operation can be used by a low level driver in order to obtain immediate attention from the high level layers.

Parameters

in	*oqp*	pointer to an `output_queue_t` structure

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.12.4.8 msg_t oqPutTimeout (output_queue_t ∗ *oqp*, uint8_t *b*, systime_t *timeout*)

Output queue write with timeout.

This function writes a byte value to an output queue. If the queue is full then the calling thread is suspended until there is space in the queue or a timeout occurs.

Note

The callback is invoked after putting the character into the queue.

Parameters

in	*oqp*	pointer to an `output_queue_t` structure
in	*b*	the byte value to be written in the queue
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The operation status.

Return values

MSG_OK	if the operation succeeded.
MSG_TIMEOUT	if the specified time expired.
MSG_RESET	if the queue has been reset.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.12.4.9 msg_t oqGetI (output_queue_t ∗ *oqp*)

Output queue read.

A byte value is read from the low end of an output queue.

Parameters

in	*oqp*	pointer to an `output_queue_t` structure

Returns

The byte value from the queue.

Return values

MSG_TIMEOUT	if the queue is empty.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.12.4.10 size_t oqWriteTimeout (output_queue_t ∗ *oqp,* const uint8_t ∗ *bp,* size_t *n,* systime_t *timeout*)

Output queue write with timeout.

The function writes data from a buffer to an output queue. The operation completes when the specified amount of data has been transferred or after the specified timeout or if the queue has been reset.

Note

The function is not atomic, if you need atomicity it is suggested to use a semaphore or a mutex for mutual exclusion.
The callback is invoked after putting each character into the queue.

Parameters

in	*oqp*	pointer to an `output_queue_t` structure
in	*bp*	pointer to the data buffer
in	*n*	the maximum amount of data to be transferred, the value 0 is reserved
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The number of bytes effectively transferred.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.13 Abstract Streams

7.13.1 Detailed Description

This module define an abstract interface for generic data streams. Note that no code is present, just abstract interfaces-like structures, you should look at the system as to a set of abstract C++ classes (even if written in C). This system has then advantage to make the access to data streams independent from the implementation logic.
The stream interface can be used as base class for high level object types such as files, sockets, serial ports, pipes etc.

Macros

- #define _base_sequential_stream_methods

  BaseSequentialStream specific methods.
- #define _base_sequential_stream_data

  `BaseSequentialStream` *specific data.*

Streams return codes

- #define **STM_OK** MSG_OK
- #define **STM_TIMEOUT** MSG_TIMEOUT
- #define **STM_RESET** MSG_RESET

Macro Functions (BaseSequentialStream)

- #define streamWrite(ip, bp, n) ((ip)->vmt->write(ip, bp, n))

  Sequential Stream write.
- #define streamRead(ip, bp, n) ((ip)->vmt->read(ip, bp, n))

  Sequential Stream read.
- #define streamPut(ip, b) ((ip)->vmt->put(ip, b))

  Sequential Stream blocking byte write.
- #define streamGet(ip) ((ip)->vmt->get(ip))

  Sequential Stream blocking byte read.

Data Structures

- struct BaseSequentialStreamVMT

  `BaseSequentialStream` *virtual methods table.*
- struct BaseSequentialStream

  Base stream class.

7.13.2 Macro Definition Documentation

7.13.2.1 #define _base_sequential_stream_methods

Value:

```
/* Stream write buffer method.*/                                         \
  size_t (*write)(void *instance, const uint8_t *bp, size_t n);           \
  /* Stream read buffer method.*/                                          \
  size_t (*read)(void *instance, uint8_t *bp, size_t n);                   \
  /* Channel put method, blocking.*/                                       \
  msg_t (*put)(void *instance, uint8_t b);                                 \
  /* Channel get method, blocking.*/                                       \
  msg_t (*get)(void *instance);                                            \
```

BaseSequentialStream specific methods.

7.13.2.2 #define _base_sequential_stream_data

`BaseSequentialStream` specific data.

Note

It is empty because `BaseSequentialStream` is only an interface without implementation.

7.13.2.3 #define streamWrite(*ip*, *bp*, *n*) ((ip)->vmt->write(ip, bp, n))

Sequential Stream write.

The function writes data from a buffer to a stream.

Parameters

`in`	*ip*	pointer to a `BaseSequentialStream` or derived class
`in`	*bp*	pointer to the data buffer
`in`	*n*	the maximum amount of data to be transferred

Returns

The number of bytes transferred. The return value can be less than the specified number of bytes if an end-of-file condition has been met.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.13.2.4 #define streamRead(*ip*, *bp*, *n*) ((ip)->vmt->read(ip, bp, n))

Sequential Stream read.

The function reads data from a stream into a buffer.

Parameters

`in`	*ip*	pointer to a `BaseSequentialStream` or derived class
`out`	*bp*	pointer to the data buffer
`in`	*n*	the maximum amount of data to be transferred

Returns

The number of bytes transferred. The return value can be less than the specified number of bytes if an end-of-file condition has been met.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.13.2.5 #define streamPut(*ip*, *b*) ((ip)->vmt->put(ip, b))

Sequential Stream blocking byte write.

This function writes a byte value to a channel. If the channel is not ready to accept data then the calling thread is suspended.

Parameters

in	*ip*	pointer to a BaseChannel or derived class
in	*b*	the byte value to be written to the channel

Returns

The operation status.

Return values

STM_OK	if the operation succeeded.
STM_RESET	if an end-of-file condition has been met.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.13.2.6 #define streamGet(*ip*) ((ip)->vmt->get(ip))

Sequential Stream blocking byte read.

This function reads a byte value from a channel. If the data is not available then the calling thread is suspended.

Parameters

in	*ip*	pointer to a BaseChannel or derived class

Returns

A byte value from the queue.

Return values

STM_RESET	if an end-of-file condition has been met.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.14 I2C Driver

Generic I2C Driver.

7.14.1 Detailed Description

Generic I2C Driver.

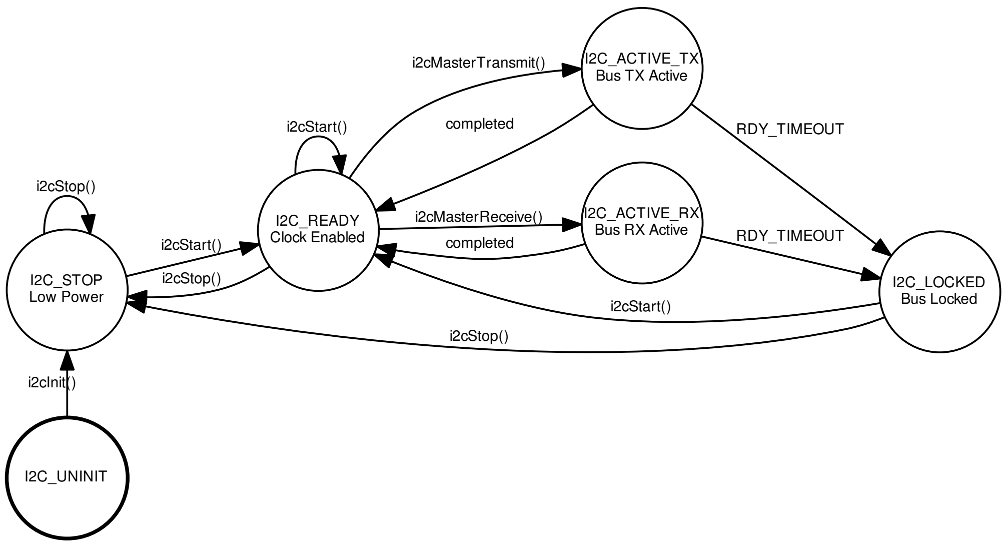
This module implements a generic I2C (Inter-Integrated Circuit) driver.

Precondition

In order to use the I2C driver the `HAL_USE_I2C` option must be enabled in `halconf.h`.

7.14.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



The driver is not thread safe for performance reasons, if you need to access the I2C bus from multiple threads then use the `i2cAcquireBus()` and `i2cReleaseBus()` APIs in order to gain exclusive access.

Macros

- #define I2C_USE_MUTUAL_EXCLUSION TRUE

  Enables the mutual exclusion APIs on the I2C bus.
- #define _i2c_wakeup_isr(i2cp)

  Wakes up the waiting thread notifying no errors.
- #define _i2c_wakeup_error_isr(i2cp)

Wakes up the waiting thread notifying errors.
- #define i2cMasterTransmit(i2cp, addr, txbuf, txbytes, rxbuf, rxbytes)

  Wrap i2cMasterTransmitTimeout function with TIME_INFINITE timeout.
- #define i2cMasterReceive(i2cp, addr, rxbuf, rxbytes) (i2cMasterReceiveTimeout(i2cp, addr, rxbuf, rxbytes, TIME_INFINITE))

  Wrap i2cMasterReceiveTimeout function with TIME_INFINITE timeout.
- #define i2c_lld_get_errors(i2cp) ((i2cp)->errors)

  Get errors from I2C driver.

I2C bus error conditions

- #define I2C_NO_ERROR 0x00

  No error.
- #define I2C_BUS_ERROR 0x01

  Bus Error.
- #define I2C_ARBITRATION_LOST 0x02

  Arbitration Lost.
- #define I2C_ACK_FAILURE 0x04

  Acknowledge Failure.
- #define I2C_OVERRUN 0x08

  Overrun/Underrun.
- #define I2C_PEC_ERROR 0x10

  PEC Error in reception.
- #define I2C_TIMEOUT 0x20

  Hardware timeout.
- #define I2C_SMB_ALERT 0x40

  SMBus Alert.

PLATFORM configuration options

- #define PLATFORM_I2C_USE_I2C1 FALSE

  I2C1 driver enable switch.

Typedefs

- typedef uint16_t i2caddr_t

  Type representing an I2C address.
- typedef uint32_t i2cflags_t

  Type of I2C Driver condition flags.
- typedef struct I2CDriver I2CDriver

  Type of a structure representing an I2C driver.

Data Structures

- struct I2CConfig

  Type of I2C driver configuration structure.
- struct I2CDriver

  Structure representing an I2C driver.

Functions

- void i2cInit (void)

  I2C Driver initialization.
- void i2cObjectInit (I2CDriver ∗i2cp)

  Initializes the standard part of a `I2CDriver` structure.
- void i2cStart (I2CDriver ∗i2cp, const I2CConfig ∗config)

  Configures and activates the I2C peripheral.
- void i2cStop (I2CDriver ∗i2cp)

  Deactivates the I2C peripheral.
- i2cflags_t i2cGetErrors (I2CDriver ∗i2cp)

  Returns the errors mask associated to the previous operation.
- msg_t i2cMasterTransmitTimeout (I2CDriver ∗i2cp, i2caddr_t addr, const uint8_t ∗txbuf, size_t txbytes, uint8_t ∗rxbuf, size_t rxbytes, systime_t timeout)

  Sends data via the I2C bus.
- msg_t i2cMasterReceiveTimeout (I2CDriver ∗i2cp, i2caddr_t addr, uint8_t ∗rxbuf, size_t rxbytes, systime_t timeout)

  Receives data from the I2C bus.
- void i2cAcquireBus (I2CDriver ∗i2cp)

  Gains exclusive access to the I2C bus.
- void i2cReleaseBus (I2CDriver ∗i2cp)

  Releases exclusive access to the I2C bus.
- void i2c_lld_init (void)

  Low level I2C driver initialization.
- void i2c_lld_start (I2CDriver ∗i2cp)

  Configures and activates the I2C peripheral.
- void i2c_lld_stop (I2CDriver ∗i2cp)

  Deactivates the I2C peripheral.
- msg_t i2c_lld_master_receive_timeout (I2CDriver ∗i2cp, i2caddr_t addr, uint8_t ∗rxbuf, size_t rxbytes, systime_t timeout)

  Receives data via the I2C bus as master.
- msg_t i2c_lld_master_transmit_timeout (I2CDriver ∗i2cp, i2caddr_t addr, const uint8_t ∗txbuf, size_t txbytes, uint8_t ∗rxbuf, size_t rxbytes, systime_t timeout)

  Transmits data via the I2C bus as master.

Enumerations

Variables

- I2CDriver I2CD1

  I2C1 driver identifier.

7.14.3 Macro Definition Documentation

7.14.3.1 #define I2C_NO_ERROR 0x00

No error.

7.14.3.2 #define I2C_BUS_ERROR 0x01

Bus Error.

7.14.3.3 #define I2C_ARBITRATION_LOST 0x02

Arbitration Lost.

7.14.3.4 #define I2C_ACK_FAILURE 0x04

Acknowledge Failure.

7.14.3.5 #define I2C_OVERRUN 0x08

Overrun/Underrun.

7.14.3.6 #define I2C_PEC_ERROR 0x10

PEC Error in reception.

7.14.3.7 #define I2C_TIMEOUT 0x20

Hardware timeout.

7.14.3.8 #define I2C_SMB_ALERT 0x40

SMBus Alert.

7.14.3.9 #define I2C_USE_MUTUAL_EXCLUSION TRUE

Enables the mutual exclusion APIs on the I2C bus.

7.14.3.10 #define _i2c_wakeup_isr(*i2cp*)

Value:

```
do {                                              \
  osalSysLockFromISR();                                                      \
  osalThreadResumeI(&(i2cp)->thread, MSG_OK);                                \
  osalSysUnlockFromISR();                                                    \
} while(0)
```

Wakes up the waiting thread notifying no errors.

Parameters

in	*i2cp*	pointer to the `I2CDriver` object

Function Class:

Not an API, this function is for internal use only.

7.14.3.11 #define _i2c_wakeup_error_isr(*i2cp*)

Value:

```
do {                                    \
  osalSysLockFromISR();                                                     \
  osalThreadResumeI(&(i2cp)->thread, MSG_RESET);                            \
  osalSysUnlockFromISR();                                                   \
} while(0)
```

Wakes up the waiting thread notifying errors.

Parameters

in	*i2cp*	pointer to the `I2CDriver` object

Function Class:

Not an API, this function is for internal use only.

7.14.3.12 #define i2cMasterTransmit(*i2cp, addr, txbuf, txbytes, rxbuf, rxbytes*)

Value:

```
(i2cMasterTransmitTimeout(i2cp, addr, txbuf, txbytes, rxbuf, rxbytes,     \
                            TIME_INFINITE))
```

Wrap i2cMasterTransmitTimeout function with TIME_INFINITE timeout.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.14.3.13 #define i2cMasterReceive(*i2cp, addr, rxbuf, rxbytes*) (i2cMasterReceiveTimeout(i2cp, addr, rxbuf, rxbytes, TIME_INFINITE))

Wrap i2cMasterReceiveTimeout function with TIME_INFINITE timeout.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.14.3.14 #define PLATFORM_I2C_USE_I2C1 FALSE

I2C1 driver enable switch.

If set to `TRUE` the support for I2C1 is included.

Note

The default is `FALSE`.

7.14.3.15 #define i2c_lld_get_errors(*i2cp*) ((i2cp)->errors)

Get errors from I2C driver.

Parameters

in	*i2cp*	pointer to the `I2CDriver` object

Function Class:

Not an API, this function is for internal use only.

7.14.4 Typedef Documentation

7.14.4.1 typedef uint16_t **i2caddr_t**

Type representing an I2C address.

7.14.4.2 typedef uint32_t **i2cflags_t**

Type of I2C Driver condition flags.

7.14.4.3 typedef struct **I2CDriver I2CDriver**

Type of a structure representing an I2C driver.

7.14.5 Enumeration Type Documentation

7.14.5.1 enum **i2cstate_t**

Driver state machine possible states.

Enumerator

I2C_UNINIT Not initialized.
I2C_STOP Stopped.
I2C_READY Ready.
I2C_ACTIVE_TX Transmitting.
I2C_ACTIVE_RX Receiving.

7.14.6 Function Documentation

7.14.6.1 void i2cInit (void)

I2C Driver initialization.

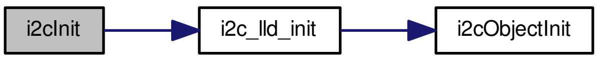
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



7.14.6.2 void i2cObjectInit (**I2CDriver** ∗ *i2cp*)

Initializes the standard part of a `I2CDriver` structure.

Parameters

out	*i2cp*	pointer to the `I2CDriver` object

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.14.6.3 void i2cStart (**I2CDriver** ∗ *i2cp*, const **I2CConfig** ∗ *config*)

Configures and activates the I2C peripheral.

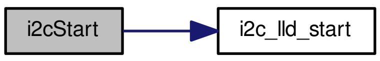
Parameters

in	*i2cp*	pointer to the `I2CDriver` object
in	*config*	pointer to the `I2CConfig` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.14.6.4 void i2cStop (**I2CDriver** ∗ *i2cp*)

Deactivates the I2C peripheral.

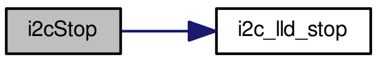
Parameters

in	*i2cp*	pointer to the `I2CDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.14.6.5 i2cflags_t i2cGetErrors (I2CDriver ∗ *i2cp*)

Returns the errors mask associated to the previous operation.

Parameters

in	*i2cp*	pointer to the I2CDriver object

Returns

The errors mask.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.14.6.6 msg_t i2cMasterTransmitTimeout (I2CDriver ∗ *i2cp*, i2caddr_t *addr*, const uint8_t ∗ *txbuf*, size_t *txbytes*, uint8_t ∗ *rxbuf*, size_t *rxbytes*, systime_t *timeout*)

Sends data via the I2C bus.

Function designed to realize "read-through-write" transfer paradigm. If you want transmit data without any further read, than set **rxbytes** field to 0.

Parameters

in	*i2cp*	pointer to the I2CDriver object
in	*addr*	slave device address (7 bits) without R/W bit
in	*txbuf*	pointer to transmit buffer
in	*txbytes*	number of bytes to be transmitted
out	*rxbuf*	pointer to receive buffer
in	*rxbytes*	number of bytes to be received, set it to 0 if you want transmit only
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_INFINITE* no timeout.

Returns

The operation status.

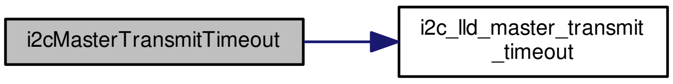
Return values

MSG_OK	if the function succeeded.
MSG_RESET	if one or more I2C errors occurred, the errors can be retrieved using `i2cGetErrors()`.
MSG_TIMEOUT	if a timeout occurred before operation end.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.14.6.7 msg_t i2cMasterReceiveTimeout (I2CDriver ∗ *i2cp*, i2caddr_t *addr*, uint8_t ∗ *rxbuf*, size_t *rxbytes*, systime_t *timeout*)

Receives data from the I2C bus.

Parameters

`in`	*i2cp*	pointer to the `I2CDriver` object
`in`	*addr*	slave device address (7 bits) without R/W bit
`out`	*rxbuf*	pointer to receive buffer
`in`	*rxbytes*	number of bytes to be received
`in`	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_INFINITE* no timeout.

Returns

The operation status.

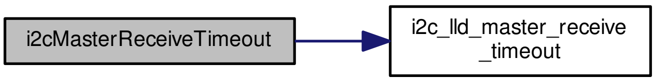
Return values

MSG_OK	if the function succeeded.
MSG_RESET	if one or more I2C errors occurred, the errors can be retrieved using `i2cGetErrors()`.
MSG_TIMEOUT	if a timeout occurred before operation end.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.14.6.8 void i2cAcquireBus (**I2CDriver** ∗ *i2cp*)

Gains exclusive access to the I2C bus.

This function tries to gain ownership to the I2C bus, if the bus is already being used then the invoking thread is queued.

Precondition

In order to use this function the option `I2C_USE_MUTUAL_EXCLUSION` must be enabled.

Parameters

`in`	*i2cp*	pointer to the `I2CDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.14.6.9 void i2cReleaseBus (**I2CDriver** ∗ *i2cp*)

Releases exclusive access to the I2C bus.

Precondition

In order to use this function the option `I2C_USE_MUTUAL_EXCLUSION` must be enabled.

Parameters

`in`	*i2cp*	pointer to the `I2CDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

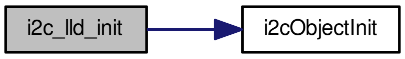
7.14.6.10 void i2c_lld_init (void)

Low level I2C driver initialization.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.14.6.11 void i2c_lld_start (**I2CDriver** ∗ *i2cp*)

Configures and activates the I2C peripheral.

Parameters

`in`	*i2cp*	pointer to the `I2CDriver` object

Function Class:

Not an API, this function is for internal use only.

7.14.6.12 void i2c_lld_stop (**I2CDriver** ∗ *i2cp*)

Deactivates the I2C peripheral.

Parameters

`in`	*i2cp*	pointer to the `I2CDriver` object

Function Class:

Not an API, this function is for internal use only.

7.14.6.13 msg_t i2c_lld_master_receive_timeout (**I2CDriver** ∗ *i2cp,* **i2caddr_t** *addr,* uint8_t ∗ *rxbuf,* size_t *rxbytes,* systime_t *timeout*)

Receives data via the I2C bus as master.

Parameters

`in`	*i2cp*	pointer to the `I2CDriver` object
`in`	*addr*	slave device address
`out`	*rxbuf*	pointer to the receive buffer
`in`	*rxbytes*	number of bytes to be received

Parameters

`in`	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_INFINITE* no timeout.

Returns

The operation status.

Return values

MSG_OK	if the function succeeded.
MSG_RESET	if one or more I2C errors occurred, the errors can be retrieved using `i2cGetErrors()`.
MSG_TIMEOUT	if a timeout occurred before operation end. **After a timeout the driver must be stopped and restarted because the bus is in an uncertain state**.

Function Class:

Not an API, this function is for internal use only.

7.14.6.14 msg_t i2c_lld_master_transmit_timeout (I2CDriver ∗ *i2cp*, i2caddr_t *addr*, const uint8_t ∗ *txbuf*, size_t *txbytes*, uint8_t ∗ *rxbuf*, size_t *rxbytes*, systime_t *timeout*)

Transmits data via the I2C bus as master.

Parameters

`in`	*i2cp*	pointer to the `I2CDriver` object
`in`	*addr*	slave device address
`in`	*txbuf*	pointer to the transmit buffer
`in`	*txbytes*	number of bytes to be transmitted
`out`	*rxbuf*	pointer to the receive buffer
`in`	*rxbytes*	number of bytes to be received
`in`	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_INFINITE* no timeout.

Returns

The operation status.

Return values

MSG_OK	if the function succeeded.
MSG_RESET	if one or more I2C errors occurred, the errors can be retrieved using `i2cGetErrors()`.
MSG_TIMEOUT	if a timeout occurred before operation end. **After a timeout the driver must be stopped and restarted because the bus is in an uncertain state**.

Function Class:

Not an API, this function is for internal use only.

7.14.7 Variable Documentation

7.14.7.1 I2CDriver I2CD1

I2C1 driver identifier.

7.15 I2S Driver

Generic I2S Driver.

7.15.1 Detailed Description

Generic I2S Driver.

This module implements a generic I2S driver.

Precondition

In order to use the I2S driver the `HAL_USE_I2S` option must be enabled in `halconf.h`.

7.15.2 Driver State Machine

I2S modes

- #define **I2S_MODE_SLAVE** 0
- #define **I2S_MODE_MASTER** 1

Macro Functions

- #define i2sStartExchangeI(i2sp)

  Starts a I2S data exchange.
- #define i2sStopExchangeI(i2sp)

  Stops the ongoing data exchange.
- #define _i2s_isr_half_code(i2sp)

  Common ISR code, half buffer event.
- #define _i2s_isr_full_code(i2sp)

  Common ISR code.

PLATFORM configuration options

- #define PLATFORM_I2S_USE_I2S1 FALSE

  I2SD1 driver enable switch.

Typedefs

- typedef struct I2SDriver I2SDriver

  Type of a structure representing an I2S driver.
- typedef void(∗ i2scallback_t) (I2SDriver ∗i2sp, size_t offset, size_t n)

  I2S notification callback type.

Data Structures

- struct I2SConfig

  Driver configuration structure.
- struct I2SDriver

  Structure representing an I2S driver.

Functions

- void i2sInit (void)

  I2S Driver initialization.
- void i2sObjectInit (I2SDriver ∗i2sp)

  Initializes the standard part of a `I2SDriver` structure.
- void i2sStart (I2SDriver ∗i2sp, const I2SConfig ∗config)

  Configures and activates the I2S peripheral.
- void i2sStop (I2SDriver ∗i2sp)

  Deactivates the I2S peripheral.
- void i2sStartExchange (I2SDriver ∗i2sp)

  Starts a I2S data exchange.
- void i2sStopExchange (I2SDriver ∗i2sp)

  Stops the ongoing data exchange.
- void i2s_lld_init (void)

  Low level I2S driver initialization.
- void i2s_lld_start (I2SDriver ∗i2sp)

  Configures and activates the I2S peripheral.
- void i2s_lld_stop (I2SDriver ∗i2sp)

  Deactivates the I2S peripheral.
- void i2s_lld_start_exchange (I2SDriver ∗i2sp)

  Starts a I2S data exchange.
- void i2s_lld_stop_exchange (I2SDriver ∗i2sp)

  Stops the ongoing data exchange.

Enumerations

Variables

- I2SDriver I2SD1

  I2S2 driver identifier.

7.15.3 Macro Definition Documentation

7.15.3.1 #define i2sStartExchangeI(*i2sp*)

Value:

```
{                                                 \
  i2s_lld_start_exchange(i2sp);                                  \
  (i2sp)->state = I2S_ACTIVE;                                    \
}
```

Starts a I2S data exchange.

Parameters

in	*i2sp*	pointer to the `I2SDriver` object

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.15.3.2 #define i2sStopExchangeI(*i2sp*)

Value:

```
{                                                          \
  i2s_lld_stop_exchange(i2sp);                                         \
  (i2sp)->state = I2S_READY;                                           \
}
```

Stops the ongoing data exchange.

The ongoing data exchange, if any, is stopped, if the driver was not active the function does nothing.

Parameters

in	*i2sp*	pointer to the I2SDriver object

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.15.3.3 #define _i2s_isr_half_code(*i2sp*)

Value:

```
{                                                       \
  if ((i2sp)->config->end_cb != NULL) {                                \
    (i2sp)->config->end_cb(i2sp, 0, (i2sp)->config->size / 2);         \
  }                                                                    \
}
```

Common ISR code, half buffer event.

This code handles the portable part of the ISR code:

- Callback invocation.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

in	*i2sp*	pointer to the I2CDriver object

Function Class:

Not an API, this function is for internal use only.

7.15.3.4 #define _i2s_isr_full_code(*i2sp*)

Value:

```
{                                                            \
  if ((i2sp)->config->end_cb) {                                        \
    (i2sp)->state = I2S_COMPLETE;                                      \
    (i2sp)->config->end_cb(i2sp,                                       \
                           (i2sp)->config->size / 2,                   \
                           (i2sp)->config->size / 2);                  \
    if ((i2sp)->state == I2S_COMPLETE)                                 \
      (i2sp)->state = I2S_READY;                                       \
  }                                                                    \
  else                                                                 \
    (i2sp)->state = I2S_READY;                                         \
}
```

Common ISR code.

This code handles the portable part of the ISR code:

- Callback invocation.
- Driver state transitions.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

in	*i2sp*	pointer to the I2CDriver object

Function Class:

Not an API, this function is for internal use only.

7.15.3.5 #define PLATFORM_I2S_USE_I2S1 FALSE

I2SD1 driver enable switch.

If set to TRUE the support for I2S1 is included.

Note

The default is FALSE.

7.15.4 Typedef Documentation

7.15.4.1 typedef struct I2SDriver I2SDriver

Type of a structure representing an I2S driver.

7.15.4.2 typedef void(∗ i2scallback_t) (I2SDriver ∗i2sp, size_t offset, size_t n)

I2S notification callback type.

Parameters

in	*i2sp*	pointer to the I2SDriver object
in	*offset*	offset in buffers of the data to read/write
in	*n*	number of samples to read/write

7.15.5 Enumeration Type Documentation

7.15.5.1 enum i2sstate_t

Driver state machine possible states.

Enumerator

I2S_UNINIT	Not initialized.
I2S_STOP	Stopped.
I2S_READY	Ready.
I2S_ACTIVE	Active.
I2S_COMPLETE	Transmission complete.

7.15.6 Function Documentation

7.15.6.1 void i2sInit (void)

I2S Driver initialization.

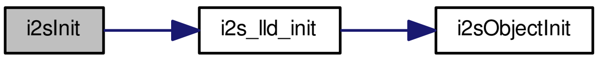
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



7.15.6.2 void i2sObjectInit (I2SDriver ∗ *i2sp*)

Initializes the standard part of a `I2SDriver` structure.

Parameters

`out`	*i2sp*	pointer to the `I2SDriver` object

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.15.6.3 void i2sStart (I2SDriver ∗ *i2sp*, const I2SConfig ∗ *config*)

Configures and activates the I2S peripheral.

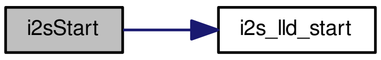
Parameters

in	*i2sp*	pointer to the `I2SDriver` object
in	*config*	pointer to the `I2SConfig` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.15.6.4 void i2sStop (I2SDriver ∗ *i2sp*)

Deactivates the I2S peripheral.

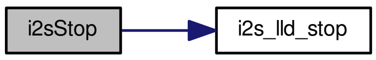
Parameters

in	*i2sp*	pointer to the `I2SDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.15.6.5 void i2sStartExchange (I2SDriver ∗ *i2sp*)

Starts a I2S data exchange.

Parameters

in	*i2sp*	pointer to the `I2SDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.15.6.6 void i2sStopExchange (**I2SDriver** ∗ *i2sp*)

Stops the ongoing data exchange.

The ongoing data exchange, if any, is stopped, if the driver was not active the function does nothing.

Parameters

in	*i2sp*	pointer to the `I2SDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

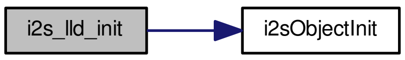
7.15.6.7 void i2s_lld_init (void)

Low level I2S driver initialization.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.15.6.8 void i2s_lld_start (**I2SDriver** ∗ *i2sp*)

Configures and activates the I2S peripheral.

Parameters

in	*i2sp*	pointer to the `I2SDriver` object

Function Class:

Not an API, this function is for internal use only.

7.15.6.9 void i2s_lld_stop (I2SDriver ∗ *i2sp*)

Deactivates the I2S peripheral.

Parameters

`in`	*i2sp*	pointer to the `I2SDriver` object

Function Class:

Not an API, this function is for internal use only.

7.15.6.10 void i2s_lld_start_exchange (I2SDriver ∗ *i2sp*)

Starts a I2S data exchange.

Parameters

`in`	*i2sp*	pointer to the `I2SDriver` object

Function Class:

Not an API, this function is for internal use only.

7.15.6.11 void i2s_lld_stop_exchange (I2SDriver ∗ *i2sp*)

Stops the ongoing data exchange.

The ongoing data exchange, if any, is stopped, if the driver was not active the function does nothing.

Parameters

`in`	*i2sp*	pointer to the `I2SDriver` object

Function Class:

Not an API, this function is for internal use only.

7.15.7 Variable Documentation

7.15.7.1 I2SDriver I2SD1

I2S2 driver identifier.

7.16 ICU Driver

Generic ICU Driver.

7.16.1 Detailed Description

Generic ICU Driver.

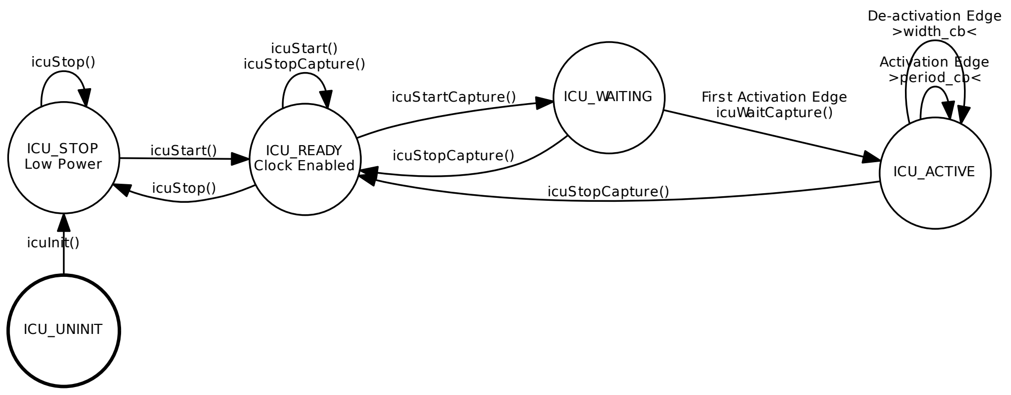
This module implements a generic ICU (Input Capture Unit) driver. The purpose of the driver is to measure period and duty cycle of an input digital signal (PWM input).

Precondition

In order to use the ICU driver the `HAL_USE_ICU` option must be enabled in `halconf.h`.

7.16.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



7.16.3 ICU Operations.

This driver abstracts a generic Input Capture Unit composed of:

- A clock prescaler.
- A main up counter.
- Two capture registers triggered by the rising and falling edges on the sampled input.

The ICU unit can be programmed to synchronize on the rising or falling edge of the sample input:

- **ICU_INPUT_ACTIVE_HIGH**, a rising edge is the start signal.

- **ICU_INPUT_ACTIVE_LOW**, a falling edge is the start signal.

Callbacks are optionally invoked when:

- On the PWM de-activation edge.
- On the PWM activation edge, measurements for the previous cycle are available from this callback and can be retrieved using `icuGetPeriodX()` and `icuGetWidthX()`.

Macros

- #define icu_lld_get_width(icup) 0

  Returns the width of the latest pulse.
- #define icu_lld_get_period(icup) 0

  Returns the width of the latest cycle.
- #define icu_lld_are_notifications_enabled(icup) false

  Check on notifications status.

Macro Functions

- #define icuStartCaptureI(icup)

  Starts the input capture.
- #define icuStopCaptureI(icup)

  Stops the input capture.
- #define icuEnableNotificationsI(icup) icu_lld_enable_notifications(icup)

  Enables notifications.
- #define icuDisableNotificationsI(icup) icu_lld_disable_notifications(icup)

  Disables notifications.
- #define icuAreNotificationsEnabledX(icup) icu_lld_are_notifications_enabled(icup)

  Check on notifications status.
- #define icuGetWidthX(icup) icu_lld_get_width(icup)

  Returns the width of the latest pulse.
- #define icuGetPeriodX(icup) icu_lld_get_period(icup)

  Returns the width of the latest cycle.

Low level driver helper macros

- #define _icu_isr_invoke_width_cb(icup)

  Common ISR code, ICU width event.
- #define _icu_isr_invoke_period_cb(icup)

  Common ISR code, ICU period event.
- #define _icu_isr_invoke_overflow_cb(icup)

  Common ISR code, ICU timer overflow event.

PLATFORM configuration options

- #define PLATFORM_ICU_USE_ICU1 FALSE

  ICUD1 driver enable switch.

Typedefs

- typedef struct ICUDriver ICUDriver

  Type of a structure representing an ICU driver.
- typedef void(∗ icucallback_t) (ICUDriver ∗icup)

  ICU notification callback type.
- typedef uint32_t icufreq_t

  ICU frequency type.
- typedef uint32_t icucnt_t

  ICU counter type.

Data Structures

- struct ICUConfig

  Driver configuration structure.
- struct ICUDriver

  Structure representing an ICU driver.

Functions

- void icuInit (void)

  ICU Driver initialization.
- void icuObjectInit (ICUDriver ∗icup)

  Initializes the standard part of a `ICUDriver` structure.
- void icuStart (ICUDriver ∗icup, const ICUConfig ∗config)

  Configures and activates the ICU peripheral.
- void icuStop (ICUDriver ∗icup)

  Deactivates the ICU peripheral.
- void icuStartCapture (ICUDriver ∗icup)

  Starts the input capture.
- bool icuWaitCapture (ICUDriver ∗icup)

  Waits for a completed capture.
- void icuStopCapture (ICUDriver ∗icup)

  Stops the input capture.
- void icuEnableNotifications (ICUDriver ∗icup)

  Enables notifications.
- void icuDisableNotifications (ICUDriver ∗icup)

  Disables notifications.
- void icu_lld_init (void)

  Low level ICU driver initialization.
- void icu_lld_start (ICUDriver ∗icup)

  Configures and activates the ICU peripheral.
- void icu_lld_stop (ICUDriver ∗icup)

  Deactivates the ICU peripheral.
- void icu_lld_start_capture (ICUDriver ∗icup)

  Starts the input capture.
- bool icu_lld_wait_capture (ICUDriver ∗icup)

  Waits for a completed capture.
- void icu_lld_stop_capture (ICUDriver ∗icup)

  Stops the input capture.

- void icu_lld_enable_notifications (ICUDriver ∗icup)

  Enables notifications.
- void icu_lld_disable_notifications (ICUDriver ∗icup)

  Disables notifications.

Enumerations

Variables

- ICUDriver ICUD1

  ICUD1 driver identifier.

7.16.4 Macro Definition Documentation

7.16.4.1 #define icuStartCaptureI(*icup*)

Value:

```
do {                                                \
  icu_lld_start_capture(icup);                                              \
  (icup)->state = ICU_WAITING;                                              \
} while (false)
```

Starts the input capture.

Parameters

in	*icup*	pointer to the `ICUDriver` object

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.16.4.2 #define icuStopCaptureI(*icup*)

Value:

```
do {                                                 \
  icu_lld_stop_capture(icup);                                               \
  (icup)->state = ICU_READY;                                                \
} while (false)
```

Stops the input capture.

Parameters

in	*icup*	pointer to the `ICUDriver` object

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.16.4.3 #define icuEnableNotificationsI(*icup*) **icu_lld_enable_notifications**(icup)

Enables notifications.

Precondition

The ICU unit must have been activated using `icuStart()`.

Note

If the notification is already enabled then the call has no effect.

Parameters

`in`	*icup*	pointer to the `ICUDriver` object

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.16.4.4 #define icuDisableNotificationsI(*icup*) **icu_lld_disable_notifications**(icup)

Disables notifications.

Precondition

The ICU unit must have been activated using `icuStart()`.

Note

If the notification is already disabled then the call has no effect.

Parameters

`in`	*icup*	pointer to the `ICUDriver` object

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.16.4.5 #define icuAreNotificationsEnabledX(*icup*) **icu_lld_are_notifications_enabled**(icup)

Check on notifications status.

Parameters

`in`	*icup*	pointer to the `ICUDriver` object

Returns

The notifications status.

Return values

false	if notifications are not enabled.
true	if notifications are enabled.

Function Class:

Not an API, this function is for internal use only.

7.16.4.6 #define icuGetWidthX(*icup*) icu_lld_get_width(icup)

Returns the width of the latest pulse.

The pulse width is defined as number of ticks between the start edge and the stop edge.

Note

This function is meant to be invoked from the width capture callback.

Parameters

`in`	*icup*	pointer to the `ICUDriver` object

Returns

The number of ticks.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

7.16.4.7 #define icuGetPeriodX(*icup*) icu_lld_get_period(icup)

Returns the width of the latest cycle.

The cycle width is defined as number of ticks between a start edge and the next start edge.

Note

This function is meant to be invoked from the width capture callback.

Parameters

`in`	*icup*	pointer to the `ICUDriver` object

Returns

The number of ticks.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

7.16.4.8 #define _icu_isr_invoke_width_cb(*icup*)

Value:

```
do {                                          \
  if (((icup)->state == ICU_ACTIVE) &&                                          \
      ((icup)->config->width_cb != NULL))                                       \
    (icup)->config->width_cb(icup);                                             \
} while (0)
```

Common ISR code, ICU width event.

Parameters

in	*icup*	pointer to the `ICUDriver` object

Function Class:

Not an API, this function is for internal use only.

7.16.4.9 #define _icu_isr_invoke_period_cb(*icup*)

Value:

```
do {                                         \
  if (((icup)->state == ICU_ACTIVE) &&                                          \
      ((icup)->config->period_cb != NULL))                                      \
    (icup)->config->period_cb(icup);                                            \
  (icup)->state = ICU_ACTIVE;                                                   \
} while (0)
```

Common ISR code, ICU period event.

Note

A period event brings the driver into the `ICU_ACTIVE` state.

Parameters

in	*icup*	pointer to the `ICUDriver` object

Function Class:

Not an API, this function is for internal use only.

7.16.4.10 #define _icu_isr_invoke_overflow_cb(*icup*)

Value:

```
do {                                \
  (icup)->config->overflow_cb(icup);                                  \
  (icup)->state = ICU_WAITING;                                        \
} while (0)
```

Common ISR code, ICU timer overflow event.

Note

An overflow always brings the driver back to the `ICU_WAITING` state.

Parameters

in	*icup*	pointer to the `ICUDriver` object

Function Class:

Not an API, this function is for internal use only.

7.16.4.11 #define PLATFORM_ICU_USE_ICU1 FALSE

ICUD1 driver enable switch.

If set to `TRUE` the support for ICUD1 is included.

Note

The default is `FALSE`.

7.16.4.12 #define icu_lld_get_width(*icup*) 0

Returns the width of the latest pulse.

The pulse width is defined as number of ticks between the start edge and the stop edge.

Parameters

in	*icup*	pointer to the `ICUDriver` object

Returns

The number of ticks.

Function Class:

Not an API, this function is for internal use only.

7.16.4.13 #define icu_lld_get_period(*icup*) 0

Returns the width of the latest cycle.

The cycle width is defined as number of ticks between a start edge and the next start edge.

Parameters

in	*icup*	pointer to the `ICUDriver` object

Returns

The number of ticks.

Function Class:

Not an API, this function is for internal use only.

7.16.4.14 #define icu_lld_are_notifications_enabled(*icup*) false

Check on notifications status.

Parameters

in	*icup*	pointer to the `ICUDriver` object

Returns

The notifications status.

Return values

false	if notifications are not enabled.
true	if notifications are enabled.

Function Class:

Not an API, this function is for internal use only.

7.16.5 Typedef Documentation

7.16.5.1 typedef struct **ICUDriver** **ICUDriver**

Type of a structure representing an ICU driver.

7.16.5.2 typedef void(∗ icucallback_t) (**ICUDriver** ∗icup)

ICU notification callback type.

Parameters

in	*icup*	pointer to a `ICUDriver` object

7.16.5.3 typedef uint32_t **icufreq_t**

ICU frequency type.

7.16.5.4 typedef uint32_t **icucnt_t**

ICU counter type.

7.16.6 Enumeration Type Documentation

7.16.6.1 enum **icustate_t**

Driver state machine possible states.

Enumerator

ICU_UNINIT Not initialized.

ICU_STOP Stopped.

ICU_READY Ready.

ICU_WAITING Waiting for first front.

ICU_ACTIVE First front detected.

7.16.6.2 enum **icumode_t**

ICU driver mode.

Enumerator

ICU_INPUT_ACTIVE_HIGH Trigger on rising edge.

ICU_INPUT_ACTIVE_LOW Trigger on falling edge.

7.16.7 Function Documentation

7.16.7.1 void icuInit (void)

ICU Driver initialization.

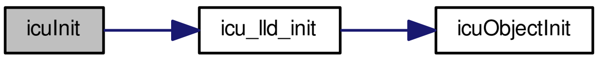
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



7.16.7.2 void icuObjectInit (**ICUDriver** * *icup*)

Initializes the standard part of a `ICUDriver` structure.

Parameters

out	*icup*	pointer to the `ICUDriver` object

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.16.7.3 void icuStart (ICUDriver ∗ *icup*, const ICUConfig ∗ *config*)

Configures and activates the ICU peripheral.

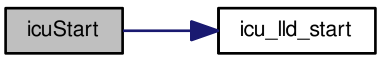
Parameters

in	*icup*	pointer to the `ICUDriver` object
in	*config*	pointer to the `ICUConfig` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.16.7.4 void icuStop (ICUDriver ∗ *icup*)

Deactivates the ICU peripheral.

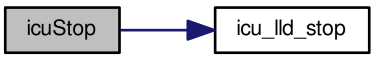
Parameters

in	*icup*	pointer to the `ICUDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.16.7.5 void icuStartCapture (**ICUDriver** ∗ *icup*)

Starts the input capture.

Parameters

in	*icup*	pointer to the `ICUDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.16.7.6 bool icuWaitCapture (**ICUDriver** ∗ *icup*)

Waits for a completed capture.

Note

The operation could be performed in polled mode depending on.
In order to use this function notifications must be disabled.

Precondition

The driver must be in `ICU_WAITING` or `ICU_ACTIVE` states.

Postcondition

After the capture is available the driver is in `ICU_ACTIVE` state. If a capture fails then the driver is in `ICU↩_WAITING` state.

Parameters

in	*icup*	pointer to the `ICUDriver` object

Returns

The capture status.

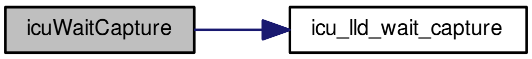
Return values

false	if the capture is successful.
true	if a timer overflow occurred.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.16.7.7 void icuStopCapture (ICUDriver * *icup*)

Stops the input capture.

Parameters

`in`	*icup*	pointer to the `ICUDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.16.7.8 void icuEnableNotifications (ICUDriver * *icup*)

Enables notifications.

Precondition

The ICU unit must have been activated using `icuStart()`.

Note

If the notification is already enabled then the call has no effect.

Parameters

`in`	*icup*	pointer to the `ICUDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.16.7.9 void icuDisableNotifications (**ICUDriver** ∗ *icup*)

Disables notifications.

Precondition

The ICU unit must have been activated using `icuStart()`.

Note

If the notification is already disabled then the call has no effect.

Parameters

`in`	*icup*	pointer to the `ICUDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

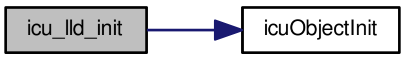
7.16.7.10 void icu_lld_init (void)

Low level ICU driver initialization.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.16.7.11 void icu_lld_start (**ICUDriver** ∗ *icup*)

Configures and activates the ICU peripheral.

Parameters

`in`	*icup*	pointer to the `ICUDriver` object

Function Class:

Not an API, this function is for internal use only.

7.16.7.12 void icu_lld_stop (**ICUDriver** * *icup*)

Deactivates the ICU peripheral.

Parameters

in	*icup*	pointer to the `ICUDriver` object

Function Class:

Not an API, this function is for internal use only.

7.16.7.13 void icu_lld_start_capture (**ICUDriver** * *icup*)

Starts the input capture.

Parameters

in	*icup*	pointer to the `ICUDriver` object

Function Class:

Not an API, this function is for internal use only.

7.16.7.14 bool icu_lld_wait_capture (**ICUDriver** * *icup*)

Waits for a completed capture.

Note

The operation is performed in polled mode.
In order to use this function notifications must be disabled.

Parameters

in	*icup*	pointer to the `ICUDriver` object

Returns

The capture status.

Return values

false	if the capture is successful.
true	if a timer overflow occurred.

Function Class:

Not an API, this function is for internal use only.

7.16.7.15 void icu_lld_stop_capture (ICUDriver ∗ *icup*)

Stops the input capture.

Parameters

in	*icup*	pointer to the ICUDriver object

Function Class:

Not an API, this function is for internal use only.

7.16.7.16 void icu_lld_enable_notifications (ICUDriver ∗ *icup*)

Enables notifications.

Precondition

The ICU unit must have been activated using `icuStart()`.

Note

If the notification is already enabled then the call has no effect.

Parameters

in	*icup*	pointer to the ICUDriver object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.16.7.17 void icu_lld_disable_notifications (ICUDriver ∗ *icup*)

Disables notifications.

Precondition

The ICU unit must have been activated using `icuStart()`.

Note

If the notification is already disabled then the call has no effect.

Parameters

in	*icup*	pointer to the ICUDriver object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.16.8 Variable Documentation

7.16.8.1 ICUDriver ICUD1

ICUD1 driver identifier.

Note

The driver ICUD1 allocates the complex timer TIM1 when enabled.

7.17 MAC Driver

Generic MAC Driver.

7.17.1 Detailed Description

Generic MAC Driver.

This module implements a generic MAC (Media Access Control) driver for Ethernet controllers.

Precondition

In order to use the MAC driver the `HAL_USE_MAC` option must be enabled in `halconf.h`.

Macros

- #define MAC_SUPPORTS_ZERO_COPY TRUE

  This implementation supports the zero-copy mode API.

MAC configuration options

- #define MAC_USE_ZERO_COPY FALSE

  Enables an event sources for incoming packets.
- #define MAC_USE_EVENTS TRUE

  Enables an event sources for incoming packets.

Macro Functions

- #define macGetReceiveEventSource(macp) (&(macp)->rdevent)

  Returns the received frames event source.
- #define macWriteTransmitDescriptor(tdp, buf, size) mac_lld_write_transmit_descriptor(tdp, buf, size)

  Writes to a transmit descriptor's stream.
- #define macReadReceiveDescriptor(rdp, buf, size) mac_lld_read_receive_descriptor(rdp, buf, size)

  Reads from a receive descriptor's stream.
- #define macGetNextTransmitBuffer(tdp, size, sizep) mac_lld_get_next_transmit_buffer(tdp, size, sizep)

  Returns a pointer to the next transmit buffer in the descriptor chain.
- #define macGetNextReceiveBuffer(rdp, sizep) mac_lld_get_next_receive_buffer(rdp, sizep)

  Returns a pointer to the next receive buffer in the descriptor chain.

PLATFORM configuration options

- #define PLATFORM_MAC_USE_MAC1 FALSE

  MAC driver enable switch.

Typedefs

- typedef struct MACDriver MACDriver

  Type of a structure representing a MAC driver.

Data Structures

- struct MACConfig

  Driver configuration structure.
- struct MACDriver

  Structure representing a MAC driver.
- struct MACTransmitDescriptor

  Structure representing a transmit descriptor.
- struct MACReceiveDescriptor

  Structure representing a receive descriptor.

Functions

- void macInit (void)

  MAC Driver initialization.
- void macObjectInit (MACDriver ∗macp)

  Initialize the standard part of a `MACDriver` structure.
- void macStart (MACDriver ∗macp, const MACConfig ∗config)

  Configures and activates the MAC peripheral.
- void macStop (MACDriver ∗macp)

  Deactivates the MAC peripheral.
- msg_t macWaitTransmitDescriptor (MACDriver ∗macp, MACTransmitDescriptor ∗tdp, systime_t timeout)

  Allocates a transmission descriptor.
- void macReleaseTransmitDescriptor (MACTransmitDescriptor ∗tdp)

  Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.
- msg_t macWaitReceiveDescriptor (MACDriver ∗macp, MACReceiveDescriptor ∗rdp, systime_t timeout)

  Waits for a received frame.
- void macReleaseReceiveDescriptor (MACReceiveDescriptor ∗rdp)

  Releases a receive descriptor.
- bool macPollLinkStatus (MACDriver ∗macp)

  Updates and returns the link status.
- void mac_lld_init (void)

  Low level MAC initialization.
- void mac_lld_start (MACDriver ∗macp)

  Configures and activates the MAC peripheral.
- void mac_lld_stop (MACDriver ∗macp)

  Deactivates the MAC peripheral.
- msg_t mac_lld_get_transmit_descriptor (MACDriver ∗macp, MACTransmitDescriptor ∗tdp)

  Returns a transmission descriptor.
- void mac_lld_release_transmit_descriptor (MACTransmitDescriptor ∗tdp)

  Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.
- msg_t mac_lld_get_receive_descriptor (MACDriver ∗macp, MACReceiveDescriptor ∗rdp)

  Returns a receive descriptor.
- void mac_lld_release_receive_descriptor (MACReceiveDescriptor ∗rdp)

  Releases a receive descriptor.
- bool mac_lld_poll_link_status (MACDriver ∗macp)

  Updates and returns the link status.
- size_t mac_lld_write_transmit_descriptor (MACTransmitDescriptor ∗tdp, uint8_t ∗buf, size_t size)

  Writes to a transmit descriptor's stream.
- size_t mac_lld_read_receive_descriptor (MACReceiveDescriptor ∗rdp, uint8_t ∗buf, size_t size)

  Reads from a receive descriptor's stream.

- uint8_t ∗ mac_lld_get_next_transmit_buffer (MACTransmitDescriptor ∗tdp, size_t size, size_t ∗sizep)

  Returns a pointer to the next transmit buffer in the descriptor chain.
- const uint8_t ∗ mac_lld_get_next_receive_buffer (MACReceiveDescriptor ∗rdp, size_t ∗sizep)

  Returns a pointer to the next receive buffer in the descriptor chain.

Enumerations

Variables

- MACDriver ETHD1

  MAC1 driver identifier.

7.17.2 Macro Definition Documentation

7.17.2.1 #define MAC_USE_ZERO_COPY FALSE

Enables an event sources for incoming packets.

7.17.2.2 #define MAC_USE_EVENTS TRUE

Enables an event sources for incoming packets.

7.17.2.3 #define macGetReceiveEventSource(*macp*) (&(macp)->rdevent)

Returns the received frames event source.

Parameters

`in`	*macp*	pointer to the `MACDriver` object

Returns

The pointer to the `EventSource` structure.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.17.2.4 #define macWriteTransmitDescriptor(*tdp,* *buf,* *size*) mac_lld_write_transmit_descriptor(tdp, buf, size)

Writes to a transmit descriptor's stream.

Parameters

`in`	*tdp*	pointer to a `MACTransmitDescriptor` structure
`in`	*buf*	pointer to the buffer containing the data to be written
`in`	*size*	number of bytes to be written

Returns

The number of bytes written into the descriptor's stream, this value can be less than the amount specified in the parameter `size` if the maximum frame size is reached.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.17.2.5 #define macReadReceiveDescriptor(*rdp,* *buf,* *size*) **mac_lld_read_receive_descriptor**(rdp, buf, size)

Reads from a receive descriptor's stream.

Parameters

`in`	*rdp*	pointer to a `MACReceiveDescriptor` structure
`in`	*buf*	pointer to the buffer that will receive the read data
`in`	*size*	number of bytes to be read

Returns

The number of bytes read from the descriptor's stream, this value can be less than the amount specified in the parameter `size` if there are no more bytes to read.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.17.2.6 #define macGetNextTransmitBuffer(*tdp,* *size,* *sizep*) **mac_lld_get_next_transmit_buffer**(tdp, size, sizep)

Returns a pointer to the next transmit buffer in the descriptor chain.

Note

The API guarantees that enough buffers can be requested to fill a whole frame.

Parameters

`in`	*tdp*	pointer to a `MACTransmitDescriptor` structure
`in`	*size*	size of the requested buffer. Specify the frame size on the first call then scale the value down subtracting the amount of data already copied into the previous buffers.
`out`	*sizep*	pointer to variable receiving the real buffer size. The returned value can be less than the amount requested, this means that more buffers must be requested in order to fill the frame data entirely.

Returns

Pointer to the returned buffer.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.17.2.7 #define macGetNextReceiveBuffer(*rdp*, *sizep*) mac_lld_get_next_receive_buffer(rdp, sizep)

Returns a pointer to the next receive buffer in the descriptor chain.

Note

The API guarantees that the descriptor chain contains a whole frame.

Parameters

`in`	*rdp*	pointer to a `MACReceiveDescriptor` structure
`out`	*sizep*	pointer to variable receiving the buffer size, it is zero when the last buffer has already been returned.

Returns

Pointer to the returned buffer.

Return values

NULL	if the buffer chain has been entirely scanned.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.17.2.8 #define MAC_SUPPORTS_ZERO_COPY TRUE

This implementation supports the zero-copy mode API.

7.17.2.9 #define PLATFORM_MAC_USE_MAC1 FALSE

MAC driver enable switch.

If set to `TRUE` the support for MAC1 is included.

Note

The default is `FALSE`.

7.17.3 Typedef Documentation

7.17.3.1 typedef struct MACDriver MACDriver

Type of a structure representing a MAC driver.

7.17.4 Enumeration Type Documentation

7.17.4.1 enum macstate_t

Driver state machine possible states.

Enumerator

MAC_UNINIT	Not initialized.
MAC_STOP	Stopped.
MAC_ACTIVE	Active.

7.17.5 Function Documentation

7.17.5.1 void macInit (void)

MAC Driver initialization.

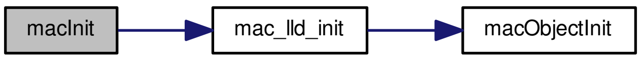
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



7.17.5.2 void macObjectInit (MACDriver ∗ *macp*)

Initialize the standard part of a `MACDriver` structure.

Parameters

out	*macp*	pointer to the `MACDriver` object

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.17.5.3 void macStart (MACDriver ∗ *macp*, const MACConfig ∗ *config*)

Configures and activates the MAC peripheral.

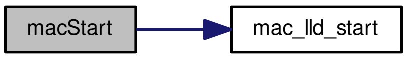
Parameters

in	*macp*	pointer to the `MACDriver` object
in	*config*	pointer to the `MACConfig` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.17.5.4 void macStop (MACDriver ∗ *macp*)

Deactivates the MAC peripheral.

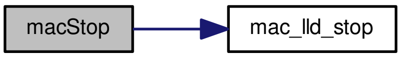
Parameters

in	*macp*	pointer to the MACDriver object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.17.5.5 msg_t macWaitTransmitDescriptor (MACDriver ∗ *macp,* MACTransmitDescriptor ∗ *tdp,* systime_t *timeout*)

Allocates a transmission descriptor.

One of the available transmission descriptors is locked and returned. If a descriptor is not currently available then the invoking thread is queued until one is freed.

Parameters

in	*macp*	pointer to the MACDriver object
out	*tdp*	pointer to a MACTransmitDescriptor structure
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The operation status.

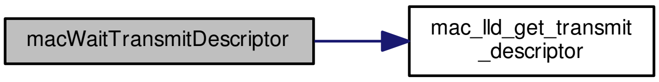
Return values

MSG_OK	the descriptor was obtained.
MSG_TIMEOUT	the operation timed out, descriptor not initialized.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.17.5.6 void macReleaseTransmitDescriptor (**MACTransmitDescriptor** ∗ *tdp*)

Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.

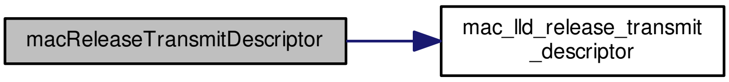
Parameters

`in`	*tdp*	the pointer to the `MACTransmitDescriptor` structure

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:


macReleaseTransmitDescriptor
mac_lld_release_transmit _descriptor

7.17.5.7 msg_t macWaitReceiveDescriptor (**MACDriver** ∗ *macp*, **MACReceiveDescriptor** ∗ *rdp*, systime_t *timeout*)

Waits for a received frame.

Stops until a frame is received and buffered. If a frame is not immediately available then the invoking thread is queued until one is received.

Parameters

in	*macp*	pointer to the `MACDriver` object
out	*rdp*	pointer to a `MACReceiveDescriptor` structure
in	*timeout*	the number of ticks before the operation timeouts, the following special values are allowed: • *TIME_IMMEDIATE* immediate timeout. • *TIME_INFINITE* no timeout.

Returns

The operation status.

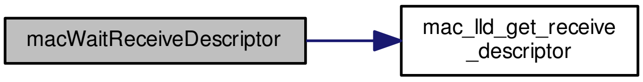
Return values

MSG_OK	the descriptor was obtained.
MSG_TIMEOUT	the operation timed out, descriptor not initialized.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.17.5.8 void macReleaseReceiveDescriptor (MACReceiveDescriptor ∗ *rdp*)

Releases a receive descriptor.

The descriptor and its buffer are made available for more incoming frames.

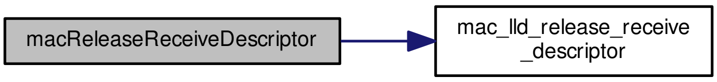
Parameters

in	*rdp*	the pointer to the `MACReceiveDescriptor` structure

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.17.5.9 bool macPollLinkStatus (MACDriver * *macp*)

Updates and returns the link status.

Parameters

in	*macp*	pointer to the MACDriver object

Returns

The link status.

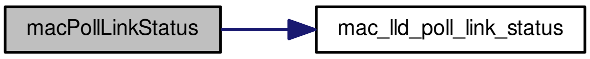
Return values

true	if the link is active.
false	if the link is down.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



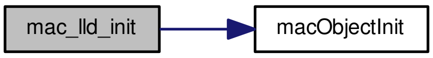
7.17.5.10 void mac_lld_init (void)

Low level MAC initialization.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.17.5.11 void mac_lld_start (MACDriver * *macp*)

Configures and activates the MAC peripheral.

Parameters

in	*macp*	pointer to the `MACDriver` object

Function Class:

Not an API, this function is for internal use only.

7.17.5.12 void mac_lld_stop (MACDriver * *macp*)

Deactivates the MAC peripheral.

Parameters

in	*macp*	pointer to the `MACDriver` object

Function Class:

Not an API, this function is for internal use only.

7.17.5.13 msg_t mac_lld_get_transmit_descriptor (MACDriver * *macp*, MACTransmitDescriptor * *tdp*)

Returns a transmission descriptor.

One of the available transmission descriptors is locked and returned.

Parameters

in	*macp*	pointer to the `MACDriver` object
out	*tdp*	pointer to a `MACTransmitDescriptor` structure

Returns

The operation status.

Return values

MSG_OK	the descriptor has been obtained.
MSG_TIMEOUT	descriptor not available.

Function Class:

Not an API, this function is for internal use only.

7.17.5.14 void mac_lld_release_transmit_descriptor (**MACTransmitDescriptor** ∗ *tdp*)

Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.

Parameters

`in`	*tdp*	the pointer to the `MACTransmitDescriptor` structure

Function Class:

Not an API, this function is for internal use only.

7.17.5.15 msg_t mac_lld_get_receive_descriptor (**MACDriver** ∗ *macp,* **MACReceiveDescriptor** ∗ *rdp*)

Returns a receive descriptor.

Parameters

`in`	*macp*	pointer to the `MACDriver` object
`out`	*rdp*	pointer to a `MACReceiveDescriptor` structure

Returns

The operation status.

Return values

MSG_OK	the descriptor has been obtained.
MSG_TIMEOUT	descriptor not available.

Function Class:

Not an API, this function is for internal use only.

7.17.5.16 void mac_lld_release_receive_descriptor (**MACReceiveDescriptor** ∗ *rdp*)

Releases a receive descriptor.

The descriptor and its buffer are made available for more incoming frames.

Parameters

in	*rdp*	the pointer to the `MACReceiveDescriptor` structure

Function Class:

Not an API, this function is for internal use only.

7.17.5.17 bool mac_lld_poll_link_status (MACDriver ∗ *macp*)

Updates and returns the link status.

Parameters

in	*macp*	pointer to the `MACDriver` object

Returns

The link status.

Return values

true	if the link is active.
false	if the link is down.

Function Class:

Not an API, this function is for internal use only.

7.17.5.18 size_t mac_lld_write_transmit_descriptor (MACTransmitDescriptor ∗ *tdp*, uint8_t ∗ *buf*, size_t *size*)

Writes to a transmit descriptor's stream.

Parameters

in	*tdp*	pointer to a `MACTransmitDescriptor` structure
in	*buf*	pointer to the buffer containing the data to be written
in	*size*	number of bytes to be written

Returns

The number of bytes written into the descriptor's stream, this value can be less than the amount specified in the parameter `size` if the maximum frame size is reached.

Function Class:

Not an API, this function is for internal use only.

7.17.5.19 size_t mac_lld_read_receive_descriptor (MACReceiveDescriptor ∗ *rdp*, uint8_t ∗ *buf*, size_t *size*)

Reads from a receive descriptor's stream.

Parameters

`in`	*rdp*	pointer to a `MACReceiveDescriptor` structure
`in`	*buf*	pointer to the buffer that will receive the read data
`in`	*size*	number of bytes to be read

Returns

The number of bytes read from the descriptor's stream, this value can be less than the amount specified in the parameter `size` if there are no more bytes to read.

Function Class:

Not an API, this function is for internal use only.

7.17.5.20 uint8_t * mac_lld_get_next_transmit_buffer (MACTransmitDescriptor * *tdp*, size_t *size*, size_t * *sizep*)

Returns a pointer to the next transmit buffer in the descriptor chain.

Note

The API guarantees that enough buffers can be requested to fill a whole frame.

Parameters

`in`	*tdp*	pointer to a `MACTransmitDescriptor` structure
`in`	*size*	size of the requested buffer. Specify the frame size on the first call then scale the value down subtracting the amount of data already copied into the previous buffers.
`out`	*sizep*	pointer to variable receiving the buffer size, it is zero when the last buffer has already been returned. Note that a returned size lower than the amount requested means that more buffers must be requested in order to fill the frame data entirely.

Returns

Pointer to the returned buffer.

Return values

NULL	if the buffer chain has been entirely scanned.

Function Class:

Not an API, this function is for internal use only.

7.17.5.21 const uint8_t * mac_lld_get_next_receive_buffer (MACReceiveDescriptor * *rdp*, size_t * *sizep*)

Returns a pointer to the next receive buffer in the descriptor chain.

Note

The API guarantees that the descriptor chain contains a whole frame.

Parameters

`in`	*rdp*	pointer to a `MACReceiveDescriptor` structure
`out`	*sizep*	pointer to variable receiving the buffer size, it is zero when the last buffer has already been returned.

Returns

Pointer to the returned buffer.

Return values

NULL	if the buffer chain has been entirely scanned.

Function Class:

Not an API, this function is for internal use only.

7.17.6 Variable Documentation

7.17.6.1 MACDriver ETHD1

MAC1 driver identifier.

7.18 HAL

Hardware Abstraction Layer.

7.18.1 Detailed Description

Hardware Abstraction Layer.

Under ChibiOS the set of the various device driver interfaces is called the HAL subsystem: Hardware Abstraction Layer. The HAL is the abstract interface between ChibiOS applications and hardware.

7.18.2 HAL Device Drivers Architecture

The HAL contains several kind of modules:

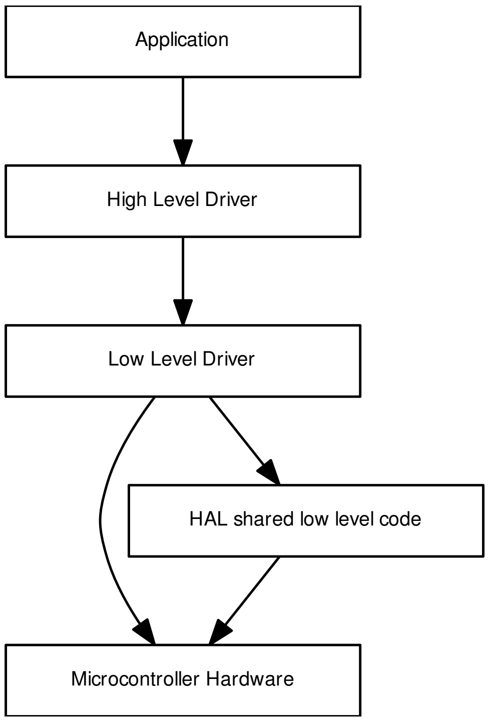
- Normal Device Drivers
- Complex Device Drivers
- Interfaces
- Inner Code

7.18.3 HAL Normal Device Drivers

Normal device are meant to interface the application to the underlying hardware through an high level API. Normal Device Drivers are split in two layers:

- High Level Device Driver (**HLD**). This layer contains the definitions of the driver's APIs and the platform independent part of the driver.
  An HLD is composed by two files:
  - `<driver>`.c, the HLD implementation file. This file must be included in the Makefile in order to use the driver.
  - `<driver>`.h, the HLD header file. This file is implicitly included by the HAL header file `hal.h`.
- Low Level Device Driver (**LLD**). This layer contains the platform dependent part of the driver.
  A LLD is composed by two files:
  - `<driver>_lld.c`, the LLD implementation file. This file must be included in the Makefile in order to use the driver.
  - `<driver>_lld.h`, the LLD header file. This file is implicitly included by the HLD header file.

7.18.3.1 Diagram



7.18.4 HAL Complex Device Drivers

It is a class of device drivers that offer an high level API but do not use the hardware directly. Complex device drivers use other drivers for accessing the machine resources.

7.18.5 HAL Interfaces

An interface is a binary structure allowing the access to a service using virtual functions. This allows to create drivers that can be accessed using a common interface. The concept of interface is commonly found in object-oriented languages like Java or C++, their meaning in ChibiOS/HAL is exactly the same.

7.18.6 HAL Inner Code

Some modules are shared among multiple device drivers and are not necessarily meant to be used by the application layer.

Modules

- Configuration

  HAL Configuration.
- Normal Drivers

HAL Normal Drivers.

- Complex Drivers

    HAL Complex Drivers.
- Interfaces

    HAL Interfaces.
- Inner Code

    HAL Inner Code.
- Support Code

    HAL Support Code.
- OSAL

    Operating System Abstraction Layer.

7.19 Configuration

HAL Configuration.

7.19.1 Detailed Description

HAL Configuration.

The file halconf.h contains the high level settings for all the drivers supported by the HAL. The low level, platform dependent, settings are contained in the mcuconf.h file instead and are describe in the various platforms reference manuals.

Drivers enable switches

- #define HAL_USE_PAL TRUE

  Enables the PAL subsystem.
- #define HAL_USE_ADC TRUE

  Enables the ADC subsystem.
- #define HAL_USE_CAN TRUE

  Enables the CAN subsystem.
- #define HAL_USE_DAC FALSE

  Enables the DAC subsystem.
- #define HAL_USE_EXT TRUE

  Enables the EXT subsystem.
- #define HAL_USE_GPT TRUE

  Enables the GPT subsystem.
- #define HAL_USE_I2C TRUE

  Enables the I2C subsystem.
- #define HAL_USE_I2S TRUE

  Enables the I2S subsystem.
- #define HAL_USE_ICU TRUE

  Enables the ICU subsystem.
- #define HAL_USE_MAC TRUE

  Enables the MAC subsystem.
- #define HAL_USE_MMC_SPI TRUE

  Enables the MMC_SPI subsystem.
- #define HAL_USE_PWM TRUE

  Enables the PWM subsystem.
- #define HAL_USE_QSPI TRUE

  Enables the QSPI subsystem.
- #define HAL_USE_RTC TRUE

  Enables the RTC subsystem.
- #define HAL_USE_SDC TRUE

  Enables the SDC subsystem.
- #define HAL_USE_SERIAL TRUE

  Enables the SERIAL subsystem.
- #define HAL_USE_SERIAL_USB TRUE

  Enables the SERIAL over USB subsystem.
- #define HAL_USE_SPI TRUE

  Enables the SPI subsystem.
- #define HAL_USE_UART TRUE

Enables the UART subsystem.

- #define HAL_USE_USB TRUE

  Enables the USB subsystem.
- #define HAL_USE_WDG TRUE

  Enables the WDG subsystem.

ADC driver related setting

- #define ADC_USE_WAIT TRUE

  Enables synchronous APIs.
- #define ADC_USE_MUTUAL_EXCLUSION TRUE

  Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.

CAN driver related setting

- #define CAN_USE_SLEEP_MODE TRUE

  Sleep mode related APIs inclusion switch.

I2C driver related setting

- #define I2C_USE_MUTUAL_EXCLUSION TRUE

  Enables the mutual exclusion APIs on the I2C bus.

MAC driver related setting

- #define MAC_USE_ZERO_COPY TRUE

  Enables an event sources for incoming packets.
- #define MAC_USE_EVENTS TRUE

  Enables an event sources for incoming packets.

MMC_SPI driver related setting

- #define MMC_NICE_WAITING TRUE

  Delays insertions.

SDC driver related setting

- #define SDC_INIT_RETRY 100

  Number of initialization attempts before rejecting the card.
- #define SDC_MMC_SUPPORT TRUE

  Include support for MMC cards.
- #define SDC_NICE_WAITING TRUE

  Delays insertions.

SERIAL driver related setting

- #define SERIAL_DEFAULT_BITRATE 38400

  Default bit rate.
- #define SERIAL_BUFFERS_SIZE 16

  Serial buffers size.

SERIAL_USB driver related setting

- #define SERIAL_USB_BUFFERS_SIZE 256

  Serial over USB buffers size.
- #define SERIAL_USB_BUFFERS_NUMBER 2

  Serial over USB number of buffers.

SPI driver related setting

- #define SPI_USE_WAIT TRUE

  Enables synchronous APIs.
- #define SPI_USE_MUTUAL_EXCLUSION TRUE

  Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.

UART driver related setting

- #define UART_USE_WAIT TRUE

  Enables synchronous APIs.
- #define UART_USE_MUTUAL_EXCLUSION TRUE

  Enables the `uartAcquireBus()` and `uartReleaseBus()` APIs.

USB driver related setting

- #define USB_USE_WAIT TRUE

  Enables synchronous APIs.

7.19.2 Macro Definition Documentation

7.19.2.1 #define HAL_USE_PAL TRUE

Enables the PAL subsystem.

7.19.2.2 #define HAL_USE_ADC TRUE

Enables the ADC subsystem.

7.19.2.3 #define HAL_USE_CAN TRUE

Enables the CAN subsystem.

7.19.2.4 #define HAL_USE_DAC FALSE

Enables the DAC subsystem.

7.19.2.5 #define HAL_USE_EXT TRUE

Enables the EXT subsystem.

7.19.2.6 #define HAL_USE_GPT TRUE

Enables the GPT subsystem.

7.19.2.7 #define HAL_USE_I2C TRUE

Enables the I2C subsystem.

7.19.2.8 #define HAL_USE_I2S TRUE

Enables the I2S subsystem.

7.19.2.9 #define HAL_USE_ICU TRUE

Enables the ICU subsystem.

7.19.2.10 #define HAL_USE_MAC TRUE

Enables the MAC subsystem.

7.19.2.11 #define HAL_USE_MMC_SPI TRUE

Enables the MMC_SPI subsystem.

7.19.2.12 #define HAL_USE_PWM TRUE

Enables the PWM subsystem.

7.19.2.13 #define HAL_USE_QSPI TRUE

Enables the QSPI subsystem.

7.19.2.14 #define HAL_USE_RTC TRUE

Enables the RTC subsystem.

7.19.2.15 #define HAL_USE_SDC TRUE

Enables the SDC subsystem.

7.19.2.16 #define HAL_USE_SERIAL TRUE

Enables the SERIAL subsystem.

7.19.2.17 #define HAL_USE_SERIAL_USB TRUE

Enables the SERIAL over USB subsystem.

7.19.2.18 #define HAL_USE_SPI TRUE

Enables the SPI subsystem.

7.19.2.19 #define HAL_USE_UART TRUE

Enables the UART subsystem.

7.19.2.20 #define HAL_USE_USB TRUE

Enables the USB subsystem.

7.19.2.21 #define HAL_USE_WDG TRUE

Enables the WDG subsystem.

7.19.2.22 #define ADC_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.19.2.23 #define ADC_USE_MUTUAL_EXCLUSION TRUE

Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.

Note

Disabling this option saves both code and data space.

7.19.2.24 #define CAN_USE_SLEEP_MODE TRUE

Sleep mode related APIs inclusion switch.

7.19.2.25 #define I2C_USE_MUTUAL_EXCLUSION TRUE

Enables the mutual exclusion APIs on the I2C bus.

7.19.2.26 #define MAC_USE_ZERO_COPY TRUE

Enables an event sources for incoming packets.

7.19.2.27 #define MAC_USE_EVENTS TRUE

Enables an event sources for incoming packets.

7.19.2.28 #define MMC_NICE_WAITING TRUE

Delays insertions.

If enabled this options inserts delays into the MMC waiting routines releasing some extra CPU time for the threads with lower priority, this may slow down the driver a bit however. This option is recommended also if the SPI driver does not use a DMA channel and heavily loads the CPU.

7.19.2.29 #define SDC_INIT_RETRY 100

Number of initialization attempts before rejecting the card.

Note

Attempts are performed at 10mS intervals.

7.19.2.30 #define SDC_MMC_SUPPORT TRUE

Include support for MMC cards.

Note

MMC support is not yet implemented so this option must be kept at `FALSE`.

7.19.2.31 #define SDC_NICE_WAITING TRUE

Delays insertions.

If enabled this options inserts delays into the MMC waiting routines releasing some extra CPU time for the threads with lower priority, this may slow down the driver a bit however.

7.19.2.32 #define SERIAL_DEFAULT_BITRATE 38400

Default bit rate.

Configuration parameter, this is the baud rate selected for the default configuration.

7.19.2.33 #define SERIAL_BUFFERS_SIZE 16

Serial buffers size.

Configuration parameter, you can change the depth of the queue buffers depending on the requirements of your application.

Note

The default is 16 bytes for both the transmission and receive buffers.

7.19.2.34 #define SERIAL_USB_BUFFERS_SIZE 256

Serial over USB buffers size.

Configuration parameter, the buffer size must be a multiple of the USB data endpoint maximum packet size.

Note

The default is 256 bytes for both the transmission and receive buffers.

7.19.2.35 #define SERIAL_USB_BUFFERS_NUMBER 2

Serial over USB number of buffers.

Note

The default is 2 buffers.

7.19.2.36 #define SPI_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.19.2.37 #define SPI_USE_MUTUAL_EXCLUSION TRUE

Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.

Note

Disabling this option saves both code and data space.

7.19.2.38 #define UART_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.19.2.39 #define UART_USE_MUTUAL_EXCLUSION TRUE

Enables the `uartAcquireBus()` and `uartReleaseBus()` APIs.

Note

Disabling this option saves both code and data space.

7.19.2.40 #define USB_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.20 Normal Drivers

HAL Normal Drivers.

7.20.1 Detailed Description

HAL Normal Drivers.

Modules

- ADC Driver

  Generic ADC Driver.
- CAN Driver

  Generic CAN Driver.
- DAC Driver

  Generic DAC Driver.
- EXT Driver

  Generic EXT Driver.
- GPT Driver

  Generic GPT Driver.
- HAL Driver

  Hardware Abstraction Layer.
- I2C Driver

  Generic I2C Driver.
- I2S Driver

  Generic I2S Driver.
- ICU Driver

  Generic ICU Driver.
- MAC Driver

  Generic MAC Driver.
- PAL Driver

  I/O Ports Abstraction Layer.
- PWM Driver

  Generic PWM Driver.
- QSPI Driver

  Generic QSPI Driver.
- RTC Driver

  Generic RTC Driver.
- SDC Driver

  Generic SD Card Driver.
- Serial Driver

  Generic Serial Driver.
- SPI Driver

  Generic SPI Driver.
- ST Driver

  Generic System Tick Driver.
- UART Driver

  Generic UART Driver.
- USB Driver

  Generic USB Driver.
- WDG Driver

  Generic WDG Driver.

7.21 Complex Drivers

HAL Complex Drivers.

7.21.1 Detailed Description

HAL Complex Drivers.

Modules

- MMC over SPI Driver

  Generic MMC driver.
- Serial over USB Driver

  Serial over USB Driver.

7.22 Interfaces

HAL Interfaces.

7.22.1 Detailed Description

HAL Interfaces.

Modules

- Abstract NOR Flash Class

  Generic NOR Flash interface.
- Abstract I/O Channel
- Abstract Files
- Abstract I/O Block Device
- Abstract Streams

7.23 Inner Code

HAL Inner Code.

7.23.1 Detailed Description

HAL Inner Code.

Modules

- I/O Buffers Queues
- I/O Bytes Queues
- MMC/SD Block Device

7.24 Support Code

HAL Support Code.

7.24.1 Detailed Description

HAL Support Code.

Modules

- MII/RMII Header

  MII/RMII Support Header.
- USB CDC Header

  USB CDC Support Header.

7.25 OSAL

Operating System Abstraction Layer.

Operating System Abstraction Layer.

The OSAL

The OSAL is the link between ChibiOS/HAL and services provided by operating systems like:

- Critical Zones handling.
- Interrupts handling.
- Runtime Errors management.
- Inter-task synchronization.
- Task-ISR synchronization.
- Time management.
- Events.

ChibiOS/HAL is designed to tightly integrate with the underlying RTOS in order to provide the best experience to developers and minimize integration issues.
This section describes the API that OSALs are expected to expose to the HAL.

RTOS Requirements

The OSAL API closely resembles the ChibiOS/RT API, for obvious reasons, however an OSAL module can be implemented for any reasonably complete RTOS or even a RTOS-less bare metal machine, if required.
In order to be able to support an HAL an RTOS should support the following minimal set of features:

- Task-level critical zones API.
- ISR-level critical zones API, only required on those CPU architectures supporting preemptable ISRs like Cortex-Mx cores.
- Ability to invoke API functions from inside a task critical zone. Functions that are required to support this feature are marked with an "I" or "S" letter at the end of the name.
- Ability to invoke API functions from inside an ISR critical zone. Functions that are required to support this feature are marked with an "I" letter at the end of the name.
- Tasks Queues or Counting Semaphores with Timeout capability.
- Ability to suspend a task and wakeup it from ISR with Timeout capability.
- Event flags, the mechanism can be simulated using callbacks in case the RTOS does not support it.
- Mutual Exclusion mechanism like Semaphores or Mutexes.

All the above requirements can be satisfied even on naked HW with a very think SW layer. In case that the HAL is required to work without an RTOS.

Supported RTOSes

The RTOSes supported out of the box are:

- ChibiOS/RT
- ChibiOS/NIL

Implementations have also been successfully created on RTOSes not belonging to the ChibiOS products family but are not supported as a core feature of ChibiOS/HAL.

7.26 MII/RMII Header

MII/RMII Support Header.

7.26.1 Detailed Description

MII/RMII Support Header.

This header contains definitions and types related to MII/RMII.

Generic MII registers

- #define MII_BMCR 0x00
- #define MII_BMSR 0x01
- #define MII_PHYSID1 0x02
- #define MII_PHYSID2 0x03
- #define MII_ADVERTISE 0x04
- #define MII_LPA 0x05
- #define MII_EXPANSION 0x06
- #define MII_ANNPTR 0x07
- #define MII_CTRL1000 0x09
- #define MII_STAT1000 0x0a
- #define MII_ESTATUS 0x0f
- #define MII_PHYSTS 0x10
- #define MII_MICR 0x11
- #define MII_DCOUNTER 0x12
- #define MII_FCSCOUNTER 0x13
- #define MII_NWAYTEST 0x14
- #define MII_RERRCOUNTER 0x15
- #define MII_SREVISION 0x16
- #define MII_RESV1 0x17
- #define MII_LBRERROR 0x18
- #define MII_PHYADDR 0x19
- #define MII_RESV2 0x1a
- #define MII_TPISTATUS 0x1b
- #define MII_NCONFIG 0x1c

Basic mode control register

- #define BMCR_RESV 0x007f
- #define BMCR_CTST 0x0080
- #define BMCR_FULLDPLX 0x0100
- #define BMCR_ANRESTART 0x0200
- #define BMCR_ISOLATE 0x0400
- #define BMCR_PDOWN 0x0800
- #define BMCR_ANENABLE 0x1000
- #define BMCR_SPEED100 0x2000
- #define BMCR_LOOPBACK 0x4000
- #define BMCR_RESET 0x8000

Basic mode status register

- #define BMSR_ERCAP 0x0001
- #define BMSR_JCD 0x0002
- #define BMSR_LSTATUS 0x0004
- #define BMSR_ANEGCAPABLE 0x0008
- #define BMSR_RFAULT 0x0010
- #define BMSR_ANEGCOMPLETE 0x0020
- #define BMSR_MFPRESUPPCAP 0x0040
- #define BMSR_RESV 0x0780
- #define BMSR_10HALF 0x0800
- #define BMSR_10FULL 0x1000
- #define BMSR_100HALF 0x2000
- #define BMSR_100FULL 0x4000
- #define BMSR_100BASE4 0x8000

Advertisement control register

- #define ADVERTISE_SLCT 0x001f
- #define ADVERTISE_CSMA 0x0001
- #define ADVERTISE_10HALF 0x0020
- #define ADVERTISE_10FULL 0x0040
- #define ADVERTISE_100HALF 0x0080
- #define ADVERTISE_100FULL 0x0100
- #define ADVERTISE_100BASE4 0x0200
- #define ADVERTISE_PAUSE_CAP 0x0400
- #define ADVERTISE_PAUSE_ASYM 0x0800
- #define ADVERTISE_RESV 0x1000
- #define ADVERTISE_RFAULT 0x2000
- #define ADVERTISE_LPACK 0x4000
- #define ADVERTISE_NPAGE 0x8000
- #define **ADVERTISE_FULL**
- #define **ADVERTISE_ALL**

Link partner ability register

- #define LPA_SLCT 0x001f
- #define LPA_10HALF 0x0020
- #define LPA_10FULL 0x0040
- #define LPA_100HALF 0x0080
- #define LPA_100FULL 0x0100
- #define LPA_100BASE4 0x0200
- #define LPA_PAUSE_CAP 0x0400
- #define LPA_PAUSE_ASYM 0x0800
- #define LPA_RESV 0x1000
- #define LPA_RFAULT 0x2000
- #define LPA_LPACK 0x4000
- #define LPA_NPAGE 0x8000
- #define **LPA_DUPLEX** (LPA_10FULL | LPA_100FULL)
- #define **LPA_100** (LPA_100FULL | LPA_100HALF | LPA_100BASE4)

Expansion register for auto-negotiation

- #define EXPANSION_NWAY 0x0001
- #define EXPANSION_LCWP 0x0002
- #define EXPANSION_ENABLENPAGE 0x0004
- #define EXPANSION_NPCAPABLE 0x0008
- #define EXPANSION_MFAULTS 0x0010
- #define EXPANSION_RESV 0xffe0

N-way test register

- #define NWAYTEST_RESV1 0x00ff
- #define NWAYTEST_LOOPBACK 0x0100
- #define NWAYTEST_RESV2 0xfe00

PHY identifiers

- #define **MII_DM9161_ID** 0x0181b8a0
- #define **MII_AM79C875_ID** 0x00225540
- #define **MII_KS8721_ID** 0x00221610
- #define **MII_STE101P_ID** 0x00061C50
- #define **MII_DP83848I_ID** 0x20005C90
- #define **MII_LAN8710A_ID** 0x0007C0F1
- #define **MII_LAN8720_ID** 0x0007C0F0
- #define **MII_LAN8742A_ID** 0x0007C130

7.26.2 Macro Definition Documentation

7.26.2.1 #define MII_BMCR 0x00

Basic mode control register.

7.26.2.2 #define MII_BMSR 0x01

Basic mode status register.

7.26.2.3 #define MII_PHYSID1 0x02

PHYS ID 1.

7.26.2.4 #define MII_PHYSID2 0x03

PHYS ID 2.

7.26.2.5 #define MII_ADVERTISE 0x04

Advertisement control reg.

7.26.2.6 #define MII_LPA 0x05

Link partner ability reg.

7.26.2.7 #define MII_EXPANSION 0x06

Expansion register.

7.26.2.8 #define MII_ANNPTR 0x07

1000BASE-T control.

7.26.2.9 #define MII_CTRL1000 0x09

1000BASE-T control.

7.26.2.10 #define MII_STAT1000 0x0a

1000BASE-T status.

7.26.2.11 #define MII_ESTATUS 0x0f

Extended Status.

7.26.2.12 #define MII_PHYSTS 0x10

PHY Status register.

7.26.2.13 #define MII_MICR 0x11

MII Interrupt ctrl register.

7.26.2.14 #define MII_DCOUNTER 0x12

Disconnect counter.

7.26.2.15 #define MII_FCSCOUNTER 0x13

False carrier counter.

7.26.2.16 #define MII_NWAYTEST 0x14

N-way auto-neg test reg.

7.26.2.17 #define MII_RERRCOUNTER 0x15

Receive error counter.

7.26.2.18 #define MII_SREVISION 0x16

Silicon revision.

7.26.2.19 #define MII_RESV1 0x17

Reserved.

7.26.2.20 #define MII_LBRERROR 0x18

Lpback, rx, bypass error.

7.26.2.21 #define MII_PHYADDR 0x19

PHY address.

7.26.2.22 #define MII_RESV2 0x1a

Reserved.

7.26.2.23 #define MII_TPISTATUS 0x1b

TPI status for 10Mbps.

7.26.2.24 #define MII_NCONFIG 0x1c

Network interface config.

7.26.2.25 #define BMCR_RESV 0x007f

Unused.

7.26.2.26 #define BMCR_CTST 0x0080

Collision test.

7.26.2.27 #define BMCR_FULLDPLX 0x0100

Full duplex.

7.26.2.28 #define BMCR_ANRESTART 0x0200

Auto negotiation restart.

7.26.2.29 #define BMCR_ISOLATE 0x0400

Disconnect DP83840 from MII.

7.26.2.30 #define BMCR_PDOWN 0x0800

Powerdown.

7.26.2.31 #define BMCR_ANENABLE 0x1000

Enable auto negotiation.

7.26.2.32 #define BMCR_SPEED100 0x2000

Select 100Mbps.

7.26.2.33 #define BMCR_LOOPBACK 0x4000

TXD loopback bit.

7.26.2.34 #define BMCR_RESET 0x8000

Reset.

7.26.2.35 #define BMSR_ERCAP 0x0001

Ext-reg capability.

7.26.2.36 #define BMSR_JCD 0x0002

Jabber detected.

7.26.2.37 #define BMSR_LSTATUS 0x0004

Link status.

7.26.2.38 #define BMSR_ANEGCAPABLE 0x0008

Able to do auto-negotiation.

7.26.2.39 #define BMSR_RFAULT 0x0010

Remote fault detected.

7.26.2.40 #define BMSR_ANEGCOMPLETE 0x0020

Auto-negotiation complete.

7.26.2.41 #define BMSR_MFPRESUPPCAP 0x0040

Able to suppress preamble.

7.26.2.42 #define BMSR_RESV 0x0780

Unused.

7.26.2.43 #define BMSR_10HALF 0x0800

Can do 10mbps, half-duplex.

7.26.2.44 #define BMSR_10FULL 0x1000

Can do 10mbps, full-duplex.

7.26.2.45 #define BMSR_100HALF 0x2000

Can do 100mbps, half-duplex.

7.26.2.46 #define BMSR_100FULL 0x4000

Can do 100mbps, full-duplex.

7.26.2.47 #define BMSR_100BASE4 0x8000

Can do 100mbps, 4k packets.

7.26.2.48 #define ADVERTISE_SLCT 0x001f

Selector bits.

7.26.2.49 #define ADVERTISE_CSMA 0x0001

Only selector supported.

7.26.2.50 #define ADVERTISE_10HALF 0x0020

Try for 10mbps half-duplex.

7.26.2.51 #define ADVERTISE_10FULL 0x0040

Try for 10mbps full-duplex.

7.26.2.52 #define ADVERTISE_100HALF 0x0080

Try for 100mbps half-duplex.

7.26.2.53 #define ADVERTISE_100FULL 0x0100

Try for 100mbps full-duplex.

7.26.2.54 #define ADVERTISE_100BASE4 0x0200

Try for 100mbps 4k packets.

7.26.2.55 #define ADVERTISE_PAUSE_CAP 0x0400

Try for pause.

7.26.2.56 #define ADVERTISE_PAUSE_ASYM 0x0800

Try for asymetric pause.

7.26.2.57 #define ADVERTISE_RESV 0x1000

Unused.

7.26.2.58 #define ADVERTISE_RFAULT 0x2000

Say we can detect faults.

7.26.2.59 #define ADVERTISE_LPACK 0x4000

Ack link partners response.

7.26.2.60 #define ADVERTISE_NPAGE 0x8000

Next page bit.

7.26.2.61 #define LPA_SLCT 0x001f

Same as advertise selector.

7.26.2.62 #define LPA_10HALF 0x0020

Can do 10mbps half-duplex.

7.26.2.63 #define LPA_10FULL 0x0040

Can do 10mbps full-duplex.

7.26.2.64 #define LPA_100HALF 0x0080

Can do 100mbps half-duplex.

7.26.2.65 #define LPA_100FULL 0x0100

Can do 100mbps full-duplex.

7.26.2.66 #define LPA_100BASE4 0x0200

Can do 100mbps 4k packets.

7.26.2.67 #define LPA_PAUSE_CAP 0x0400

Can pause.

7.26.2.68 #define LPA_PAUSE_ASYM 0x0800

Can pause asymetrically.

7.26.2.69 #define LPA_RESV 0x1000

Unused.

7.26.2.70 #define LPA_RFAULT 0x2000

Link partner faulted.

7.26.2.71 #define LPA_LPACK 0x4000

Link partner acked us.

7.26.2.72 #define LPA_NPAGE 0x8000

Next page bit.

7.26.2.73 #define EXPANSION_NWAY 0x0001

Can do N-way auto-nego.

7.26.2.74 #define EXPANSION_LCWP 0x0002

Got new RX page code word.

7.26.2.75 #define EXPANSION_ENABLENPAGE 0x0004

This enables npage words.

7.26.2.76 #define EXPANSION_NPCAPABLE 0x0008

Link partner supports npage.

7.26.2.77 #define EXPANSION_MFAULTS 0x0010

Multiple faults detected.

7.26.2.78 #define EXPANSION_RESV 0xffe0

Unused.

7.26.2.79 #define NWAYTEST_RESV1 0x00ff

Unused.

7.26.2.80 #define NWAYTEST_LOOPBACK 0x0100

Enable loopback for N-way.

7.26.2.81 #define NWAYTEST_RESV2 0xfe00

Unused.

7.27 MMC over SPI Driver

Generic MMC driver.

7.27.1 Detailed Description

Generic MMC driver.

This module implements a portable MMC/SD driver that uses a SPI driver as physical layer. Hot plugging and removal are supported through kernel events.

Precondition

In order to use the MMC_SPI driver the `HAL_USE_MMC_SPI` and `HAL_USE_SPI` options must be enabled in `halconf.h`.

7.27.2 Driver State Machine

This driver implements a state machine internally, see the Abstract I/O Block Device module documentation for details.

7.27.3 Driver Operations

This driver allows to read or write single or multiple 512 bytes blocks on a SD Card.

Macros

- #define _mmc_driver_methods _mmcsd_block_device_methods

  `MMCDriver` specific methods.

MMC_SPI configuration options

- #define MMC_NICE_WAITING TRUE

  Delays insertions.

Macro Functions

- #define mmcIsCardInserted(mmcp) mmc_lld_is_card_inserted(mmcp)

  Returns the card insertion status.
- #define mmcIsWriteProtected(mmcp) mmc_lld_is_write_protected(mmcp)

  Returns the write protect status.

Data Structures

- struct MMCConfig

  MMC/SD over SPI driver configuration structure.
- struct MMCDriverVMT

  `MMCDriver` virtual methods table.
- struct MMCDriver

  Structure representing a MMC/SD over SPI driver.

Functions

- static uint8_t crc7 (uint8_t crc, const uint8_t ∗buffer, size_t len)

  Calculate the MMC standard CRC-7 based on a lookup table.
- static void wait (MMCDriver ∗mmcp)

  Waits an idle condition.
- static void send_hdr (MMCDriver ∗mmcp, uint8_t cmd, uint32_t arg)

  Sends a command header.
- static uint8_t recvr1 (MMCDriver ∗mmcp)

  Receives a single byte response.
- static uint8_t recvr3 (MMCDriver ∗mmcp, uint8_t ∗buffer)

  Receives a three byte response.
- static uint8_t send_command_R1 (MMCDriver ∗mmcp, uint8_t cmd, uint32_t arg)

  Sends a command an returns a single byte response.
- static uint8_t send_command_R3 (MMCDriver ∗mmcp, uint8_t cmd, uint32_t arg, uint8_t ∗response)

  Sends a command which returns a five bytes response (R3).
- static bool read_CxD (MMCDriver ∗mmcp, uint8_t cmd, uint32_t cxd[4])

  Reads the CSD.
- static void sync (MMCDriver ∗mmcp)

  Waits that the card reaches an idle state.
- void mmcInit (void)

  MMC over SPI driver initialization.
- void mmcObjectInit (MMCDriver ∗mmcp)

  Initializes an instance.
- void mmcStart (MMCDriver ∗mmcp, const MMCConfig ∗config)

  Configures and activates the MMC peripheral.
- void mmcStop (MMCDriver ∗mmcp)

  Disables the MMC peripheral.
- bool mmcConnect (MMCDriver ∗mmcp)

  Performs the initialization procedure on the inserted card.
- bool mmcDisconnect (MMCDriver ∗mmcp)

  Brings the driver in a state safe for card removal.
- bool mmcStartSequentialRead (MMCDriver ∗mmcp, uint32_t startblk)

  Starts a sequential read.
- bool mmcSequentialRead (MMCDriver ∗mmcp, uint8_t ∗buffer)

  Reads a block within a sequential read operation.
- bool mmcStopSequentialRead (MMCDriver ∗mmcp)

  Stops a sequential read gracefully.
- bool mmcStartSequentialWrite (MMCDriver ∗mmcp, uint32_t startblk)

  Starts a sequential write.
- bool mmcSequentialWrite (MMCDriver ∗mmcp, const uint8_t ∗buffer)

  Writes a block within a sequential write operation.
- bool mmcStopSequentialWrite (MMCDriver ∗mmcp)

  Stops a sequential write gracefully.
- bool mmcSync (MMCDriver ∗mmcp)

  Waits for card idle condition.
- bool mmcGetInfo (MMCDriver ∗mmcp, BlockDeviceInfo ∗bdip)

  Returns the media info.
- bool mmcErase (MMCDriver ∗mmcp, uint32_t startblk, uint32_t endblk)

  Erases blocks.

Variables

- static const struct MMCDriverVMT mmc_vmt

  Virtual methods table.
- static const uint8_t crc7_lookup_table [256]

  Lookup table for CRC-7 (based on polynomial x^ 7 + x^ 3 + 1).

7.27.4 Macro Definition Documentation

7.27.4.1 #define MMC_NICE_WAITING TRUE

Delays insertions.

If enabled this options inserts delays into the MMC waiting routines releasing some extra CPU time for the threads with lower priority, this may slow down the driver a bit however. This option is recommended also if the SPI driver does not use a DMA channel and heavily loads the CPU.

7.27.4.2 #define _mmc_driver_methods _mmcsd_block_device_methods

`MMCDriver` specific methods.

7.27.4.3 #define mmcIsCardInserted(*mmcp*) mmc_lld_is_card_inserted(mmcp)

Returns the card insertion status.

Note

This macro wraps a low level function named `sdc_lld_is_card_inserted()`, this function must be provided by the application because it is not part of the SDC driver.

Parameters

in	*mmcp*	pointer to the `MMCDriver` object

Returns

The card state.

Return values

FALSE	card not inserted.
TRUE	card inserted.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.27.4.4 #define mmcIsWriteProtected(*mmcp*) mmc_lld_is_write_protected(mmcp)

Returns the write protect status.

Parameters

`in`	*mmcp*	pointer to the `MMCDriver` object

Returns

The card state.

Return values

FALSE	card not inserted.
TRUE	card inserted.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.27.5 Function Documentation

7.27.5.1 static uint8_t crc7 (uint8_t *crc,* const uint8_t ∗ *buffer,* size_t *len*) `[static]`

Calculate the MMC standard CRC-7 based on a lookup table.

Parameters

`in`	*crc*	start value for CRC
`in`	*buffer*	pointer to data buffer
`in`	*len*	length of data

Returns

Calculated CRC

7.27.5.2 static void wait (MMCDriver ∗ *mmcp*) `[static]`

Waits an idle condition.

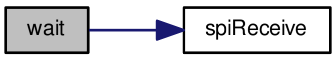
Parameters

`in`	*mmcp*	pointer to the `MMCDriver` object

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.27.5.3 static void send_hdr (**MMCDriver** ∗ *mmcp*, uint8_t *cmd*, uint32_t *arg*) [static]

Sends a command header.

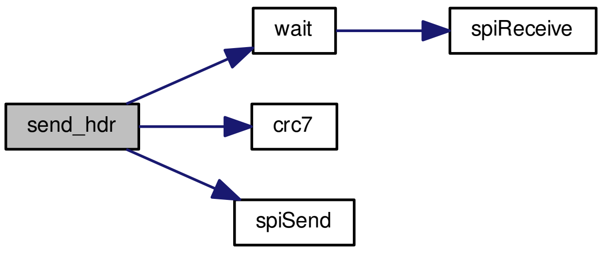
Parameters

in	*mmcp*	pointer to the MMCDriver object
in	*cmd*	the command id
in	*arg*	the command argument

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.27.5.4 static uint8_t recvr1 (**MMCDriver** ∗ *mmcp*) [static]

Receives a single byte response.

Parameters

in	*mmcp*	pointer to the MMCDriver object

Returns

The response as an `uint8_t` value.

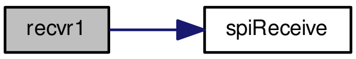
Return values

0xFF	timed out.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.27.5.5 static uint8_t recvr3 (**MMCDriver** ∗ *mmcp*, uint8_t ∗ *buffer*) `[static]`

Receives a three byte response.

Parameters

`in`	*mmcp*	pointer to the `MMCDriver` object
`out`	*buffer*	pointer to four bytes wide buffer

Returns

First response byte as an `uint8_t` value.

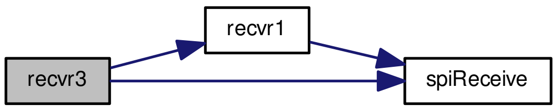
Return values

0xFF	timed out.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.27.5.6 static uint8_t send_command_R1 (**MMCDriver** * *mmcp*, uint8_t *cmd*, uint32_t *arg*) [static]

Sends a command an returns a single byte response.

Parameters

in	*mmcp*	pointer to the MMCDriver object
in	*cmd*	the command id
in	*arg*	the command argument

Returns

The response as an uint8_t value.

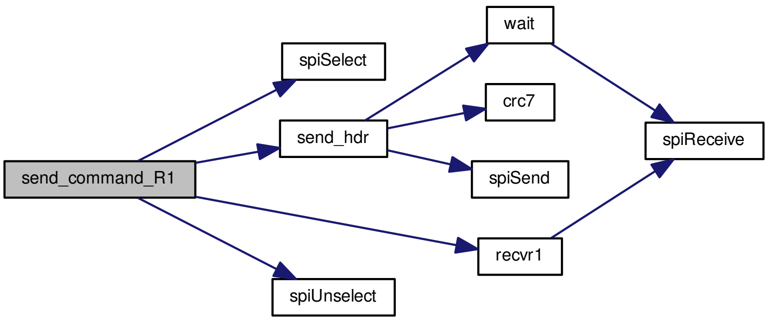
Return values

0xFF	timed out.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.27.5.7 static uint8_t send_command_R3 (MMCDriver ∗ *mmcp*, uint8_t *cmd*, uint32_t *arg*, uint8_t ∗ *response*) [static]

Sends a command which returns a five bytes response (R3).

Parameters

in	*mmcp*	pointer to the MMCDriver object
in	*cmd*	the command id
in	*arg*	the command argument
out	*response*	pointer to four bytes wide uint8_t buffer

Returns

The first byte of the response (R1) as an `uint8_t` value.

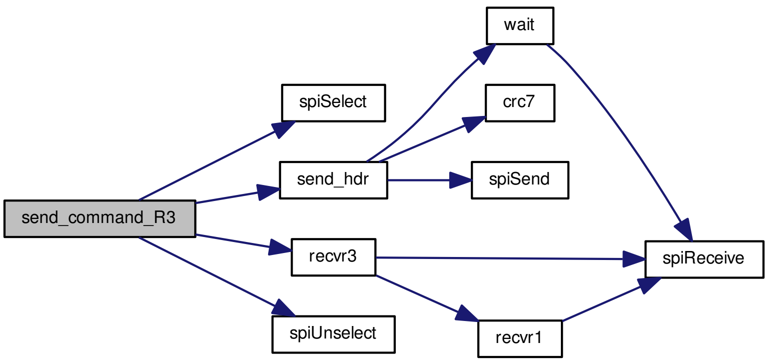
Return values

0xFF	timed out.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.27.5.8 static bool read_CxD (MMCDriver ∗ *mmcp*, uint8_t *cmd*, uint32_t *cxd[4]*) [static]

Reads the CSD.

Parameters

in	*mmcp*	pointer to the MMCDriver object
out	*cmd*	command
out	*cxd*	pointer to the CSD/CID buffer

Returns

The operation status.

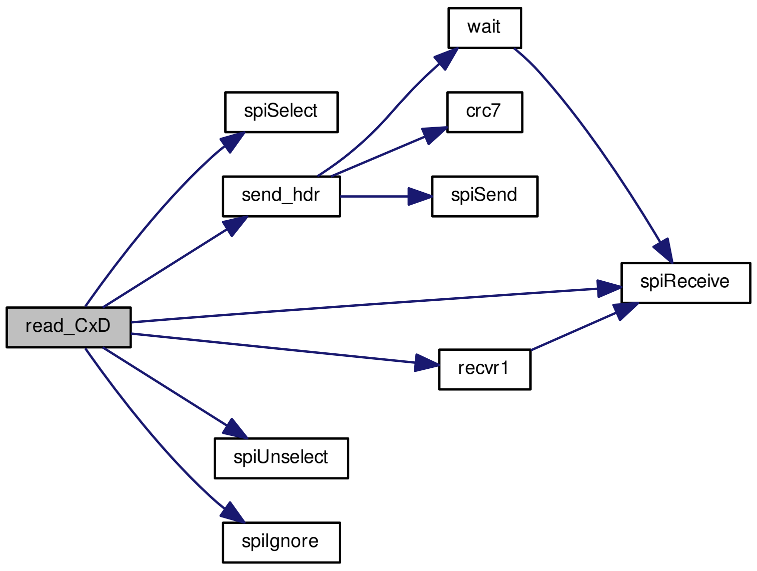
Return values

HAL_SUCCESS	the operation succeeded.
HAL_FAILED	the operation failed.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.27.5.9 static void sync (**MMCDriver** ∗ *mmcp*) `[static]`

Waits that the card reaches an idle state.

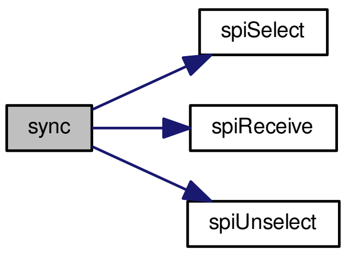
Parameters

`in`	*mmcp*	pointer to the `MMCDriver` object

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.27.5.10 void mmcInit (void)

MMC over SPI driver initialization.

Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.27.5.11 void mmcObjectInit (**MMCDriver** ∗ *mmcp*)

Initializes an instance.

Parameters

`out`	*mmcp*	pointer to the `MMCDriver` object

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.27.5.12 void mmcStart (**MMCDriver** ∗ *mmcp*, const **MMCConfig** ∗ *config*)

Configures and activates the MMC peripheral.

Parameters

`in`	*mmcp*	pointer to the `MMCDriver` object
`in`	*config*	pointer to the `MMCConfig` object.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.27.5.13 void mmcStop (MMCDriver * *mmcp*)

Disables the MMC peripheral.

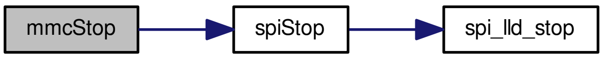
Parameters

`in`	*mmcp*	pointer to the `MMCDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.27.5.14 bool mmcConnect (MMCDriver * *mmcp*)

Performs the initialization procedure on the inserted card.

This function should be invoked when a card is inserted and brings the driver in the `MMC_READY` state where it is possible to perform read and write operations.

Note

It is possible to invoke this function from the insertion event handler.

Parameters

`in`	*mmcp*	pointer to the `MMCDriver` object

Returns

The operation status.

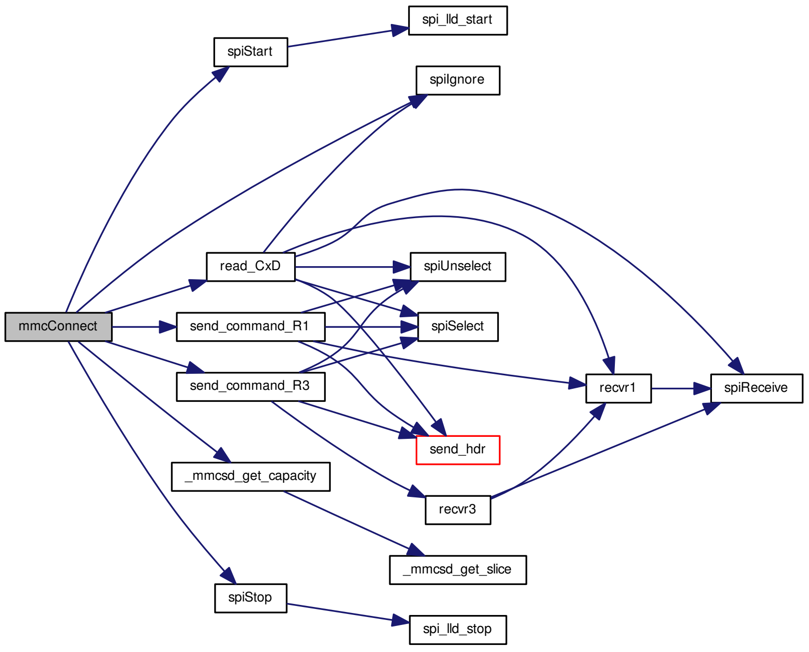
Return values

HAL_SUCCESS	the operation succeeded and the driver is now in the `MMC_READY` state.
HAL_FAILED	the operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.27.5.15 bool mmcDisconnect (MMCDriver * *mmcp*)

Brings the driver in a state safe for card removal.

Parameters

`in`	*mmcp*	pointer to the `MMCDriver` object

Returns

The operation status.

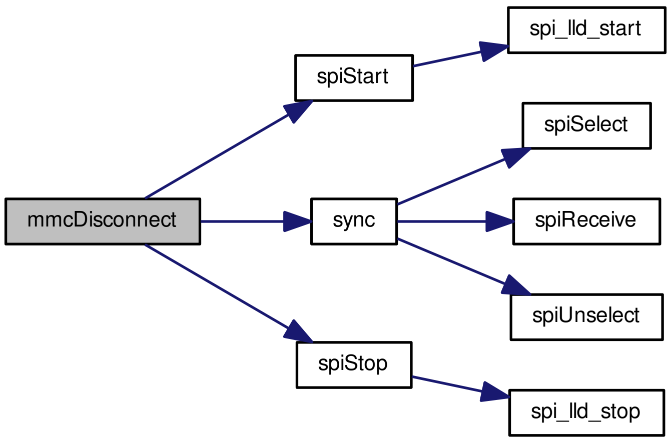
Return values

HAL_SUCCESS	the operation succeeded and the driver is now in the `MMC_INSERTED` state.
HAL_FAILED	the operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.27.5.16 bool mmcStartSequentialRead (MMCDriver ∗ *mmcp*, uint32_t *startblk*)

Starts a sequential read.

Parameters

`in`	*mmcp*	pointer to the `MMCDriver` object
`in`	*startblk*	first block to read

Returns

The operation status.

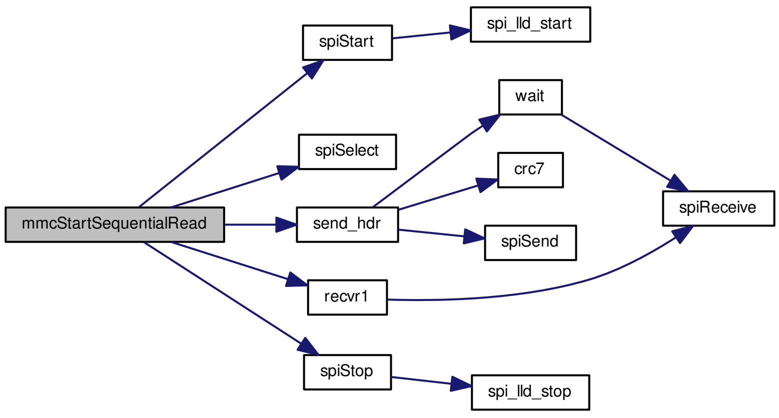
Return values

HAL_SUCCESS	the operation succeeded.
HAL_FAILED	the operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.27.5.17 bool mmcSequentialRead (MMCDriver ∗ *mmcp*, uint8_t ∗ *buffer*)

Reads a block within a sequential read operation.

Parameters

`in`	*mmcp*	pointer to the `MMCDriver` object
`out`	*buffer*	pointer to the read buffer

Returns

The operation status.

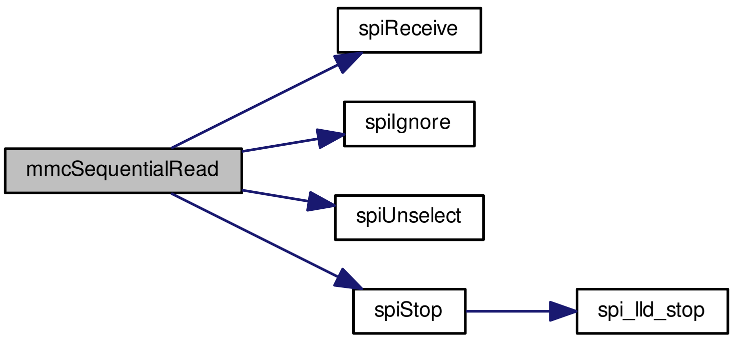
Return values

HAL_SUCCESS	the operation succeeded.
HAL_FAILED	the operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.27.5.18 bool mmcStopSequentialRead (**MMCDriver** ∗ *mmcp*)

Stops a sequential read gracefully.

Parameters

`in`	*mmcp*	pointer to the `MMCDriver` object

Returns

The operation status.

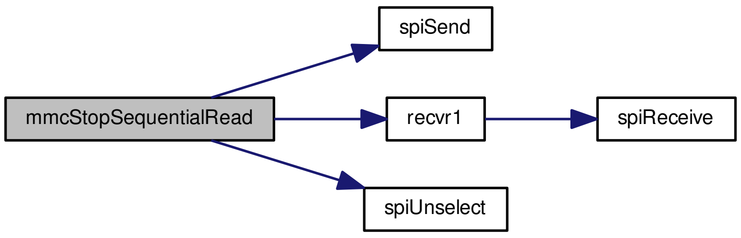
Return values

HAL_SUCCESS	the operation succeeded.
HAL_FAILED	the operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.27.5.19 bool mmcStartSequentialWrite (**MMCDriver** ∗ *mmcp,* uint32_t *startblk*)

Starts a sequential write.

Parameters

`in`	*mmcp*	pointer to the `MMCDriver` object
`in`	*startblk*	first block to write

Returns

The operation status.

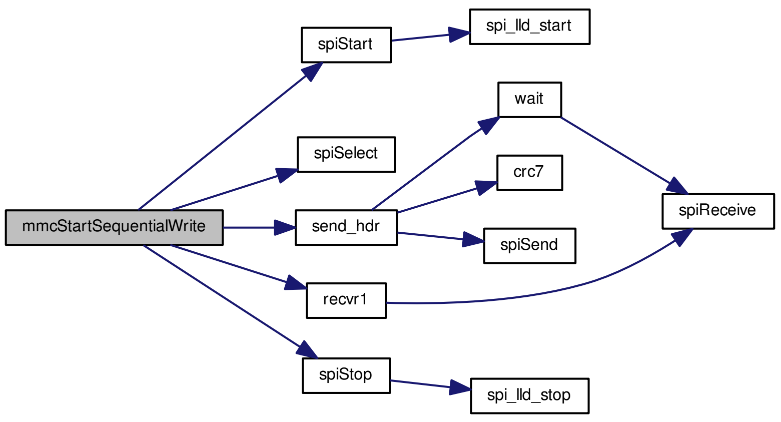
Return values

HAL_SUCCESS	the operation succeeded.
HAL_FAILED	the operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.27.5.20 bool mmcSequentialWrite (**MMCDriver** ∗ *mmcp*, const uint8_t ∗ *buffer*)

Writes a block within a sequential write operation.

Parameters

`in`	*mmcp*	pointer to the `MMCDriver` object
`out`	*buffer*	pointer to the write buffer

Returns

The operation status.

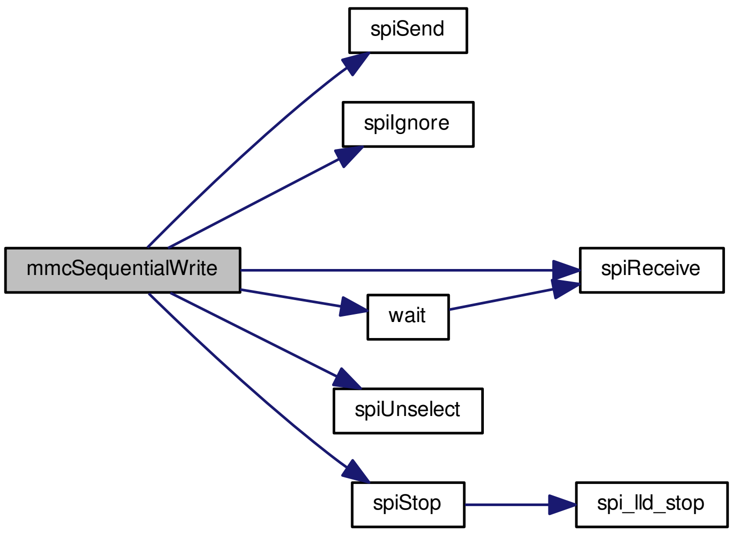
Return values

HAL_SUCCESS	the operation succeeded.
HAL_FAILED	the operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.27.5.21 bool mmcStopSequentialWrite (**MMCDriver** ∗ *mmcp*)

Stops a sequential write gracefully.

Parameters

in	*mmcp*	pointer to the MMCDriver object

Returns

The operation status.

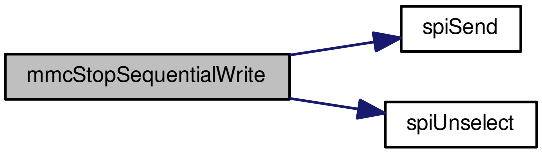
Return values

HAL_SUCCESS	the operation succeeded.
HAL_FAILED	the operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.27.5.22 bool mmcSync (**MMCDriver** ∗ *mmcp*)

Waits for card idle condition.

Parameters

`in`	*mmcp*	pointer to the `MMCDriver` object

Returns

The operation status.

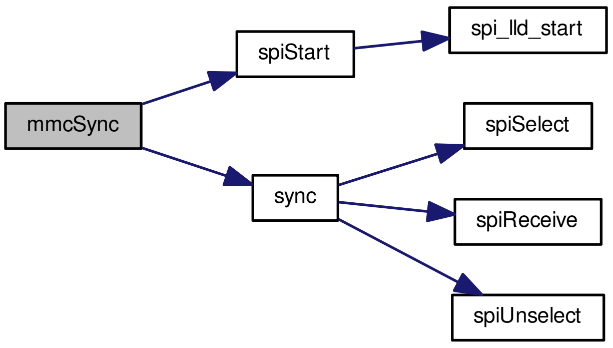
Return values

HAL_SUCCESS	the operation succeeded.
HAL_FAILED	the operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.27.5.23 bool mmcGetInfo (**MMCDriver** ∗ *mmcp,* **BlockDeviceInfo** ∗ *bdip*)

Returns the media info.

Parameters

in	*mmcp*	pointer to the MMCDriver object
out	*bdip*	pointer to a BlockDeviceInfo structure

Returns

The operation status.

Return values

HAL_SUCCESS	the operation succeeded.
HAL_FAILED	the operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.27.5.24 bool mmcErase (**MMCDriver** ∗ *mmcp,* uint32_t *startblk,* uint32_t *endblk*)

Erases blocks.

Parameters

in	*mmcp*	pointer to the MMCDriver object
in	*startblk*	starting block number
in	*endblk*	ending block number

Returns

The operation status.

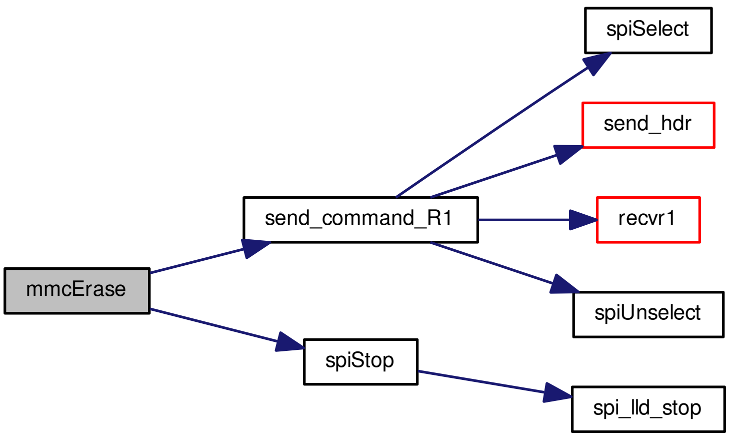
Return values

HAL_SUCCESS	the operation succeeded.
HAL_FAILED	the operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.27.6 Variable Documentation

7.27.6.1 const struct MMCDriverVMT mmc_vmt [static]

Initial value:

```
= {
   (bool (*)(void *))mmc_lld_is_card_inserted,
   (bool (*)(void *))mmc_lld_is_write_protected,
   (bool (*)(void *))mmcConnect,
   (bool (*)(void *))mmcDisconnect,
  mmc_read,
  mmc_write,
   (bool (*)(void *))mmcSync,
   (bool (*)(void *, BlockDeviceInfo *))mmcGetInfo
}
```

Virtual methods table.

7.27.6.2 const uint8_t crc7_lookup_table[256] [static]

Initial value:

```
= {
  0x00, 0x09, 0x12, 0x1b, 0x24, 0x2d, 0x36, 0x3f, 0x48, 0x41, 0x5a, 0x53,
  0x6c, 0x65, 0x7e, 0x77, 0x19, 0x10, 0x0b, 0x02, 0x3d, 0x34, 0x2f, 0x26,
  0x51, 0x58, 0x43, 0x4a, 0x75, 0x7c, 0x67, 0x6e, 0x32, 0x3b, 0x20, 0x29,
  0x16, 0x1f, 0x04, 0x0d, 0x7a, 0x73, 0x68, 0x61, 0x5e, 0x57, 0x4c, 0x45,
  0x2b, 0x22, 0x39, 0x30, 0x0f, 0x06, 0x1d, 0x14, 0x63, 0x6a, 0x71, 0x78,
  0x47, 0x4e, 0x55, 0x5c, 0x64, 0x6d, 0x76, 0x7f, 0x40, 0x49, 0x52, 0x5b,
  0x2c, 0x25, 0x3e, 0x37, 0x08, 0x01, 0x1a, 0x13, 0x7d, 0x74, 0x6f, 0x66,
  0x59, 0x50, 0x4b, 0x42, 0x35, 0x3c, 0x27, 0x2e, 0x11, 0x18, 0x03, 0x0a,
  0x56, 0x5f, 0x44, 0x4d, 0x72, 0x7b, 0x60, 0x69, 0x1e, 0x17, 0x0c, 0x05,
  0x3a, 0x33, 0x28, 0x21, 0x4f, 0x46, 0x5d, 0x54, 0x6b, 0x62, 0x79, 0x70,
  0x07, 0x0e, 0x15, 0x1c, 0x23, 0x2a, 0x31, 0x38, 0x41, 0x48, 0x53, 0x5a,
  0x65, 0x6c, 0x77, 0x7e, 0x09, 0x00, 0x1b, 0x12, 0x2d, 0x24, 0x3f, 0x36,
  0x58, 0x51, 0x4a, 0x43, 0x7c, 0x75, 0x6e, 0x67, 0x10, 0x19, 0x02, 0x0b,
  0x34, 0x3d, 0x26, 0x2f, 0x73, 0x7a, 0x61, 0x68, 0x57, 0x5e, 0x45, 0x4c,
  0x3b, 0x32, 0x29, 0x20, 0x1f, 0x16, 0x0d, 0x04, 0x6a, 0x63, 0x78, 0x71,
  0x4e, 0x47, 0x5c, 0x55, 0x22, 0x2b, 0x30, 0x39, 0x06, 0x0f, 0x14, 0x1d,
  0x25, 0x2c, 0x37, 0x3e, 0x01, 0x08, 0x13, 0x1a, 0x6d, 0x64, 0x7f, 0x76,
  0x49, 0x40, 0x5b, 0x52, 0x3c, 0x35, 0x2e, 0x27, 0x18, 0x11, 0x0a, 0x03,
  0x74, 0x7d, 0x66, 0x6f, 0x50, 0x59, 0x42, 0x4b, 0x17, 0x1e, 0x05, 0x0c,
  0x33, 0x3a, 0x21, 0x28, 0x5f, 0x56, 0x4d, 0x44, 0x7b, 0x72, 0x69, 0x60,
  0x0e, 0x07, 0x1c, 0x15, 0x2a, 0x23, 0x38, 0x31, 0x46, 0x4f, 0x54, 0x5d,
  0x62, 0x6b, 0x70, 0x79
}
```

Lookup table for CRC-7 (based on polynomial x^7 + x^3 + 1).

7.28 MMC/SD Block Device

7.28.1 Detailed Description

This module implements a common ancestor for all device drivers accessing MMC or SD cards. This interface inherits the state machine and the interface from the Abstract I/O Block Device module.

Macros

- #define MMCSD_BLOCK_SIZE 512U

  Fixed block size for MMC/SD block devices.
- #define MMCSD_R1_ERROR_MASK 0xFDFFE008U

  Mask of error bits in R1 responses.
- #define MMCSD_CMD8_PATTERN 0x000001AAU

  Fixed pattern for CMD8.
- #define _mmcsd_block_device_methods _base_block_device_methods

  `MMCSDBlockDevice` *specific methods.*
- #define _mmcsd_block_device_data

  `MMCSDBlockDevice` *specific data.*

SD/MMC status conditions

- #define **MMCSD_STS_IDLE** 0U
- #define **MMCSD_STS_READY** 1U
- #define **MMCSD_STS_IDENT** 2U
- #define **MMCSD_STS_STBY** 3U
- #define **MMCSD_STS_TRAN** 4U
- #define **MMCSD_STS_DATA** 5U
- #define **MMCSD_STS_RCV** 6U
- #define **MMCSD_STS_PRG** 7U
- #define **MMCSD_STS_DIS** 8U

SD/MMC commands

- #define **MMCSD_CMD_GO_IDLE_STATE** 0U
- #define **MMCSD_CMD_INIT** 1U
- #define **MMCSD_CMD_ALL_SEND_CID** 2U
- #define **MMCSD_CMD_SEND_RELATIVE_ADDR** 3U
- #define **MMCSD_CMD_SET_BUS_WIDTH** 6U
- #define **MMCSD_CMD_SWITCH** MMCSD_CMD_SET_BUS_WIDTH
- #define **MMCSD_CMD_SEL_DESEL_CARD** 7U
- #define **MMCSD_CMD_SEND_IF_COND** 8U
- #define **MMCSD_CMD_SEND_EXT_CSD** MMCSD_CMD_SEND_IF_COND
- #define **MMCSD_CMD_SEND_CSD** 9U
- #define **MMCSD_CMD_SEND_CID** 10U
- #define **MMCSD_CMD_STOP_TRANSMISSION** 12U
- #define **MMCSD_CMD_SEND_STATUS** 13U
- #define **MMCSD_CMD_SET_BLOCKLEN** 16U
- #define **MMCSD_CMD_READ_SINGLE_BLOCK** 17U
- #define **MMCSD_CMD_READ_MULTIPLE_BLOCK** 18U
- #define **MMCSD_CMD_SET_BLOCK_COUNT** 23U
- #define **MMCSD_CMD_WRITE_BLOCK** 24U

- #define **MMCSD_CMD_WRITE_MULTIPLE_BLOCK** 25U
- #define **MMCSD_CMD_ERASE_RW_BLK_START** 32U
- #define **MMCSD_CMD_ERASE_RW_BLK_END** 33U
- #define **MMCSD_CMD_ERASE** 38U
- #define **MMCSD_CMD_APP_OP_COND** 41U
- #define **MMCSD_CMD_LOCK_UNLOCK** 42U
- #define **MMCSD_CMD_APP_CMD** 55U
- #define **MMCSD_CMD_READ_OCR** 58U

CSD record offsets

- #define MMCSD_CSD_MMC_CSD_STRUCTURE_SLICE 127U,126U

    Slice position of values in CSD register.
- #define **MMCSD_CSD_MMC_SPEC_VERS_SLICE** 125U,122U
- #define **MMCSD_CSD_MMC_TAAC_SLICE** 119U,112U
- #define **MMCSD_CSD_MMC_NSAC_SLICE** 111U,104U
- #define **MMCSD_CSD_MMC_TRAN_SPEED_SLICE** 103U,96U
- #define **MMCSD_CSD_MMC_CCC_SLICE** 95U,84U
- #define **MMCSD_CSD_MMC_READ_BL_LEN_SLICE** 83U,80U
- #define **MMCSD_CSD_MMC_READ_BL_PARTIAL_SLICE** 79U,79U
- #define **MMCSD_CSD_MMC_WRITE_BLK_MISALIGN_SLICE** 78U,78U
- #define **MMCSD_CSD_MMC_READ_BLK_MISALIGN_SLICE** 77U,77U
- #define **MMCSD_CSD_MMC_DSR_IMP_SLICE** 76U,76U
- #define **MMCSD_CSD_MMC_C_SIZE_SLICE** 73U,62U
- #define **MMCSD_CSD_MMC_VDD_R_CURR_MIN_SLICE** 61U,59U
- #define **MMCSD_CSD_MMC_VDD_R_CURR_MAX_SLICE** 58U,56U
- #define **MMCSD_CSD_MMC_VDD_W_CURR_MIN_SLICE** 55U,53U
- #define **MMCSD_CSD_MMC_VDD_W_CURR_MAX_SLICE** 52U,50U
- #define **MMCSD_CSD_MMC_C_SIZE_MULT_SLICE** 49U,47U
- #define **MMCSD_CSD_MMC_ERASE_GRP_SIZE_SLICE** 46U,42U
- #define **MMCSD_CSD_MMC_ERASE_GRP_MULT_SLICE** 41U,37U
- #define **MMCSD_CSD_MMC_WP_GRP_SIZE_SLICE** 36U,32U
- #define **MMCSD_CSD_MMC_WP_GRP_ENABLE_SLICE** 31U,31U
- #define **MMCSD_CSD_MMC_DEFAULT_ECC_SLICE** 30U,29U
- #define **MMCSD_CSD_MMC_R2W_FACTOR_SLICE** 28U,26U
- #define **MMCSD_CSD_MMC_WRITE_BL_LEN_SLICE** 25U,22U
- #define **MMCSD_CSD_MMC_WRITE_BL_PARTIAL_SLICE** 21U,21U
- #define **MMCSD_CSD_MMC_CONTENT_PROT_APP_SLICE** 16U,16U
- #define **MMCSD_CSD_MMC_FILE_FORMAT_GRP_SLICE** 15U,15U
- #define **MMCSD_CSD_MMC_COPY_SLICE** 14U,14U
- #define **MMCSD_CSD_MMC_PERM_WRITE_PROTECT_SLICE** 13U,13U
- #define **MMCSD_CSD_MMC_TMP_WRITE_PROTECT_SLICE** 12U,12U
- #define **MMCSD_CSD_MMC_FILE_FORMAT_SLICE** 11U,10U
- #define **MMCSD_CSD_MMC_ECC_SLICE** 9U,8U
- #define **MMCSD_CSD_MMC_CRC_SLICE** 7U,1U
- #define **MMCSD_CSD_20_CRC_SLICE** 7U,1U
- #define **MMCSD_CSD_20_FILE_FORMAT_SLICE** 11U,10U
- #define **MMCSD_CSD_20_TMP_WRITE_PROTECT_SLICE** 12U,12U
- #define **MMCSD_CSD_20_PERM_WRITE_PROTECT_SLICE** 13U,13U
- #define **MMCSD_CSD_20_COPY_SLICE** 14U,14U
- #define **MMCSD_CSD_20_FILE_FORMAT_GRP_SLICE** 15U,15U
- #define **MMCSD_CSD_20_WRITE_BL_PARTIAL_SLICE** 21U,21U
- #define **MMCSD_CSD_20_WRITE_BL_LEN_SLICE** 25U,12U
- #define **MMCSD_CSD_20_R2W_FACTOR_SLICE** 28U,26U

- #define **MMCSD_CSD_20_WP_GRP_ENABLE_SLICE** 31U,31U
- #define **MMCSD_CSD_20_WP_GRP_SIZE_SLICE** 38U,32U
- #define **MMCSD_CSD_20_ERASE_SECTOR_SIZE_SLICE** 45U,39U
- #define **MMCSD_CSD_20_ERASE_BLK_EN_SLICE** 46U,46U
- #define **MMCSD_CSD_20_C_SIZE_SLICE** 69U,48U
- #define **MMCSD_CSD_20_DSR_IMP_SLICE** 76U,76U
- #define **MMCSD_CSD_20_READ_BLK_MISALIGN_SLICE** 77U,77U
- #define **MMCSD_CSD_20_WRITE_BLK_MISALIGN_SLICE** 78U,78U
- #define **MMCSD_CSD_20_READ_BL_PARTIAL_SLICE** 79U,79U
- #define **MMCSD_CSD_20_READ_BL_LEN_SLICE** 83U,80U
- #define **MMCSD_CSD_20_CCC_SLICE** 95U,84U
- #define **MMCSD_CSD_20_TRANS_SPEED_SLICE** 103U,96U
- #define **MMCSD_CSD_20_NSAC_SLICE** 111U,104U
- #define **MMCSD_CSD_20_TAAC_SLICE** 119U,112U
- #define **MMCSD_CSD_20_CSD_STRUCTURE_SLICE** 127U,126U
- #define **MMCSD_CSD_10_CRC_SLICE** MMCSD_CSD_20_CRC_SLICE
- #define **MMCSD_CSD_10_FILE_FORMAT_SLICE** MMCSD_CSD_20_FILE_FORMAT_SLICE
- #define **MMCSD_CSD_10_TMP_WRITE_PROTECT_SLICE** MMCSD_CSD_20_TMP_WRITE_PROTEC↩
T_SLICE
- #define **MMCSD_CSD_10_PERM_WRITE_PROTECT_SLICE** MMCSD_CSD_20_PERM_WRITE_PROT↩
ECT_SLICE
- #define **MMCSD_CSD_10_COPY_SLICE** MMCSD_CSD_20_COPY_SLICE
- #define **MMCSD_CSD_10_FILE_FORMAT_GRP_SLICE** MMCSD_CSD_20_FILE_FORMAT_GRP_SLICE
- #define **MMCSD_CSD_10_WRITE_BL_PARTIAL_SLICE** MMCSD_CSD_20_WRITE_BL_PARTIAL_SLICE
- #define **MMCSD_CSD_10_WRITE_BL_LEN_SLICE** MMCSD_CSD_20_WRITE_BL_LEN_SLICE
- #define **MMCSD_CSD_10_R2W_FACTOR_SLICE** MMCSD_CSD_20_R2W_FACTOR_SLICE
- #define **MMCSD_CSD_10_WP_GRP_ENABLE_SLICE** MMCSD_CSD_20_WP_GRP_ENABLE_SLICE
- #define **MMCSD_CSD_10_WP_GRP_SIZE_SLICE** MMCSD_CSD_20_WP_GRP_SIZE_SLICE
- #define **MMCSD_CSD_10_ERASE_SECTOR_SIZE_SLICE** MMCSD_CSD_20_ERASE_SECTOR_SIZE↩
_SLICE
- #define **MMCSD_CSD_10_ERASE_BLK_EN_SLICE** MMCSD_CSD_20_ERASE_BLK_EN_SLICE
- #define **MMCSD_CSD_10_C_SIZE_MULT_SLICE** 49U,47U
- #define **MMCSD_CSD_10_VDD_W_CURR_MAX_SLICE** 52U,50U
- #define **MMCSD_CSD_10_VDD_W_CURR_MIN_SLICE** 55U,53U
- #define **MMCSD_CSD_10_VDD_R_CURR_MAX_SLICE** 58U,56U
- #define **MMCSD_CSD_10_VDD_R_CURR_MIX_SLICE** 61U,59U
- #define **MMCSD_CSD_10_C_SIZE_SLICE** 73U,62U
- #define **MMCSD_CSD_10_DSR_IMP_SLICE** MMCSD_CSD_20_DSR_IMP_SLICE
- #define **MMCSD_CSD_10_READ_BLK_MISALIGN_SLICE** MMCSD_CSD_20_READ_BLK_MISALIGN_↩
SLICE
- #define **MMCSD_CSD_10_WRITE_BLK_MISALIGN_SLICE** MMCSD_CSD_20_WRITE_BLK_MISALIG↩
N_SLICE
- #define **MMCSD_CSD_10_READ_BL_PARTIAL_SLICE** MMCSD_CSD_20_READ_BL_PARTIAL_SLICE
- #define **MMCSD_CSD_10_READ_BL_LEN_SLICE** 83U,80U
- #define **MMCSD_CSD_10_CCC_SLICE** MMCSD_CSD_20_CCC_SLICE
- #define **MMCSD_CSD_10_TRANS_SPEED_SLICE** MMCSD_CSD_20_TRANS_SPEED_SLICE
- #define **MMCSD_CSD_10_NSAC_SLICE** MMCSD_CSD_20_NSAC_SLICE
- #define **MMCSD_CSD_10_TAAC_SLICE** MMCSD_CSD_20_TAAC_SLICE
- #define **MMCSD_CSD_10_CSD_STRUCTURE_SLICE** MMCSD_CSD_20_CSD_STRUCTURE_SLICE

CID record offsets

- #define MMCSD_CID_SDC_CRC_SLICE 7U,1U

  Slice position of values in CID register.
- #define **MMCSD_CID_SDC_MDT_M_SLICE** 11U,8U
- #define **MMCSD_CID_SDC_MDT_Y_SLICE** 19U,12U
- #define **MMCSD_CID_SDC_PSN_SLICE** 55U,24U
- #define **MMCSD_CID_SDC_PRV_M_SLICE** 59U,56U
- #define **MMCSD_CID_SDC_PRV_N_SLICE** 63U,60U
- #define **MMCSD_CID_SDC_PNM0_SLICE** 71U,64U
- #define **MMCSD_CID_SDC_PNM1_SLICE** 79U,72U
- #define **MMCSD_CID_SDC_PNM2_SLICE** 87U,80U
- #define **MMCSD_CID_SDC_PNM3_SLICE** 95U,88U
- #define **MMCSD_CID_SDC_PNM4_SLICE** 103U,96U
- #define **MMCSD_CID_SDC_OID_SLICE** 119U,104U
- #define **MMCSD_CID_SDC_MID_SLICE** 127U,120U
- #define **MMCSD_CID_MMC_CRC_SLICE** 7U,1U
- #define **MMCSD_CID_MMC_MDT_Y_SLICE** 11U,8U
- #define **MMCSD_CID_MMC_MDT_M_SLICE** 15U,12U
- #define **MMCSD_CID_MMC_PSN_SLICE** 47U,16U
- #define **MMCSD_CID_MMC_PRV_M_SLICE** 51U,48U
- #define **MMCSD_CID_MMC_PRV_N_SLICE** 55U,52U
- #define **MMCSD_CID_MMC_PNM0_SLICE** 63U,56U
- #define **MMCSD_CID_MMC_PNM1_SLICE** 71U,64U
- #define **MMCSD_CID_MMC_PNM2_SLICE** 79U,72U
- #define **MMCSD_CID_MMC_PNM3_SLICE** 87U,80U
- #define **MMCSD_CID_MMC_PNM4_SLICE** 95U,88U
- #define **MMCSD_CID_MMC_PNM5_SLICE** 103U,96U
- #define **MMCSD_CID_MMC_OID_SLICE** 119U,104U
- #define **MMCSD_CID_MMC_MID_SLICE** 127U,120U

R1 response utilities

- #define MMCSD_R1_ERROR(r1) (((r1) & MMCSD_R1_ERROR_MASK) != 0U)

  Evaluates to `TRUE` if the R1 response contains error flags.
- #define MMCSD_R1_STS(r1) (((r1) >> 9U) & 15U)

  Returns the status field of an R1 response.
- #define MMCSD_R1_IS_CARD_LOCKED(r1) ((((r1) >> 21U) & 1U) != 0U)

  Evaluates to `TRUE` if the R1 response indicates a locked card.

Macro Functions

- #define mmcsdGetCardCapacity(ip) ((ip)->capacity)

  Returns the card capacity in blocks.

Data Structures

- struct MMCSDBlockDeviceVMT

  `MMCSDBlockDevice` virtual methods table.
- struct MMCSDBlockDevice

  MCC/SD block device class.
- struct unpacked_sdc_cid_t

Unpacked CID register from SDC.
- struct unpacked_mmc_cid_t

  Unpacked CID register from MMC.
- struct unpacked_sdc_csd_10_t

  Unpacked CSD v1.0 register from SDC.
- struct unpacked_sdc_csd_20_t

  Unpacked CSD v2.0 register from SDC.
- struct unpacked_mmc_csd_t

  Unpacked CSD register from MMC.

Functions

- uint32_t _mmcsd_get_slice (const uint32_t ∗data, uint32_t end, uint32_t start)

  Gets a bit field from a words array.
- uint32_t _mmcsd_get_capacity (const uint32_t ∗csd)

  Extract card capacity from a CSD.
- uint32_t _mmcsd_get_capacity_ext (const uint8_t ∗ext_csd)

  Extract MMC card capacity from EXT_CSD.
- void _mmcsd_unpack_sdc_cid (const MMCSDBlockDevice ∗sdcp, unpacked_sdc_cid_t ∗cidsdc)

  Unpacks SDC CID array in structure.
- void _mmcsd_unpack_mmc_cid (const MMCSDBlockDevice ∗sdcp, unpacked_mmc_cid_t ∗cidmmc)

  Unpacks MMC CID array in structure.
- void _mmcsd_unpack_csd_mmc (const MMCSDBlockDevice ∗sdcp, unpacked_mmc_csd_t ∗csdmmc)

  Unpacks MMC CSD array in structure.
- void _mmcsd_unpack_csd_v10 (const MMCSDBlockDevice ∗sdcp, unpacked_sdc_csd_10_t ∗csd10)

  Unpacks SDC CSD v1.0 array in structure.
- void _mmcsd_unpack_csd_v20 (const MMCSDBlockDevice ∗sdcp, unpacked_sdc_csd_20_t ∗csd20)

  Unpacks SDC CSD v2.0 array in structure.

7.28.2 Macro Definition Documentation

7.28.2.1 #define MMCSD_BLOCK_SIZE 512U

Fixed block size for MMC/SD block devices.

7.28.2.2 #define MMCSD_R1_ERROR_MASK 0xFDFFE008U

Mask of error bits in R1 responses.

7.28.2.3 #define MMCSD_CMD8_PATTERN 0x000001AAU

Fixed pattern for CMD8.

7.28.2.4 #define MMCSD_CSD_MMC_CSD_STRUCTURE_SLICE 127U,126U

Slice position of values in CSD register.

7.28.2.5 #define MMCSD_CID_SDC_CRC_SLICE 7U,1U

Slice position of values in CID register.

7.28.2.6 #define _mmcsd_block_device_methods _base_block_device_methods

MMCSDBlockDevice specific methods.

7.28.2.7 #define _mmcsd_block_device_data

Value:

```
_base_block_device_data                                                          \
  /* Card CID.*/                                                                  \
  uint32_t              cid[4];                                                  \
  /* Card CSD.*/                                                                  \
  uint32_t              csd[4];                                                  \
  /* Total number of blocks in card.*/                                            \
  uint32_t              capacity;
```

MMCSDBlockDevice specific data.

Note

It is empty because MMCSDBlockDevice is only an interface without implementation.

7.28.2.8 #define MMCSD_R1_ERROR(*r1*) (((r1) & MMCSD_R1_ERROR_MASK) != 0U)

Evaluates to TRUE if the R1 response contains error flags.

Parameters

in	*r1*	the r1 response

7.28.2.9 #define MMCSD_R1_STS(*r1*) (((r1) >> 9U) & 15U)

Returns the status field of an R1 response.

Parameters

in	*r1*	the r1 response

7.28.2.10 #define MMCSD_R1_IS_CARD_LOCKED(*r1*) ((((r1) >> 21U) & 1U) != 0U)

Evaluates to TRUE if the R1 response indicates a locked card.

Parameters

in	*r1*	the r1 response

7.28.2.11 #define mmcsdGetCardCapacity(*ip*) ((ip)->capacity)

Returns the card capacity in blocks.

Parameters

in	*ip*	pointer to a `MMCSDBlockDevice` or derived class

Returns

The card capacity.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.28.3 Function Documentation

7.28.3.1 uint32_t _mmcsd_get_slice (const uint32_t ∗ *data,* uint32_t *end,* uint32_t *start*)

Gets a bit field from a words array.

Note

The bit zero is the LSb of the first word.

Parameters

in	*data*	pointer to the words array
in	*end*	bit offset of the last bit of the field, inclusive
in	*start*	bit offset of the first bit of the field, inclusive

Returns

The bits field value, left aligned.

Function Class:

Not an API, this function is for internal use only.

7.28.3.2 uint32_t _mmcsd_get_capacity (const uint32_t ∗ *csd*)

Extract card capacity from a CSD.

The capacity is returned as number of available blocks.

Parameters

in	*csd*	the CSD record

Returns

The card capacity.

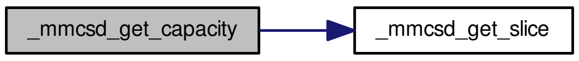
Return values

0	CSD format error

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.28.3.3 uint32_t _mmcsd_get_capacity_ext (const uint8_t ∗ *ext_csd*)

Extract MMC card capacity from EXT_CSD.

The capacity is returned as number of available blocks.

Parameters

in	*ext_csd*	the extended CSD record

Returns

The card capacity.

Function Class:

Not an API, this function is for internal use only.

7.28.3.4 void _mmcsd_unpack_sdc_cid (const MMCSDBlockDevice ∗ *sdcp*, unpacked_sdc_cid_t ∗ *cidsdc*)

Unpacks SDC CID array in structure.

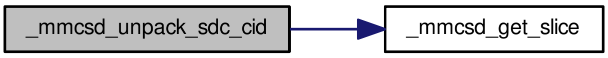
Parameters

in	*sdcp*	pointer to the `MMCSDBlockDevice` object
out	*cidsdc*	pointer to the `unpacked_sdc_cid_t` object

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.28.3.5 void _mmcsd_unpack_mmc_cid (const **MMCSDBlockDevice** ∗ *sdcp,* **unpacked_mmc_cid_t** ∗ *cidmmc*)

Unpacks MMC CID array in structure.

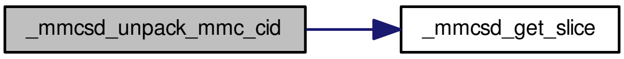
Parameters

`in`	*sdcp*	pointer to the `MMCSDBlockDevice` object
`out`	*cidmmc*	pointer to the `unpacked_mmc_cid_t` object

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.28.3.6 void _mmcsd_unpack_csd_mmc (const **MMCSDBlockDevice** ∗ *sdcp,* **unpacked_mmc_csd_t** ∗ *csdmmc*)

Unpacks MMC CSD array in structure.

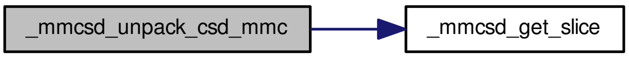
Parameters

`in`	*sdcp*	pointer to the `MMCSDBlockDevice` object
`out`	*csdmmc*	pointer to the `unpacked_mmc_csd_t` object

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.28.3.7 void _mmcsd_unpack_csd_v10 (const **MMCSDBlockDevice** * *sdcp*, **unpacked_sdc_csd_10_t** * *csd10*)

Unpacks SDC CSD v1.0 array in structure.

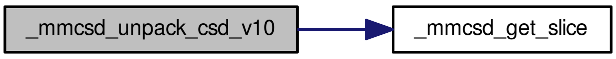
Parameters

in	*sdcp*	pointer to the `MMCSDBlockDevice` object
out	*csd10*	pointer to the `unpacked_sdc_csd_10_t` object

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.28.3.8 void _mmcsd_unpack_csd_v20 (const **MMCSDBlockDevice** * *sdcp*, **unpacked_sdc_csd_20_t** * *csd20*)

Unpacks SDC CSD v2.0 array in structure.

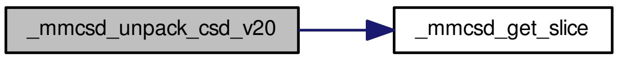
Parameters

in	*sdcp*	pointer to the `MMCSDBlockDevice` object
out	*csd20*	pointer to the `unpacked_sdc_csd_20_t` object

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.29 PAL Driver

I/O Ports Abstraction Layer.

7.29.1 Detailed Description

I/O Ports Abstraction Layer.

This module defines an abstract interface for digital I/O ports. Note that most I/O ports functions are just macros. The macros have default software implementations that can be redefined in a PAL Low Level Driver if the target hardware supports special features like, for example, atomic bit set/reset/masking. Please refer to the ports specific documentation for details.
The PAL Driver driver has the advantage to make the access to the I/O ports platform independent and still be optimized for the specific architectures.
Note that the PAL Low Level Driver may also offer non standard macro and functions in order to support specific features but, of course, the use of such interfaces would not be portable. Such interfaces shall be marked with the architecture name inside the function names.

Precondition

In order to use the PAL driver the `HAL_USE_PAL` option must be enabled in `halconf.h`.

7.29.2 Implementation Rules

In implementing a PAL Low Level Driver there are some rules/behaviors that should be respected.

7.29.2.1 Writing on input pads

The behavior is not specified but there are implementations better than others, this is the list of possible implementations, preferred options are on top:

1. The written value is not actually output but latched, should the pads be reprogrammed as outputs the value would be in effect.
2. The write operation is ignored.
3. The write operation has side effects, as example disabling/enabling pull up/down resistors or changing the pad direction. This scenario is discouraged, please try to avoid this scenario.

7.29.2.2 Reading from output pads

The behavior is not specified but there are implementations better than others, this is the list of possible implementations, preferred options are on top:

1. The actual pads states are read (not the output latch).
2. The output latch value is read (regardless of the actual pads states).
3. Unspecified, please try to avoid this scenario.

7.29.2.3 Writing unused or unimplemented port bits

The behavior is not specified.

7.29.2.4 Reading from unused or unimplemented port bits

The behavior is not specified.

7.29.2.5 Reading or writing on pins associated to other functionalities

The behavior is not specified.

Macros

- #define PAL_PORT_BIT(n) ((ioportmask_t)(1U << (n)))

  Port bit helper macro.
- #define PAL_GROUP_MASK(width) ((ioportmask_t)(1U << (width)) - 1U)

  Bits group mask helper.
- #define _IOBUS_DATA(name, port, width, offset) {port, PAL_GROUP_MASK(width), offset}

  Data part of a static I/O bus initializer.
- #define IOBUS_DECL(name, port, width, offset) IOBus name = _IOBUS_DATA(name, port, width, offset)

  Static I/O bus initializer.
- #define IOPORT1 0

  First I/O port identifier.
- #define pal_lld_init(config) _pal_lld_init(config)

  Low level PAL subsystem initialization.
- #define pal_lld_readport(port) 0U

  Reads the physical I/O port states.
- #define pal_lld_readlatch(port) 0U

  Reads the output latch.
- #define pal_lld_writeport(port, bits)

  Writes a bits mask on a I/O port.
- #define pal_lld_setport(port, bits)

  Sets a bits mask on a I/O port.
- #define pal_lld_clearport(port, bits)

  Clears a bits mask on a I/O port.
- #define pal_lld_toggleport(port, bits)

  Toggles a bits mask on a I/O port.
- #define pal_lld_readgroup(port, mask, offset) 0U

  Reads a group of bits.
- #define pal_lld_writegroup(port, mask, offset, bits)

  Writes a group of bits.
- #define pal_lld_setgroupmode(port, mask, offset, mode) _pal_lld_setgroupmode(port, mask << offset, mode)

  Pads group mode setup.
- #define pal_lld_readpad(port, pad) PAL_LOW

  Reads a logical state from an I/O pad.
- #define pal_lld_writepad(port, pad, bit)

  Writes a logical state on an output pad.
- #define pal_lld_setpad(port, pad)

  Sets a pad logical state to `PAL_HIGH`.
- #define pal_lld_clearpad(port, pad)

  Clears a pad logical state to `PAL_LOW`.
- #define pal_lld_togglepad(port, pad)

  Toggles a pad logical state.
- #define pal_lld_setpadmode(port, pad, mode)

  Pad mode setup.

Pads mode constants

- #define PAL_MODE_RESET 0U

  After reset state.
- #define PAL_MODE_UNCONNECTED 1U

  *Safe state for **unconnected** pads.*
- #define PAL_MODE_INPUT 2U

  Regular input high-Z pad.
- #define PAL_MODE_INPUT_PULLUP 3U

  Input pad with weak pull up resistor.
- #define PAL_MODE_INPUT_PULLDOWN 4U

  Input pad with weak pull down resistor.
- #define PAL_MODE_INPUT_ANALOG 5U

  Analog input mode.
- #define PAL_MODE_OUTPUT_PUSHPULL 6U

  Push-pull output pad.
- #define PAL_MODE_OUTPUT_OPENDRAIN 7U

  Open-drain output pad.

Logic level constants

- #define PAL_LOW 0U

  Logical low state.
- #define PAL_HIGH 1U

  Logical high state.

PAL event modes

- #define PAL_EVENT_MODE_EDGES_MASK 3U

  Mask of edges field.
- #define PAL_EVENT_MODE_DISABLED 0U

  Channel disabled.
- #define PAL_EVENT_MODE_RISING_EDGE 1U

  Rising edge callback.
- #define PAL_EVENT_MODE_FALLING_EDGE 2U

  Falling edge callback.
- #define PAL_EVENT_MODE_BOTH_EDGES 3U

  Both edges callback.

Macro Functions

- #define palInit(config) pal_lld_init(config)

  PAL subsystem initialization.
- #define palReadPort(port) ((void)(port), 0U)

  Reads the physical I/O port states.
- #define palReadLatch(port) ((void)(port), 0U)

  Reads the output latch.
- #define palWritePort(port, bits) ((void)(port), (void)(bits))

  Writes a bits mask on a I/O port.
- #define palSetPort(port, bits) palWritePort(port, palReadLatch(port) | (bits))

Sets a bits mask on a I/O port.
- #define palClearPort(port, bits) palWritePort(port, palReadLatch(port) & ∼(bits))

  Clears a bits mask on a I/O port.
- #define palTogglePort(port, bits) palWritePort(port, palReadLatch(port) ^ (bits))

  Toggles a bits mask on a I/O port.
- #define palReadGroup(port, mask, offset) ((palReadPort(port) >> (offset)) & (mask))

  Reads a group of bits.
- #define palWriteGroup(port, mask, offset, bits)

  Writes a group of bits.
- #define palSetGroupMode(port, mask, offset, mode)

  Pads group mode setup.
- #define palReadPad(port, pad) ((palReadPort(port) >> (pad)) & 1U)

  Reads an input pad logic state.
- #define palWritePad(port, pad, bit)

  Writes a logic state on an output pad.
- #define palSetPad(port, pad) palSetPort(port, PAL_PORT_BIT(pad))

  Sets a pad logic state to `PAL_HIGH`.
- #define palClearPad(port, pad) palClearPort(port, PAL_PORT_BIT(pad))

  Clears a pad logic state to `PAL_LOW`.
- #define palTogglePad(port, pad) palTogglePort(port, PAL_PORT_BIT(pad))

  Toggles a pad logic state.
- #define palSetPadMode(port, pad, mode) palSetGroupMode(port, PAL_PORT_BIT(pad), 0U, mode)

  Pad mode setup.
- #define palPadEnableEventI(port, pad, mode, callback)

  Pad event enable.
- #define palPadDisableEventI(port, pad)

  Pad event disable.
- #define palReadLine(line) palReadPad(PAL_PORT(line), PAL_PAD(line))

  Reads an input line logic state.
- #define palWriteLine(line, bit) palWritePad(PAL_PORT(line), PAL_PAD(line), bit)

  Writes a logic state on an output line.
- #define palSetLine(line) palSetPad(PAL_PORT(line), PAL_PAD(line))

  Sets a line logic state to `PAL_HIGH`.
- #define palClearLine(line) palClearPad(PAL_PORT(line), PAL_PAD(line))

  Clears a line logic state to `PAL_LOW`.
- #define palToggleLine(line) palTogglePad(PAL_PORT(line), PAL_PAD(line))

  Toggles a line logic state.
- #define palSetLineMode(line, mode) palSetPadMode(PAL_PORT(line), PAL_PAD(line), mode)

  Line mode setup.
- #define palLineEnableEventI(line, mode, callback) palPadEnableEventI(PAL_PORT(line), PAL_PAD(line), mode, callback)

  Line event enable.
- #define palLineDisableEventI(line) palPadDisableEventI(PAL_PORT(line), PAL_PAD(line))

  Line event disable.

Port related definitions

- #define PAL_IOPORTS_WIDTH 16U

  Width, in bits, of an I/O port.
- #define PAL_WHOLE_PORT ((ioportmask_t)0xFFFFU)

  Whole port mask.

Line handling macros

- #define PAL_LINE(port, pad) ((ioline_t)((uint32_t)(port)) | ((uint32_t)(pad)))

  Forms a line identifier.
- #define PAL_PORT(line) ((stm32_gpio_t ∗)(((uint32_t)(line)) & 0xFFFFFFF0U))

  Decodes a port identifier from a line identifier.
- #define PAL_PAD(line) ((uint32_t)((uint32_t)(line) & 0x0000000FU))

  Decodes a pad identifier from a line identifier.
- #define PAL_NOLINE 0U

  Value identifying an invalid line.

Typedefs

- typedef void(∗ palcallback_t) (void)

  Type of a PAL event callback.
- typedef uint32_t ioportmask_t

  Digital I/O port sized unsigned type.
- typedef uint32_t iomode_t

  Digital I/O modes.
- typedef uint32_t ioline_t

  Type of an I/O line.
- typedef uint32_t ioportid_t

  Port Identifier.

Data Structures

- struct IOBus

  I/O bus descriptor.
- struct PALConfig

  Generic I/O ports static initializer.

Functions

- ioportmask_t palReadBus (IOBus ∗bus)

  Read from an I/O bus.
- void palWriteBus (IOBus ∗bus, ioportmask_t bits)

  Write to an I/O bus.
- void palSetBusMode (IOBus ∗bus, iomode_t mode)

  Programs a bus with the specified mode.
- void _pal_lld_init (const PALConfig ∗config)

  STM32 I/O ports configuration.
- void _pal_lld_setgroupmode (ioportid_t port, ioportmask_t mask, iomode_t mode)

  Pads mode setup.

7.29.3 Macro Definition Documentation

7.29.3.1 #define PAL_MODE_RESET 0U

After reset state.

The state itself is not specified and is architecture dependent, it is guaranteed to be equal to the after-reset state. It is usually an input state.

7.29.3.2 #define PAL_MODE_UNCONNECTED 1U

Safe state for **unconnected** pads.

The state itself is not specified and is architecture dependent, it may be mapped on `PAL_MODE_INPUT_PULLUP`, `PAL_MODE_INPUT_PULLDOWN` or `PAL_MODE_OUTPUT_PUSHPULL` for example.

7.29.3.3 #define PAL_MODE_INPUT 2U

Regular input high-Z pad.

7.29.3.4 #define PAL_MODE_INPUT_PULLUP 3U

Input pad with weak pull up resistor.

7.29.3.5 #define PAL_MODE_INPUT_PULLDOWN 4U

Input pad with weak pull down resistor.

7.29.3.6 #define PAL_MODE_INPUT_ANALOG 5U

Analog input mode.

7.29.3.7 #define PAL_MODE_OUTPUT_PUSHPULL 6U

Push-pull output pad.

7.29.3.8 #define PAL_MODE_OUTPUT_OPENDRAIN 7U

Open-drain output pad.

7.29.3.9 #define PAL_LOW 0U

Logical low state.

7.29.3.10 #define PAL_HIGH 1U

Logical high state.

7.29.3.11 #define PAL_EVENT_MODE_EDGES_MASK 3U

Mask of edges field.

7.29.3.12 #define PAL_EVENT_MODE_DISABLED 0U

Channel disabled.

7.29.3.13 #define PAL_EVENT_MODE_RISING_EDGE 1U

Rising edge callback.

7.29.3.14 #define PAL_EVENT_MODE_FALLING_EDGE 2U

Falling edge callback.

7.29.3.15 #define PAL_EVENT_MODE_BOTH_EDGES 3U

Both edges callback.

7.29.3.16 #define PAL_PORT_BIT(*n*) ((ioportmask_t)(1U << (n)))

Port bit helper macro.

This macro calculates the mask of a bit within a port.

Parameters

`in`	*n*	bit position within the port

Returns

The bit mask.

7.29.3.17 #define PAL_GROUP_MASK(*width*) ((ioportmask_t)(1U << (width)) - 1U)

Bits group mask helper.

This macro calculates the mask of a bits group.

Parameters

`in`	*width*	group width

Returns

The group mask.

7.29.3.18 #define _IOBUS_DATA(*name,* *port,* *width,* *offset*) {port, PAL_GROUP_MASK(width), offset}

Data part of a static I/O bus initializer.

This macro should be used when statically initializing an I/O bus that is part of a bigger structure.

Parameters

`in`	*name*	name of the IOBus variable
`in`	*port*	I/O port descriptor
`in`	*width*	bus width in bits
`in`	*offset*	bus bit offset within the port

7.29.3.19 #define IOBUS_DECL(*name,* *port,* *width,* *offset*) IOBus name = _IOBUS_DATA(name, port, width, offset)

Static I/O bus initializer.

Parameters

in	*name*	name of the IOBus variable
in	*port*	I/O port descriptor
in	*width*	bus width in bits
in	*offset*	bus bit offset within the port

7.29.3.20 #define palInit(*config*) pal_lld_init(config)

PAL subsystem initialization.

Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Parameters

in	*config*	pointer to an architecture specific configuration structure. This structure is defined in the low level driver header.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.29.3.21 #define palReadPort(*port*) ((void)(port), 0U)

Reads the physical I/O port states.

Note

The function can be called from any context.

Parameters

in	*port*	port identifier

Returns

The port logic states.

Function Class:

Special function, this function has special requirements see the notes.

7.29.3.22 #define palReadLatch(*port*) ((void)(port), 0U)

Reads the output latch.

The purpose of this function is to read back the latched output value.

Note

The function can be called from any context.

Parameters

`in`	*port*	port identifier

Returns

The latched logic states.

Function Class:

Special function, this function has special requirements see the notes.

7.29.3.23 #define palWritePort(*port*, *bits*) ((void)(port), (void)(bits))

Writes a bits mask on a I/O port.

Note

The function can be called from any context.

Parameters

`in`	*port*	port identifier
`in`	*bits*	bits to be written on the specified port

Function Class:

Special function, this function has special requirements see the notes.

7.29.3.24 #define palSetPort(*port*, *bits*) **palWritePort**(port, **palReadLatch**(port) \| (bits))

Sets a bits mask on a I/O port.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The function can be called from any context.

Parameters

`in`	*port*	port identifier
`in`	*bits*	bits to be ORed on the specified port

Function Class:

Special function, this function has special requirements see the notes.

7.29.3.25 #define palClearPort(*port, bits*) **palWritePort**(port, **palReadLatch**(port) & $\sim$(bits))

Clears a bits mask on a I/O port.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The function can be called from any context.

Parameters

`in`	*port*	port identifier
`in`	*bits*	bits to be cleared on the specified port

Function Class:

Special function, this function has special requirements see the notes.

7.29.3.26 #define palTogglePort(*port, bits*) **palWritePort**(port, **palReadLatch**(port) ^ (bits))

Toggles a bits mask on a I/O port.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The function can be called from any context.

Parameters

`in`	*port*	port identifier
`in`	*bits*	bits to be XORed on the specified port

Function Class:

Special function, this function has special requirements see the notes.

7.29.3.27 #define palReadGroup(*port, mask, offset*) ((**palReadPort**(port) $>>$ (offset)) & (mask))

Reads a group of bits.

Note

The function can be called from any context.

Parameters

`in`	*port*	port identifier
`in`	*mask*	group mask, a logic AND is performed on the input data
`in`	*offset*	group bit offset within the port

Returns

The group logic states.

Function Class:

Special function, this function has special requirements see the notes.

7.29.3.28 #define palWriteGroup(*port*, *mask*, *offset*, *bits*)

Value:

```
palWritePort(port, (palReadLatch(port) & ~((mask) << (offset))) |          \
                      (((bits) & (mask)) << (offset)))
```

Writes a group of bits.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The function can be called from any context.

Parameters

in	*port*	port identifier
in	*mask*	group mask, a logic AND is performed on the output data
in	*offset*	group bit offset within the port
in	*bits*	bits to be written. Values exceeding the group width are masked.

Function Class:

Special function, this function has special requirements see the notes.

7.29.3.29 #define palSetGroupMode(*port*, *mask*, *offset*, *mode*)

Pads group mode setup.

This function programs a pads group belonging to the same port with the specified mode.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
Programming an unknown or unsupported mode is silently ignored.
The function can be called from any context.

Parameters

in	*port*	port identifier
in	*mask*	group mask
in	*offset*	group bit offset within the port
in	*mode*	group mode

Function Class:

Special function, this function has special requirements see the notes.

7.29.3.30 #define palReadPad(*port*, *pad*) ((palReadPort(port) >> (pad)) & 1U)

Reads an input pad logic state.

Note

The function can be called from any context.

Parameters

in	*port*	port identifier
in	*pad*	pad number within the port

Returns

The logic state.

Return values

PAL_LOW	low logic state.
PAL_HIGH	high logic state.

Function Class:

Special function, this function has special requirements see the notes.

7.29.3.31 #define palWritePad(*port*, *pad*, *bit*)

Value:

```
palWritePort(port, (palReadLatch(port) & ~PAL_PORT_BIT(pad)) |
          \
                      (((bit) & 1U) << pad))
```

Writes a logic state on an output pad.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The function can be called from any context.

Parameters

in	*port*	port identifier
in	*pad*	pad number within the port
in	*bit*	logic value, the value must be `PAL_LOW` or `PAL_HIGH`

Function Class:

Special function, this function has special requirements see the notes.

7.29.3.32 #define palSetPad(*port*, *pad*) **palSetPort**(port, **PAL_PORT_BIT**(pad))

Sets a pad logic state to `PAL_HIGH`.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The function can be called from any context.

Parameters

`in`	*port*	port identifier
`in`	*pad*	pad number within the port

Function Class:

Special function, this function has special requirements see the notes.

7.29.3.33 #define palClearPad(*port*, *pad*) **palClearPort**(port, **PAL_PORT_BIT**(pad))

Clears a pad logic state to `PAL_LOW`.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The function can be called from any context.

Parameters

`in`	*port*	port identifier
`in`	*pad*	pad number within the port

Function Class:

Special function, this function has special requirements see the notes.

7.29.3.34 #define palTogglePad(*port*, *pad*) **palTogglePort**(port, **PAL_PORT_BIT**(pad))

Toggles a pad logic state.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The function can be called from any context.

Parameters

`in`	*port*	port identifier
`in`	*pad*	pad number within the port

Function Class:

Special function, this function has special requirements see the notes.

7.29.3.35 #define palSetPadMode(*port,* *pad,* *mode*) palSetGroupMode(port, PAL_PORT_BIT(pad), 0U, mode)

Pad mode setup.

This function programs a pad with the specified mode.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
Programming an unknown or unsupported mode is silently ignored.
The function can be called from any context.

Parameters

`in`	*port*	port identifier
`in`	*pad*	pad number within the port
`in`	*mode*	pad mode

Function Class:

Special function, this function has special requirements see the notes.

7.29.3.36 #define palPadEnableEventI(*port,* *pad,* *mode,* *callback*)

Pad event enable.

This function programs an event callback in the specified mode.

Note

Programming an unknown or unsupported mode is silently ignored.

Parameters

`in`	*port*	port identifier
`in`	*pad*	pad number within the port
`in`	*mode*	pad event mode
`in`	*callback*	event callback function

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.29.3.37 #define palPadDisableEventI(*port,* *pad*)

Pad event disable.

This function disables previously programmed event callbacks.

Parameters

in	*port*	port identifier
in	*pad*	pad number within the port

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.29.3.38 #define palReadLine(*line*) palReadPad(PAL_PORT(line), PAL_PAD(line))

Reads an input line logic state.

Note

The function can be called from any context.

Parameters

in	*line*	line identifier

Returns

The logic state.

Return values

PAL_LOW	low logic state.
PAL_HIGH	high logic state.

Function Class:

Special function, this function has special requirements see the notes.

7.29.3.39 #define palWriteLine(*line,* *bit*) palWritePad(PAL_PORT(line), PAL_PAD(line), bit)

Writes a logic state on an output line.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The function can be called from any context.

Parameters

in	*line*	line identifier
in	*bit*	logic value, the value must be `PAL_LOW` or `PAL_HIGH`

Function Class:

Special function, this function has special requirements see the notes.

7.29.3.40 #define palSetLine(*line*) palSetPad(PAL_PORT(line), PAL_PAD(line))

Sets a line logic state to `PAL_HIGH`.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The function can be called from any context.

Parameters

in	*line*	line identifier

Function Class:

Special function, this function has special requirements see the notes.

7.29.3.41 #define palClearLine(*line*) palClearPad(PAL_PORT(line), PAL_PAD(line))

Clears a line logic state to `PAL_LOW`.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The function can be called from any context.

Parameters

in	*line*	line identifier

Function Class:

Special function, this function has special requirements see the notes.

7.29.3.42 #define palToggleLine(*line*) **palTogglePad**(**PAL_PORT**(line), **PAL_PAD**(line))

Toggles a line logic state.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The function can be called from any context.

Parameters

in	*line*	line identifier

Function Class:

Special function, this function has special requirements see the notes.

7.29.3.43 #define palSetLineMode(*line, mode*) **palSetPadMode**(**PAL_PORT**(line), **PAL_PAD**(line), mode)

Line mode setup.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The function can be called from any context.

Parameters

in	*line*	line identifier
in	*mode*	pad mode

Function Class:

Special function, this function has special requirements see the notes.

7.29.3.44 #define palLineEnableEventI(*line, mode, callback*) **palPadEnableEventI**(**PAL_PORT**(line), **PAL_PAD**(line), mode, callback)

Line event enable.

Parameters

in	*line*	line identifier
in	*mode*	line event mode
in	*callback*	event callback function

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.29.3.45 #define palLineDisableEventI(*line*) **palPadDisableEventI**(**PAL_PORT**(line), **PAL_PAD**(line))

Line event disable.

Parameters

`in`	*line*	line identifier

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.29.3.46 #define PAL_IOPORTS_WIDTH 16U

Width, in bits, of an I/O port.

7.29.3.47 #define PAL_WHOLE_PORT ((**ioportmask_t**)0xFFFFU)

Whole port mask.

This macro specifies all the valid bits into a port.

7.29.3.48 #define PAL_LINE(*port, pad*) ((**ioline_t**)((uint32_t)(port)) | ((uint32_t)(pad)))

Forms a line identifier.

A port/pad pair are encoded into an `ioline_t` type. The encoding of this type is platform-dependent.

7.29.3.49 #define PAL_PORT(*line*) ((stm32_gpio_t *)(((uint32_t)(line)) & 0xFFFFFFF0U))

Decodes a port identifier from a line identifier.

7.29.3.50 #define PAL_PAD(*line*) ((uint32_t)((uint32_t)(line) & 0x0000000FU))

Decodes a pad identifier from a line identifier.

7.29.3.51 #define PAL_NOLINE 0U

Value identifying an invalid line.

7.29.3.52 #define IOPORT1 0

First I/O port identifier.

Low level drivers can define multiple ports, it is suggested to use this naming convention.

7.29.3.53 #define pal_lld_init(*config*) **_pal_lld_init**(config)

Low level PAL subsystem initialization.

Parameters

in	*config*	architecture-dependent ports configuration

Function Class:

Not an API, this function is for internal use only.

7.29.3.54 #define pal_lld_readport(*port*) 0U

Reads the physical I/O port states.

Parameters

in	*port*	port identifier

Returns

The port bits.

Function Class:

Not an API, this function is for internal use only.

7.29.3.55 #define pal_lld_readlatch(*port*) 0U

Reads the output latch.

The purpose of this function is to read back the latched output value.

Parameters

in	*port*	port identifier

Returns

The latched logical states.

Function Class:

Not an API, this function is for internal use only.

7.29.3.56 #define pal_lld_writeport(*port,* *bits*)

Value:

```
do {                                                                      \
    (void)port;                                                             \
    (void)bits;                                                             \
  } while (false)
```

Writes a bits mask on a I/O port.

Parameters

in	*port*	port identifier
in	*bits*	bits to be written on the specified port

Function Class:

Not an API, this function is for internal use only.

7.29.3.57 #define pal_lld_setport(*port,* *bits*)

Value:

```
do {                                                                    \
    (void)port;                                                           \
    (void)bits;                                                           \
  } while (false)
```

Sets a bits mask on a I/O port.

Note

The PAL Driver provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

in	*port*	port identifier
in	*bits*	bits to be ORed on the specified port

Function Class:

Not an API, this function is for internal use only.

7.29.3.58 #define pal_lld_clearport(*port,* *bits*)

Value:

```
do {                                                                    \
    (void)port;                                                           \
    (void)bits;                                                           \
  } while (false)
```

Clears a bits mask on a I/O port.

Note

The PAL Driver provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

in	*port*	port identifier
in	*bits*	bits to be cleared on the specified port

Function Class:

Not an API, this function is for internal use only.

7.29.3.59 #define pal_lld_toggleport(*port*, *bits*)

Value:

```
do {                                                                    \
    (void)port;                                                           \
    (void)bits;                                                           \
  } while (false)
```

Toggles a bits mask on a I/O port.

Note

The PAL Driver provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

in	*port*	port identifier
in	*bits*	bits to be XORed on the specified port

Function Class:

Not an API, this function is for internal use only.

7.29.3.60 #define pal_lld_readgroup(*port*, *mask*, *offset*) 0U

Reads a group of bits.

Note

The PAL Driver provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

in	*port*	port identifier
in	*mask*	group mask
in	*offset*	group bit offset within the port

Returns

The group logical states.

Function Class:

Not an API, this function is for internal use only.

7.29.3.61 #define pal_lld_writegroup(*port,* *mask,* *offset,* *bits*)

Value:

```
do {                                                                    \
    (void)port;                                                          \
    (void)mask;                                                          \
    (void)offset;                                                        \
    (void)bits;                                                          \
  } while (false)
```

Writes a group of bits.

Note

The PAL Driver provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

in	*port*	port identifier
in	*mask*	group mask
in	*offset*	group bit offset within the port
in	*bits*	bits to be written. Values exceeding the group width are masked.

Function Class:

Not an API, this function is for internal use only.

7.29.3.62 #define pal_lld_setgroupmode(*port,* *mask,* *offset,* *mode*) **_pal_lld_setgroupmode**(port, mask $<<$ offset, mode)

Pads group mode setup.

This function programs a pads group belonging to the same port with the specified mode.

Note

Programming an unknown or unsupported mode is silently ignored.

Parameters

in	*port*	port identifier
in	*mask*	group mask
in	*offset*	group bit offset within the port
in	*mode*	group mode

Function Class:

Not an API, this function is for internal use only.

7.29.3.63 #define pal_lld_readpad(*port,* *pad*) PAL_LOW

Reads a logical state from an I/O pad.

Note

The PAL Driver provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

in	*port*	port identifier
in	*pad*	pad number within the port

Returns

The logical state.

Return values

PAL_LOW	low logical state.
PAL_HIGH	high logical state.

Function Class:

Not an API, this function is for internal use only.

7.29.3.64 #define pal_lld_writepad(*port, pad, bit*)

Value:

```
do {                                                                \
    (void)port;                                                       \
    (void)pad;                                                        \
    (void)bit;                                                        \
  } while (false)
```

Writes a logical state on an output pad.

Note

This function is not meant to be invoked directly by the application code.
The PAL Driver provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

in	*port*	port identifier
in	*pad*	pad number within the port
in	*bit*	logical value, the value must be `PAL_LOW` or `PAL_HIGH`

Function Class:

Not an API, this function is for internal use only.

7.29.3.65 #define pal_lld_setpad(*port,* *pad*)

Value:

```
do {                                                                    \
    (void)port;                                                         \
    (void)pad;                                                          \
  } while (false)
```

Sets a pad logical state to `PAL_HIGH`.

Note

The PAL Driver provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

`in`	*port*	port identifier
`in`	*pad*	pad number within the port

Function Class:

Not an API, this function is for internal use only.

7.29.3.66 #define pal_lld_clearpad(*port,* *pad*)

Value:

```
do {                                                                    \
    (void)port;                                                         \
    (void)pad;                                                          \
  } while (false)
```

Clears a pad logical state to `PAL_LOW`.

Note

The PAL Driver provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

`in`	*port*	port identifier
`in`	*pad*	pad number within the port

Function Class:

Not an API, this function is for internal use only.

7.29.3.67 #define pal_lld_togglepad(*port,* *pad*)

Value:

```
do {                                                                    \
    (void)port;                                                           \
    (void)pad;                                                            \
  } while (false)
```

Toggles a pad logical state.

Note

The PAL Driver provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.

Parameters

in	*port*	port identifier
in	*pad*	pad number within the port

Function Class:

Not an API, this function is for internal use only.

7.29.3.68 #define pal_lld_setpadmode(*port,* *pad,* *mode*)

Value:

```
do {                                                                    \
    (void)port;                                                           \
    (void)pad;                                                            \
    (void)mode;                                                           \
  } while (false)
```

Pad mode setup.

This function programs a pad with the specified mode.

Note

The PAL Driver provides a default software implementation of this functionality, implement this function if can optimize it by using special hardware functionalities or special coding.
Programming an unknown or unsupported mode is silently ignored.

Parameters

in	*port*	port identifier
in	*pad*	pad number within the port
in	*mode*	pad mode

Function Class:

Not an API, this function is for internal use only.

7.29.4 Typedef Documentation

7.29.4.1 typedef void(∗ palcallback_t) (void)

Type of a PAL event callback.

7.29.4.2 typedef uint32_t ioportmask_t

Digital I/O port sized unsigned type.

7.29.4.3 typedef uint32_t iomode_t

Digital I/O modes.

7.29.4.4 typedef uint32_t ioline_t

Type of an I/O line.

7.29.4.5 typedef uint32_t ioportid_t

Port Identifier.

This type can be a scalar or some kind of pointer, do not make any assumption about it, use the provided macros when populating variables of this type.

7.29.5 Function Documentation

7.29.5.1 ioportmask_t palReadBus (IOBus * *bus*)

Read from an I/O bus.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The function internally uses the `palReadGroup()` macro. The use of this function is preferred when you value code size, readability and error checking over speed.
The function can be called from any context.

Parameters

`in`	*bus*	the I/O bus, pointer to a `IOBus` structure

Returns

The bus logical states.

Function Class:

Special function, this function has special requirements see the notes.

7.29.5.2 void palWriteBus (IOBus * *bus*, ioportmask_t *bits*)

Write to an I/O bus.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.
The function can be called from any context.

Parameters

`in`	*bus*	the I/O bus, pointer to a `IOBus` structure
`in`	*bits*	the bits to be written on the I/O bus. Values exceeding the bus width are masked so most significant bits are lost.

Function Class:

Special function, this function has special requirements see the notes.

7.29.5.3 void palSetBusMode (IOBus * *bus*, iomode_t *mode*)

Programs a bus with the specified mode.

Note

The operation is not guaranteed to be atomic on all the architectures, for atomicity and/or portability reasons you may need to enclose port I/O operations between `osalSysLock()` and `osalSysUnlock()`.
The default implementation is non atomic and not necessarily optimal. Low level drivers may optimize the function by using specific hardware or coding.
The function can be called from any context.

Parameters

`in`	*bus*	the I/O bus, pointer to a `IOBus` structure
`in`	*mode*	the mode

Function Class:

Special function, this function has special requirements see the notes.

7.29.5.4 void _pal_lld_init (const PALConfig * *config*)

STM32 I/O ports configuration.

Ports A-D(E, F, G, H) clocks enabled.

Parameters

`in`	*config*	the STM32 ports configuration

Function Class:

Not an API, this function is for internal use only.

7.29.5.5 void _pal_lld_setgroupmode (ioportid_t *port,* ioportmask_t *mask,* iomode_t *mode*)

Pads mode setup.

This function programs a pads group belonging to the same port with the specified mode.

Parameters

`in`	*port*	the port identifier
`in`	*mask*	the group mask
`in`	*mode*	the mode

Function Class:

Not an API, this function is for internal use only.

7.30 PWM Driver

Generic PWM Driver.

7.30.1 Detailed Description

Generic PWM Driver.

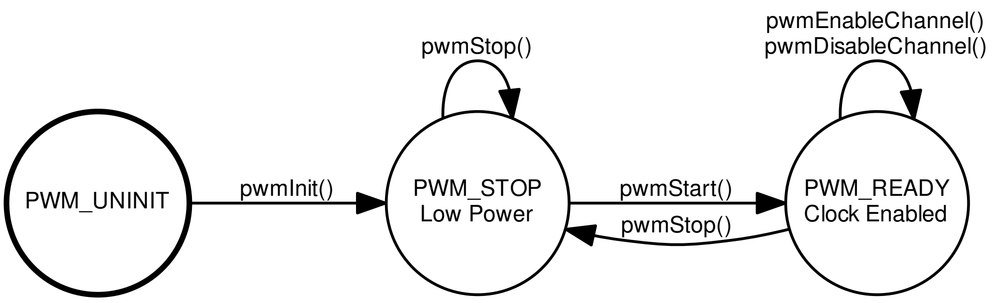
This module implements a generic PWM (Pulse Width Modulation) driver.

Precondition

In order to use the PWM driver the `HAL_USE_PWM` option must be enabled in `halconf.h`.

7.30.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



7.30.3 PWM Operations.

This driver abstracts a generic PWM timer composed of:

- A clock prescaler.
- A main up counter.
- A comparator register that resets the main counter to zero when the limit is reached. An optional callback can be generated when this happens.
- An array of `PWM_CHANNELS` PWM channels, each channel has an output, a comparator and is able to invoke an optional callback when a comparator match with the main counter happens.

A PWM channel output can be in two different states:

- **IDLE**, when the channel is disabled or after a match occurred.
- **ACTIVE**, when the channel is enabled and a match didn't occur yet in the current PWM cycle.

Note that the two states can be associated to both logical zero or one in the `PWMChannelConfig` structure.

Macros

- #define PWM_CHANNELS 4

  Number of PWM channels per PWM driver.
- #define pwm_lld_change_period(pwmp, period)

  Changes the period the PWM peripheral.

PWM output mode macros

- #define PWM_OUTPUT_MASK 0x0FU

  Standard output modes mask.
- #define PWM_OUTPUT_DISABLED 0x00U

  Output not driven, callback only.
- #define PWM_OUTPUT_ACTIVE_HIGH 0x01U

  Positive PWM logic, active is logic level one.
- #define PWM_OUTPUT_ACTIVE_LOW 0x02U

  Inverse PWM logic, active is logic level zero.

PWM duty cycle conversion

- #define PWM_FRACTION_TO_WIDTH(pwmp, denominator, numerator)

  Converts from fraction to pulse width.
- #define PWM_DEGREES_TO_WIDTH(pwmp, degrees) PWM_FRACTION_TO_WIDTH(pwmp, 36000, degrees)

  Converts from degrees to pulse width.
- #define PWM_PERCENTAGE_TO_WIDTH(pwmp, percentage) PWM_FRACTION_TO_WIDTH(pwmp, 10000, percentage)

  Converts from percentage to pulse width.

Macro Functions

- #define pwmChangePeriodI(pwmp, value)

  Changes the period the PWM peripheral.
- #define pwmEnableChannelI(pwmp, channel, width)

  Enables a PWM channel.
- #define pwmDisableChannelI(pwmp, channel)

  Disables a PWM channel.
- #define pwmIsChannelEnabledI(pwmp, channel) (((pwmp)->enabled & ((pwmchnmsk_t)1U << (pwmchnmsk↩ _t)(channel))) != 0U)

  Returns a PWM channel status.
- #define pwmEnablePeriodicNotificationI(pwmp) pwm_lld_enable_periodic_notification(pwmp)

  Enables the periodic activation edge notification.
- #define pwmDisablePeriodicNotificationI(pwmp) pwm_lld_disable_periodic_notification(pwmp)

  Disables the periodic activation edge notification.
- #define pwmEnableChannelNotificationI(pwmp, channel) pwm_lld_enable_channel_notification(pwmp, channel)

  Enables a channel de-activation edge notification.
- #define pwmDisableChannelNotificationI(pwmp, channel) pwm_lld_disable_channel_notification(pwmp, channel)

  Disables a channel de-activation edge notification.

PLATFORM configuration options

- #define PLATFORM_PWM_USE_PWM1 FALSE

  PWMD1 driver enable switch.

Typedefs

- typedef struct PWMDriver PWMDriver

  Type of a structure representing a PWM driver.
- typedef void(∗ pwmcallback_t) (PWMDriver ∗pwmp)

  Type of a PWM notification callback.
- typedef uint32_t pwmmode_t

  Type of a PWM mode.
- typedef uint8_t pwmchannel_t

  Type of a PWM channel.
- typedef uint32_t pwmchnmsk_t

  Type of a channels mask.
- typedef uint32_t pwmcnt_t

  Type of a PWM counter.

Data Structures

- struct PWMChannelConfig

  Type of a PWM driver channel configuration structure.
- struct PWMConfig

  Type of a PWM driver configuration structure.
- struct PWMDriver

  Structure representing a PWM driver.

Functions

- void pwmInit (void)

  PWM Driver initialization.
- void pwmObjectInit (PWMDriver ∗pwmp)

  Initializes the standard part of a `PWMDriver` structure.
- void pwmStart (PWMDriver ∗pwmp, const PWMConfig ∗config)

  Configures and activates the PWM peripheral.
- void pwmStop (PWMDriver ∗pwmp)

  Deactivates the PWM peripheral.
- void pwmChangePeriod (PWMDriver ∗pwmp, pwmcnt_t period)

  Changes the period the PWM peripheral.
- void pwmEnableChannel (PWMDriver ∗pwmp, pwmchannel_t channel, pwmcnt_t width)

  Enables a PWM channel.
- void pwmDisableChannel (PWMDriver ∗pwmp, pwmchannel_t channel)

  Disables a PWM channel and its notification.
- void pwmEnablePeriodicNotification (PWMDriver ∗pwmp)

  Enables the periodic activation edge notification.
- void pwmDisablePeriodicNotification (PWMDriver ∗pwmp)

  Disables the periodic activation edge notification.
- void pwmEnableChannelNotification (PWMDriver ∗pwmp, pwmchannel_t channel)

Enables a channel de-activation edge notification.

- void pwmDisableChannelNotification (PWMDriver ∗pwmp, pwmchannel_t channel)

  Disables a channel de-activation edge notification.
- void pwm_lld_init (void)

  Low level PWM driver initialization.
- void pwm_lld_start (PWMDriver ∗pwmp)

  Configures and activates the PWM peripheral.
- void pwm_lld_stop (PWMDriver ∗pwmp)

  Deactivates the PWM peripheral.
- void pwm_lld_enable_channel (PWMDriver ∗pwmp, pwmchannel_t channel, pwmcnt_t width)

  Enables a PWM channel.
- void pwm_lld_disable_channel (PWMDriver ∗pwmp, pwmchannel_t channel)

  Disables a PWM channel and its notification.
- void pwm_lld_enable_periodic_notification (PWMDriver ∗pwmp)

  Enables the periodic activation edge notification.
- void pwm_lld_disable_periodic_notification (PWMDriver ∗pwmp)

  Disables the periodic activation edge notification.
- void pwm_lld_enable_channel_notification (PWMDriver ∗pwmp, pwmchannel_t channel)

  Enables a channel de-activation edge notification.
- void pwm_lld_disable_channel_notification (PWMDriver ∗pwmp, pwmchannel_t channel)

  Disables a channel de-activation edge notification.

Enumerations

Variables

- PWMDriver PWMD1

  PWMD1 driver identifier.

7.30.4 Macro Definition Documentation

7.30.4.1 #define PWM_OUTPUT_MASK 0x0FU

Standard output modes mask.

7.30.4.2 #define PWM_OUTPUT_DISABLED 0x00U

Output not driven, callback only.

7.30.4.3 #define PWM_OUTPUT_ACTIVE_HIGH 0x01U

Positive PWM logic, active is logic level one.

7.30.4.4 #define PWM_OUTPUT_ACTIVE_LOW 0x02U

Inverse PWM logic, active is logic level zero.

7.30.4.5 #define PWM_FRACTION_TO_WIDTH(*pwmp,* *denominator,* *numerator*)

Value:

```
((pwmcnt_t)((((pwmcnt_t)(pwmp)->period) *                          \
                (pwmcnt_t)(numerator)) / (pwmcnt_t)(denominator)))
```

Converts from fraction to pulse width.

Note

Be careful with rounding errors, this is integer math not magic. You can specify tenths of thousandth but make sure you have the proper hardware resolution by carefully choosing the clock source and prescaler settings, see `PWM_COMPUTE_PSC`.

Parameters

in	*pwmp*	pointer to a `PWMDriver` object
in	*denominator*	denominator of the fraction
in	*numerator*	numerator of the fraction

Returns

The pulse width to be passed to `pwmEnableChannel()`.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.30.4.6 #define PWM_DEGREES_TO_WIDTH(*pwmp,* *degrees*) PWM_FRACTION_TO_WIDTH(pwmp, 36000, degrees)

Converts from degrees to pulse width.

Note

Be careful with rounding errors, this is integer math not magic. You can specify hundredths of degrees but make sure you have the proper hardware resolution by carefully choosing the clock source and prescaler settings, see `PWM_COMPUTE_PSC`.

Parameters

in	*pwmp*	pointer to a `PWMDriver` object
in	*degrees*	degrees as an integer between 0 and 36000

Returns

The pulse width to be passed to `pwmEnableChannel()`.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.30.4.7 #define PWM_PERCENTAGE_TO_WIDTH(*pwmp,* *percentage*) PWM_FRACTION_TO_WIDTH(pwmp, 10000, percentage)

Converts from percentage to pulse width.

Note

Be careful with rounding errors, this is integer math not magic. You can specify tenths of thousandth but make sure you have the proper hardware resolution by carefully choosing the clock source and prescaler settings, see `PWM_COMPUTE_PSC`.

Parameters

in	*pwmp*	pointer to a `PWMDriver` object
in	*percentage*	percentage as an integer between 0 and 10000

Returns

The pulse width to be passed to `pwmEnableChannel()`.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.30.4.8 #define pwmChangePeriodI(*pwmp,* *value*)

Value:

```
{                                                   \
  (pwmp)->period = (value);                                                   \
  pwm_lld_change_period(pwmp, value);                                         \
}
```

Changes the period the PWM peripheral.

This function changes the period of a PWM unit that has already been activated using `pwmStart()`.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Postcondition

The PWM unit period is changed to the new value.

Note

If a period is specified that is shorter than the pulse width programmed in one of the channels then the behavior is not guaranteed.

Parameters

in	*pwmp*	pointer to a `PWMDriver` object
in	*value*	new cycle time in ticks

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.30.4.9 #define pwmEnableChannelI(*pwmp,* *channel,* *width*)

Value:

```
do {                              \
   (pwmp)->enabled |= ((pwmchnmsk_t)1U << (pwmchnmsk_t)(channel));
      \
  pwm_lld_enable_channel(pwmp, channel, width);                              \
} while (false)
```

Enables a PWM channel.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Postcondition

The channel is active using the specified configuration.

Note

Depending on the hardware implementation this function has effect starting on the next cycle (recommended implementation) or immediately (fallback implementation).

Parameters

`in`	*pwmp*	pointer to a `PWMDriver` object
`in`	*channel*	PWM channel identifier (0...channels-1)
`in`	*width*	PWM pulse width as clock pulses number

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.30.4.10 #define pwmDisableChannelI(*pwmp,* *channel*)

Value:

```
do {                                    \
   (pwmp)->enabled &= ~((pwmchnmsk_t)1U << (pwmchnmsk_t)(channel));
      \
  pwm_lld_disable_channel(pwmp, channel);                                    \
} while (false)
```

Disables a PWM channel.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Postcondition

The channel is disabled and its output line returned to the idle state.

Note

Depending on the hardware implementation this function has effect starting on the next cycle (recommended implementation) or immediately (fallback implementation).

Parameters

`in`	*pwmp*	pointer to a `PWMDriver` object
`in`	*channel*	PWM channel identifier (0...channels-1)

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.30.4.11 #define pwmIsChannelEnabledI(*pwmp,* *channel*) (((pwmp)->enabled & ((pwmchnmsk_t)1U << (pwmchnmsk_t)(channel))) != 0U)

Returns a PWM channel status.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Parameters

`in`	*pwmp*	pointer to a `PWMDriver` object
`in`	*channel*	PWM channel identifier (0...channels-1)

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.30.4.12 #define pwmEnablePeriodicNotificationI(*pwmp*) pwm_lld_enable_periodic_notification(pwmp)

Enables the periodic activation edge notification.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Note

If the notification is already enabled then the call has no effect.

Parameters

in	*pwmp*	pointer to a PWMDriver object

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.30.4.13 #define pwmDisablePeriodicNotificationI(*pwmp*) pwm_lld_disable_periodic_notification(pwmp)

Disables the periodic activation edge notification.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Note

If the notification is already disabled then the call has no effect.

Parameters

in	*pwmp*	pointer to a PWMDriver object

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.30.4.14 #define pwmEnableChannelNotificationI(*pwmp, channel*) pwm_lld_enable_channel_notification(pwmp, channel)

Enables a channel de-activation edge notification.

Precondition

The PWM unit must have been activated using `pwmStart()`.
The channel must have been activated using `pwmEnableChannel()`.

Note

If the notification is already enabled then the call has no effect.

Parameters

in	*pwmp*	pointer to a PWMDriver object
in	*channel*	PWM channel identifier (0...channels-1)

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.30.4.15 #define pwmDisableChannelNotificationI(*pwmp, channel*) pwm_lld_disable_channel_notification(pwmp, channel)

Disables a channel de-activation edge notification.

Precondition

The PWM unit must have been activated using `pwmStart()`.
The channel must have been activated using `pwmEnableChannel()`.

Note

If the notification is already disabled then the call has no effect.

Parameters

`in`	*pwmp*	pointer to a `PWMDriver` object
`in`	*channel*	PWM channel identifier (0...channels-1)

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.30.4.16 #define PWM_CHANNELS 4

Number of PWM channels per PWM driver.

7.30.4.17 #define PLATFORM_PWM_USE_PWM1 FALSE

PWMD1 driver enable switch.

If set to `TRUE` the support for PWM1 is included.

Note

The default is `FALSE`.

7.30.4.18 #define pwm_lld_change_period(*pwmp, period*)

Changes the period the PWM peripheral.

This function changes the period of a PWM unit that has already been activated using `pwmStart()`.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Postcondition

The PWM unit period is changed to the new value.

Note

The function has effect at the next cycle start.
If a period is specified that is shorter than the pulse width programmed in one of the channels then the behavior is not guaranteed.

Parameters

`in`	*pwmp*	pointer to a `PWMDriver` object
`in`	*period*	new cycle time in ticks

Function Class:

Not an API, this function is for internal use only.

7.30.5 Typedef Documentation

7.30.5.1 typedef struct PWMDriver PWMDriver

Type of a structure representing a PWM driver.

7.30.5.2 typedef void(∗ pwmcallback_t) (PWMDriver ∗pwmp)

Type of a PWM notification callback.

Parameters

`in`	*pwmp*	pointer to a `PWMDriver` object

7.30.5.3 typedef uint32_t pwmmode_t

Type of a PWM mode.

7.30.5.4 typedef uint8_t pwmchannel_t

Type of a PWM channel.

7.30.5.5 typedef uint32_t pwmchnmsk_t

Type of a channels mask.

7.30.5.6 typedef uint32_t pwmcnt_t

Type of a PWM counter.

7.30.6 Enumeration Type Documentation

7.30.6.1 enum **pwmstate_t**

Driver state machine possible states.

Enumerator

PWM_UNINIT Not initialized.

PWM_STOP Stopped.

PWM_READY Ready.

7.30.7 Function Documentation

7.30.7.1 void pwmInit (void)

PWM Driver initialization.

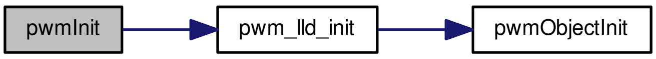
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



7.30.7.2 void pwmObjectInit (**PWMDriver** $*$ *pwmp*)

Initializes the standard part of a `PWMDriver` structure.

Parameters

`out`	*pwmp*	pointer to a `PWMDriver` object

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.30.7.3 void pwmStart (**PWMDriver** $*$ *pwmp,* const **PWMConfig** $*$ *config*)

Configures and activates the PWM peripheral.

Note

Starting a driver that is already in the `PWM_READY` state disables all the active channels.

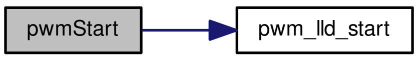
Parameters

in	*pwmp*	pointer to a `PWMDriver` object
in	*config*	pointer to a `PWMConfig` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.30.7.4 void pwmStop (**PWMDriver** ∗ *pwmp*)

Deactivates the PWM peripheral.

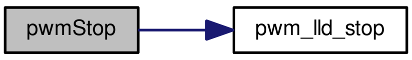
Parameters

in	*pwmp*	pointer to a `PWMDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.30.7.5 void pwmChangePeriod (**PWMDriver** ∗ *pwmp,* **pwmcnt_t** *period*)

Changes the period the PWM peripheral.

This function changes the period of a PWM unit that has already been activated using `pwmStart()`.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Postcondition

The PWM unit period is changed to the new value.

Note

If a period is specified that is shorter than the pulse width programmed in one of the channels then the behavior is not guaranteed.

Parameters

`in`	*pwmp*	pointer to a `PWMDriver` object
`in`	*period*	new cycle time in ticks

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.30.7.6 void pwmEnableChannel (**PWMDriver** ∗ *pwmp*, **pwmchannel_t** *channel*, **pwmcnt_t** *width*)

Enables a PWM channel.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Postcondition

The channel is active using the specified configuration.

Note

Depending on the hardware implementation this function has effect starting on the next cycle (recommended implementation) or immediately (fallback implementation).

Parameters

`in`	*pwmp*	pointer to a `PWMDriver` object
`in`	*channel*	PWM channel identifier (0...channels-1)
`in`	*width*	PWM pulse width as clock pulses number

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.30.7.7 void pwmDisableChannel (**PWMDriver** ∗ *pwmp*, **pwmchannel_t** *channel*)

Disables a PWM channel and its notification.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Postcondition

The channel is disabled and its output line returned to the idle state.

Note

Depending on the hardware implementation this function has effect starting on the next cycle (recommended implementation) or immediately (fallback implementation).

Parameters

`in`	*pwmp*	pointer to a `PWMDriver` object
`in`	*channel*	PWM channel identifier (0...channels-1)

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.30.7.8 void pwmEnablePeriodicNotification (**PWMDriver** ∗ ***pwmp***)

Enables the periodic activation edge notification.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Note

If the notification is already enabled then the call has no effect.

Parameters

`in`	*pwmp*	pointer to a `PWMDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.30.7.9 void pwmDisablePeriodicNotification (**PWMDriver** ∗ ***pwmp***)

Disables the periodic activation edge notification.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Note

If the notification is already disabled then the call has no effect.

Parameters

in	*pwmp*	pointer to a `PWMDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.30.7.10 void pwmEnableChannelNotification (PWMDriver ∗ *pwmp*, pwmchannel_t *channel*)

Enables a channel de-activation edge notification.

Precondition

The PWM unit must have been activated using `pwmStart()`.
The channel must have been activated using `pwmEnableChannel()`.

Note

If the notification is already enabled then the call has no effect.

Parameters

in	*pwmp*	pointer to a `PWMDriver` object
in	*channel*	PWM channel identifier (0...channels-1)

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.30.7.11 void pwmDisableChannelNotification (PWMDriver ∗ *pwmp*, pwmchannel_t *channel*)

Disables a channel de-activation edge notification.

Precondition

The PWM unit must have been activated using `pwmStart()`.
The channel must have been activated using `pwmEnableChannel()`.

Note

If the notification is already disabled then the call has no effect.

Parameters

in	*pwmp*	pointer to a `PWMDriver` object
in	*channel*	PWM channel identifier (0...channels-1)

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

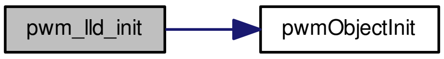
7.30.7.12 void pwm_lld_init (void)

Low level PWM driver initialization.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.30.7.13 void pwm_lld_start (PWMDriver ∗ *pwmp*)

Configures and activates the PWM peripheral.

Note

Starting a driver that is already in the `PWM_READY` state disables all the active channels.

Parameters

in	*pwmp*	pointer to a `PWMDriver` object

Function Class:

Not an API, this function is for internal use only.

7.30.7.14 void pwm_lld_stop (PWMDriver ∗ *pwmp*)

Deactivates the PWM peripheral.

Parameters

in	*pwmp*	pointer to a `PWMDriver` object

Function Class:

Not an API, this function is for internal use only.

7.30.7.15 void pwm_lld_enable_channel (PWMDriver ∗ *pwmp,* pwmchannel_t *channel,* pwmcnt_t *width*)

Enables a PWM channel.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Postcondition

The channel is active using the specified configuration.

Note

The function has effect at the next cycle start.
Channel notification is not enabled.

Parameters

`in`	*pwmp*	pointer to a `PWMDriver` object
`in`	*channel*	PWM channel identifier (0...channels-1)
`in`	*width*	PWM pulse width as clock pulses number

Function Class:

Not an API, this function is for internal use only.

7.30.7.16 void pwm_lld_disable_channel (**PWMDriver** ∗ *pwmp,* **pwmchannel_t** *channel*)

Disables a PWM channel and its notification.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Postcondition

The channel is disabled and its output line returned to the idle state.

Note

The function has effect at the next cycle start.

Parameters

`in`	*pwmp*	pointer to a `PWMDriver` object
`in`	*channel*	PWM channel identifier (0...channels-1)

Function Class:

Not an API, this function is for internal use only.

7.30.7.17 void pwm_lld_enable_periodic_notification (**PWMDriver** ∗ *pwmp*)

Enables the periodic activation edge notification.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Note

If the notification is already enabled then the call has no effect.

Parameters

in	*pwmp*	pointer to a `PWMDriver` object

Function Class:

Not an API, this function is for internal use only.

7.30.7.18 void pwm_lld_disable_periodic_notification (PWMDriver ∗ *pwmp*)

Disables the periodic activation edge notification.

Precondition

The PWM unit must have been activated using `pwmStart()`.

Note

If the notification is already disabled then the call has no effect.

Parameters

in	*pwmp*	pointer to a `PWMDriver` object

Function Class:

Not an API, this function is for internal use only.

7.30.7.19 void pwm_lld_enable_channel_notification (PWMDriver ∗ *pwmp*, pwmchannel_t *channel*)

Enables a channel de-activation edge notification.

Precondition

The PWM unit must have been activated using `pwmStart()`.
The channel must have been activated using `pwmEnableChannel()`.

Note

If the notification is already enabled then the call has no effect.

Parameters

in	*pwmp*	pointer to a `PWMDriver` object
in	*channel*	PWM channel identifier (0...channels-1)

Function Class:

Not an API, this function is for internal use only.

7.30.7.20 void pwm_lld_disable_channel_notification (**PWMDriver** ∗ *pwmp*, **pwmchannel_t** *channel*)

Disables a channel de-activation edge notification.

Precondition

The PWM unit must have been activated using `pwmStart()`.
The channel must have been activated using `pwmEnableChannel()`.

Note

If the notification is already disabled then the call has no effect.

Parameters

`in`	*pwmp*	pointer to a `PWMDriver` object
`in`	*channel*	PWM channel identifier (0...channels-1)

Function Class:

Not an API, this function is for internal use only.

7.30.8 Variable Documentation

7.30.8.1 PWMDriver PWMD1

PWMD1 driver identifier.

Note

The driver PWMD1 allocates the complex timer TIM1 when enabled.

7.31 QSPI Driver

Generic QSPI Driver.

7.31.1 Detailed Description

Generic QSPI Driver.

This module defines an abstract interface for a Quad SPI communication interface.

Precondition

In order to use the QSPI driver the `HAL_USE_QSPI` option must be enabled in `halconf.h`.

Transfer options

- #define **QSPI_CFG_CMD_MASK** (0xFFLU << 0LU)
- #define **QSPI_CFG_CMD**(n) ((n) << 0LU)
- #define **QSPI_CFG_CMD_MODE_MASK** (3LU << 8LU)
- #define **QSPI_CFG_CMD_MODE_NONE** (0LU << 8LU)
- #define **QSPI_CFG_CMD_MODE_ONE_LINE** (1LU << 8LU)
- #define **QSPI_CFG_CMD_MODE_TWO_LINES** (2LU << 8LU)
- #define **QSPI_CFG_CMD_MODE_FOUR_LINES** (3LU << 8LU)
- #define **QSPI_CFG_ADDR_MODE_MASK** (3LU << 10LU)
- #define **QSPI_CFG_ADDR_MODE_NONE** (0LU << 10LU)
- #define **QSPI_CFG_ADDR_MODE_ONE_LINE** (1LU << 10LU)
- #define **QSPI_CFG_ADDR_MODE_TWO_LINES** (2LU << 10LU)
- #define **QSPI_CFG_ADDR_MODE_FOUR_LINES** (3LU << 10LU)
- #define **QSPI_CFG_ADDR_SIZE_MASK** (3LU << 12LU)
- #define **QSPI_CFG_ADDR_SIZE_8** (0LU << 12LU)
- #define **QSPI_CFG_ADDR_SIZE_16** (1LU << 12LU)
- #define **QSPI_CFG_ADDR_SIZE_24** (2LU << 12LU)
- #define **QSPI_CFG_ADDR_SIZE_32** (3LU << 12LU)
- #define **QSPI_CFG_ALT_MODE_MASK** (3LU << 14LU)
- #define **QSPI_CFG_ALT_MODE_NONE** (0LU << 14LU)
- #define **QSPI_CFG_ALT_MODE_ONE_LINE** (1LU << 14LU)
- #define **QSPI_CFG_ALT_MODE_TWO_LINES** (2LU << 14LU)
- #define **QSPI_CFG_ALT_MODE_FOUR_LINES** (3LU << 14LU)
- #define **QSPI_CFG_ALT_SIZE_MASK** (3LU << 16LU)
- #define **QSPI_CFG_ALT_SIZE_8** (0LU << 16LU)
- #define **QSPI_CFG_ALT_SIZE_16** (1LU << 16LU)
- #define **QSPI_CFG_ALT_SIZE_24** (2LU << 16LU)
- #define **QSPI_CFG_ALT_SIZE_32** (3LU << 16LU)
- #define **QSPI_CFG_DUMMY_CYCLES_MASK** (0x1FLU << 18LU)
- #define **QSPI_CFG_DUMMY_CYCLES**(n) ((n) << 18LU)
- #define **QSPI_CFG_DATA_MODE_MASK** (3LU << 24LU)
- #define **QSPI_CFG_DATA_MODE_NONE** (0LU << 24LU)
- #define **QSPI_CFG_DATA_MODE_ONE_LINE** (1LU << 24LU)
- #define **QSPI_CFG_DATA_MODE_TWO_LINES** (2LU << 24LU)
- #define **QSPI_CFG_DATA_MODE_FOUR_LINES** (3LU << 24LU)
- #define **QSPI_CFG_SIOO** (1LU << 28LU)
- #define **QSPI_CFG_DDRM** (1LU << 31LU)

QSPI configuration options

- #define QSPI_USE_WAIT TRUE

  Enables synchronous APIs.
- #define QSPI_USE_MUTUAL_EXCLUSION TRUE

  Enables the `qspiAcquireBus()` and `qspiReleaseBus()` APIs.

Macro Functions

- #define qspiStartCommandI(qspip, cmdp)

  Sends a command without data phase.
- #define qspiStartSendI(qspip, cmdp, n, txbuf)

  Sends data over the QSPI bus.
- #define qspiStartReceiveI(qspip, cmdp, n, rxbuf)

  Receives data from the QSPI bus.
- #define qspiMapFlashI(qspip, cmdp, addrp) qspi_lld_map_flash(qspip, cmdp, addrp)

  Maps in memory space a QSPI flash device.
- #define qspiUnmapFlashI(qspip) qspi_lld_unmap_flash(qspip)

  Maps in memory space a QSPI flash device.

Low level driver helper macros

- #define _qspi_wakeup_isr(qspip)

  Wakes up the waiting thread.
- #define _qspi_isr_code(qspip)

  Common ISR code.

QSPI capabilities

- #define **QSPI_SUPPORTS_MEMMAP** TRUE

Configuration options

- #define PLATFORM_QSPI_USE_QSPI1 FALSE

  QSPID1 driver enable switch.

Typedefs

- typedef struct QSPIDriver QSPIDriver

  Type of a structure representing an QSPI driver.
- typedef void(∗ qspicallback_t) (QSPIDriver ∗qspip)

  Type of a QSPI notification callback.

Data Structures

- struct qspi_command_t

  Type of a QSPI command descriptor.
- struct QSPIConfig

  Driver configuration structure.
- struct QSPIDriver

  Structure representing an QSPI driver.

Functions

- void qspiInit (void)
  QSPI Driver initialization.
- void qspiObjectInit (QSPIDriver ∗qspip)
  Initializes the standard part of a `QSPIDriver` structure.
- void qspiStart (QSPIDriver ∗qspip, const QSPIConfig ∗config)
  Configures and activates the QSPI peripheral.
- void qspiStop (QSPIDriver ∗qspip)
  Deactivates the QSPI peripheral.
- void qspiStartCommand (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp)
  Sends a command without data phase.
- void qspiStartSend (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp, size_t n, const uint8_t ∗txbuf)
  Sends a command with data over the QSPI bus.
- void qspiStartReceive (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp, size_t n, uint8_t ∗rxbuf)
  Sends a command then receives data over the QSPI bus.
- void qspiCommand (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp)
  Sends a command without data phase.
- void qspiSend (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp, size_t n, const uint8_t ∗txbuf)
  Sends a command with data over the QSPI bus.
- void qspiReceive (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp, size_t n, uint8_t ∗rxbuf)
  Sends a command then receives data over the QSPI bus.
- void qspiMapFlash (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp, uint8_t ∗∗addrp)
  Maps in memory space a QSPI flash device.
- void qspiUnmapFlash (QSPIDriver ∗qspip)
  Maps in memory space a QSPI flash device.
- void qspiAcquireBus (QSPIDriver ∗qspip)
  Gains exclusive access to the QSPI bus.
- void qspiReleaseBus (QSPIDriver ∗qspip)
  Releases exclusive access to the QSPI bus.
- void qspi_lld_init (void)
  Low level QSPI driver initialization.
- void qspi_lld_start (QSPIDriver ∗qspip)
  Configures and activates the QSPI peripheral.
- void qspi_lld_stop (QSPIDriver ∗qspip)
  Deactivates the QSPI peripheral.
- void qspi_lld_command (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp)
  Sends a command without data phase.
- void qspi_lld_send (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp, size_t n, const uint8_t ∗txbuf)
  Sends a command with data over the QSPI bus.
- void qspi_lld_receive (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp, size_t n, uint8_t ∗rxbuf)
  Sends a command then receives data over the QSPI bus.
- void qspi_lld_map_flash (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp, uint8_t ∗∗addrp)
  Maps in memory space a QSPI flash device.
- void qspi_lld_unmap_flash (QSPIDriver ∗qspip)
  Maps in memory space a QSPI flash device.

Enumerations

Variables

- QSPIDriver QSPID1
  QSPID1 driver identifier.

7.31.2 Macro Definition Documentation

7.31.2.1 #define QSPI_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.31.2.2 #define QSPI_USE_MUTUAL_EXCLUSION TRUE

Enables the `qspiAcquireBus()` and `qspiReleaseBus()` APIs.

Note

Disabling this option saves both code and data space.

7.31.2.3 #define qspiStartCommandI(*qspip,* *cmdp*)

Value:

```
{                                                 \
  osalDbgAssert(((cmdp)->cfg & QSPI_CFG_DATA_MODE_MASK) ==                    \
                QSPI_CFG_DATA_MODE_NONE,                                     \
                "data mode specified");                                      \
  (qspip)->state = QSPI_ACTIVE;
      \
  qspi_lld_command(qspip, cmdp);                                             \
}
```

Sends a command without data phase.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	*qspip*	pointer to the `QSPIDriver` object
in	*cmdp*	pointer to the command descriptor

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.31.2.4 #define qspiStartSendI(*qspip,* *cmdp,* *n,* *txbuf*)

Value:

```
{                                       \
  osalDbgAssert(((cmdp)->cfg & QSPI_CFG_DATA_MODE_MASK) !=                    \
                QSPI_CFG_DATA_MODE_NONE,                                     \
                "data mode required");                                       \
  (qspip)->state = QSPI_ACTIVE;
      \
  qspi_lld_send(qspip, cmdp, n, txbuf);                                    \
}
```

Sends data over the QSPI bus.

This asynchronous function starts a transmit operation.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	*qspip*	pointer to the `QSPIDriver` object
in	*cmdp*	pointer to the command descriptor
in	*n*	number of bytes to send or zero if no data phase
in	*txbuf*	the pointer to the transmit buffer

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.31.2.5 #define qspiStartReceiveI(*qspip,* *cmdp,* *n,* *rxbuf*)

Value:

```
{                             \
  osalDbgAssert(((cmdp)->cfg & QSPI_CFG_DATA_MODE_MASK) !=                 \
                QSPI_CFG_DATA_MODE_NONE,                                   \
                "data mode required");                                     \
  (qspip)->state = QSPI_ACTIVE;
      \
  qspi_lld_receive(qspip, cmdp, n, rxbuf);                                 \
}
```

Receives data from the QSPI bus.

This asynchronous function starts a receive operation.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	*qspip*	pointer to the `QSPIDriver` object
in	*cmdp*	pointer to the command descriptor
in	*n*	number of bytes to receive or zero if no data phase
out	*rxbuf*	the pointer to the receive buffer

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.31.2.6 #define qspiMapFlashI(*qspip, cmdp, addrp*) **qspi_lld_map_flash**(qspip, cmdp, addrp)

Maps in memory space a QSPI flash device.

Precondition

The memory flash device must be initialized appropriately before mapping it in memory space.

Parameters

in	*qspip*	pointer to the QSPIDriver object
in	*cmdp*	pointer to the command descriptor
out	*addrp*	pointer to the memory start address of the mapped flash or NULL

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.31.2.7 #define qspiUnmapFlashI(*qspip*) **qspi_lld_unmap_flash**(qspip)

Maps in memory space a QSPI flash device.

Postcondition

The memory flash device must be re-initialized for normal commands exchange.

Parameters

in	*qspip*	pointer to the QSPIDriver object

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.31.2.8 #define _qspi_wakeup_isr(*qspip*)

Value:

```
{                                           \
  osalSysLockFromISR();                                                              \
  osalThreadResumeI(&(qspip)->thread, MSG_OK);                                       \
  osalSysUnlockFromISR();                                                            \
}
```

Wakes up the waiting thread.

Parameters

in	*qspip*	pointer to the QSPIDriver object

Function Class:

Not an API, this function is for internal use only.

7.31.2.9 #define _qspi_isr_code(*qspip*)

Value:

```
{                                                \
  if ((qspip)->config->end_cb) {                                  \
    (qspip)->state = QSPI_COMPLETE;                               \
    (qspip)->config->end_cb(qspip);                               \
    if ((qspip)->state == QSPI_COMPLETE)                          \
      (qspip)->state = QSPI_READY;                                \
  }                                                               \
  else                                                            \
    (qspip)->state = QSPI_READY;
      \
  _qspi_wakeup_isr(qspip);                                        \
}
```

Common ISR code.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

in	*qspip*	pointer to the QSPIDriver object

Function Class:

Not an API, this function is for internal use only.

7.31.2.10 #define PLATFORM_QSPI_USE_QSPI1 FALSE

QSPID1 driver enable switch.

If set to TRUE the support for QSPID1 is included.

Note

The default is FALSE.

7.31.3 Typedef Documentation

7.31.3.1 typedef struct QSPIDriver QSPIDriver

Type of a structure representing an QSPI driver.

7.31.3.2 typedef void(∗ qspicallback_t) (QSPIDriver ∗qspip)

Type of a QSPI notification callback.

Parameters

in	*qspip*	pointer to the QSPIDriver object triggering the callback

7.31.4 Enumeration Type Documentation

7.31.4.1 enum qspistate_t

Driver state machine possible states.

Enumerator

QSPI_UNINIT	Not initialized.
QSPI_STOP	Stopped.
QSPI_READY	Ready.
QSPI_ACTIVE	Exchanging data.
QSPI_COMPLETE	Asynchronous operation complete.
QSPI_MEMMAP	In memory mapped mode.

7.31.5 Function Documentation

7.31.5.1 void qspiInit (void)

QSPI Driver initialization.

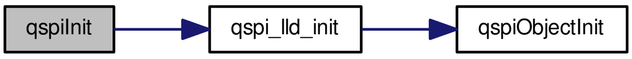
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



7.31.5.2 void qspiObjectInit (QSPIDriver ∗ *qspip*)

Initializes the standard part of a `QSPIDriver` structure.

Parameters

out	*qspip*	pointer to the `QSPIDriver` object

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.31.5.3 void qspiStart (**QSPIDriver** ∗ *qspip*, const **QSPIConfig** ∗ *config*)

Configures and activates the QSPI peripheral.

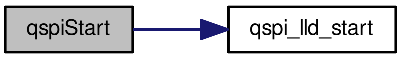
Parameters

in	*qspip*	pointer to the `QSPIDriver` object
in	*config*	pointer to the `QSPIConfig` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.31.5.4 void qspiStop (**QSPIDriver** ∗ *qspip*)

Deactivates the QSPI peripheral.

Note

Deactivating the peripheral also enforces a release of the slave select line.

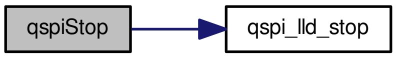
Parameters

in	*qspip*	pointer to the `QSPIDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.31.5.5 void qspiStartCommand (**QSPIDriver** ∗ *qspip,* const **qspi_command_t** ∗ *cmdp*)

Sends a command without data phase.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	*qspip*	pointer to the `QSPIDriver` object
in	*cmdp*	pointer to the command descriptor

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.31.5.6 void qspiStartSend (**QSPIDriver** ∗ *qspip,* const **qspi_command_t** ∗ *cmdp,* size_t *n,* const uint8_t ∗ *txbuf*)

Sends a command with data over the QSPI bus.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	*qspip*	pointer to the `QSPIDriver` object
in	*cmdp*	pointer to the command descriptor
in	*n*	number of bytes to send
in	*txbuf*	the pointer to the transmit buffer

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.31.5.7 void qspiStartReceive (**QSPIDriver** ∗ *qspip,* const **qspi_command_t** ∗ *cmdp,* size_t *n,* uint8_t ∗ *rxbuf*)

Sends a command then receives data over the QSPI bus.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	*qspip*	pointer to the `QSPIDriver` object
in	*cmdp*	pointer to the command descriptor
in	*n*	number of bytes to send
out	*rxbuf*	the pointer to the receive buffer

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.31.5.8 void qspiCommand (**QSPIDriver** ∗ *qspip*, const **qspi_command_t** ∗ *cmdp*)

Sends a command without data phase.

Precondition

In order to use this function the option `QSPI_USE_WAIT` must be enabled.
In order to use this function the driver must have been configured without callbacks (`end_cb` = `NULL`).

Parameters

in	*qspip*	pointer to the `QSPIDriver` object
in	*cmdp*	pointer to the command descriptor

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.31.5.9 void qspiSend (**QSPIDriver** ∗ *qspip*, const **qspi_command_t** ∗ *cmdp*, size_t *n*, const uint8_t ∗ *txbuf*)

Sends a command with data over the QSPI bus.

Precondition

In order to use this function the option `QSPI_USE_WAIT` must be enabled.
In order to use this function the driver must have been configured without callbacks (`end_cb` = `NULL`).

Parameters

in	*qspip*	pointer to the `QSPIDriver` object
in	*cmdp*	pointer to the command descriptor
in	*n*	number of bytes to send
in	*txbuf*	the pointer to the transmit buffer

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.31.5.10 void qspiReceive (QSPIDriver ∗ *qspip*, const qspi_command_t ∗ *cmdp*, size_t *n*, uint8_t ∗ *rxbuf*)

Sends a command then receives data over the QSPI bus.

Precondition

In order to use this function the option `QSPI_USE_WAIT` must be enabled.
In order to use this function the driver must have been configured without callbacks (`end_cb` = `NULL`).

Parameters

`in`	*qspip*	pointer to the `QSPIDriver` object
`in`	*cmdp*	pointer to the command descriptor
`in`	*n*	number of bytes to send
`out`	*rxbuf*	the pointer to the receive buffer

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.31.5.11 void qspiMapFlash (QSPIDriver ∗ *qspip*, const qspi_command_t ∗ *cmdp*, uint8_t ∗∗ *addrp*)

Maps in memory space a QSPI flash device.

Precondition

The memory flash device must be initialized appropriately before mapping it in memory space.

Parameters

`in`	*qspip*	pointer to the `QSPIDriver` object
`in`	*cmdp*	pointer to the command descriptor
`out`	*addrp*	pointer to the memory start address of the mapped flash or `NULL`

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.31.5.12 void qspiUnmapFlash (QSPIDriver ∗ *qspip*)

Maps in memory space a QSPI flash device.

Postcondition

The memory flash device must be re-initialized for normal commands exchange.

Parameters

in	*qspip*	pointer to the `QSPIDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.31.5.13 void qspiAcquireBus (**QSPIDriver** ∗ *qspip*)

Gains exclusive access to the QSPI bus.

This function tries to gain ownership to the QSPI bus, if the bus is already being used then the invoking thread is queued.

Precondition

In order to use this function the option `QSPI_USE_MUTUAL_EXCLUSION` must be enabled.

Parameters

in	*qspip*	pointer to the `QSPIDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.31.5.14 void qspiReleaseBus (**QSPIDriver** ∗ *qspip*)

Releases exclusive access to the QSPI bus.

Precondition

In order to use this function the option `QSPI_USE_MUTUAL_EXCLUSION` must be enabled.

Parameters

in	*qspip*	pointer to the `QSPIDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

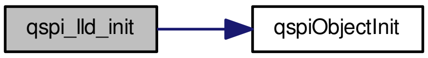
7.31.5.15 void qspi_lld_init (void)

Low level QSPI driver initialization.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.31.5.16 void qspi_lld_start (**QSPIDriver** ∗ *qspip*)

Configures and activates the QSPI peripheral.

Parameters

`in`	*qspip*	pointer to the `QSPIDriver` object

Function Class:

Not an API, this function is for internal use only.

7.31.5.17 void qspi_lld_stop (**QSPIDriver** ∗ *qspip*)

Deactivates the QSPI peripheral.

Parameters

`in`	*qspip*	pointer to the `QSPIDriver` object

Function Class:

Not an API, this function is for internal use only.

7.31.5.18 void qspi_lld_command (**QSPIDriver** ∗ *qspip*, const **qspi_command_t** ∗ *cmdp*)

Sends a command without data phase.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

`in`	*qspip*	pointer to the `QSPIDriver` object
`in`	*cmd*	pointer to the command descriptor

Function Class:

Not an API, this function is for internal use only.

7.31.5.19 void qspi_lld_send (QSPIDriver ∗ *qspip*, const qspi_command_t ∗ *cmdp*, size_t *n*, const uint8_t ∗ *txbuf*)

Sends a command with data over the QSPI bus.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	*qspip*	pointer to the `QSPIDriver` object
in	*cmd*	pointer to the command descriptor
in	*n*	number of bytes to send
in	*txbuf*	the pointer to the transmit buffer

Function Class:

Not an API, this function is for internal use only.

7.31.5.20 void qspi_lld_receive (QSPIDriver ∗ *qspip*, const qspi_command_t ∗ *cmdp*, size_t *n*, uint8_t ∗ *rxbuf*)

Sends a command then receives data over the QSPI bus.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	*qspip*	pointer to the `QSPIDriver` object
in	*cmd*	pointer to the command descriptor
in	*n*	number of bytes to send
out	*rxbuf*	the pointer to the receive buffer

Function Class:

Not an API, this function is for internal use only.

7.31.5.21 void qspi_lld_map_flash (QSPIDriver ∗ *qspip*, const qspi_command_t ∗ *cmdp*, uint8_t ∗∗ *addrp*)

Maps in memory space a QSPI flash device.

Precondition

The memory flash device must be initialized appropriately before mapping it in memory space.

Parameters

in	*qspip*	pointer to the `QSPIDriver` object
in	*cmdp*	pointer to the command descriptor
out	*addrp*	pointer to the memory start address of the mapped flash or `NULL`

Function Class:

Not an API, this function is for internal use only.

7.31.5.22 void qspi_lld_unmap_flash (QSPIDriver ∗ *qspip*)

Maps in memory space a QSPI flash device.

Postcondition

The memory flash device must be re-initialized for normal commands exchange.

Parameters

in	*qspip*	pointer to the `QSPIDriver` object

Function Class:

Not an API, this function is for internal use only.

7.31.6 Variable Documentation

7.31.6.1 QSPIDriver QSPID1

QSPID1 driver identifier.

7.32 RTC Driver

Generic RTC Driver.

7.32.1 Detailed Description

Generic RTC Driver.

This module defines an abstract interface for a Real Time Clock Peripheral.

Precondition

In order to use the RTC driver the `HAL_USE_RTC` option must be enabled in `halconf.h`.

Macros

- #define RTC_BASE_YEAR 1980U

  Base year of the calendar.
- #define _rtc_driver_methods _file_stream_methods

  FileStream specific methods.

Date/Time bit masks for FAT format

- #define **RTC_FAT_TIME_SECONDS_MASK** 0x0000001FU
- #define **RTC_FAT_TIME_MINUTES_MASK** 0x000007E0U
- #define **RTC_FAT_TIME_HOURS_MASK** 0x0000F800U
- #define **RTC_FAT_DATE_DAYS_MASK** 0x001F0000U
- #define **RTC_FAT_DATE_MONTHS_MASK** 0x01E00000U
- #define **RTC_FAT_DATE_YEARS_MASK** 0xFE000000U

Day of week encoding

- #define **RTC_DAY_CATURDAY** 0U
- #define **RTC_DAY_MONDAY** 1U
- #define **RTC_DAY_TUESDAY** 2U
- #define **RTC_DAY_WEDNESDAY** 3U
- #define **RTC_DAY_THURSDAY** 4U
- #define **RTC_DAY_FRIDAY** 5U
- #define **RTC_DAY_SATURDAY** 6U
- #define **RTC_DAY_SUNDAY** 7U

Implementation capabilities

- #define RTC_SUPPORTS_CALLBACKS TRUE

  Callback support int the driver.
- #define RTC_ALARMS 2

  Number of alarms available.
- #define RTC_HAS_STORAGE FALSE

  Presence of a local persistent storage.

PLATFORM configuration options

- #define PLATFORM_RTC_USE_RTC1 FALSE

  RTCD1 driver enable switch.

Typedefs

- typedef struct RTCDriver RTCDriver

  Type of a structure representing an RTC driver.
- typedef uint32_t rtcalarm_t

  Type of an RTC alarm number.
- typedef void(∗ rtccb_t) (RTCDriver ∗rtcp, rtcevent_t event)

  Type of a generic RTC callback.

Data Structures

- struct RTCDateTime

  Type of a structure representing an RTC date/time stamp.
- struct RTCAlarm

  Type of a structure representing an RTC alarm time stamp.
- struct RTCDriverVMT

  `RTCDriver` virtual methods table.
- struct RTCDriver

  Structure representing an RTC driver.

Functions

- void rtcInit (void)

  RTC Driver initialization.
- void rtcObjectInit (RTCDriver ∗rtcp)

  Initializes a generic RTC driver object.
- void rtcSetTime (RTCDriver ∗rtcp, const RTCDateTime ∗timespec)

  Set current time.
- void rtcGetTime (RTCDriver ∗rtcp, RTCDateTime ∗timespec)

  Get current time.
- void rtcSetAlarm (RTCDriver ∗rtcp, rtcalarm_t alarm, const RTCAlarm ∗alarmspec)

  Set alarm time.
- void rtcGetAlarm (RTCDriver ∗rtcp, rtcalarm_t alarm, RTCAlarm ∗alarmspec)

  Get current alarm.
- void rtcSetCallback (RTCDriver ∗rtcp, rtccb_t callback)

  Enables or disables RTC callbacks.
- void rtcConvertDateTimeToStructTm (const RTCDateTime ∗timespec, struct tm ∗timp, uint32_t ∗tv_msec)

  Convert `RTCDateTime` to broken-down time structure.
- void rtcConvertStructTmToDateTime (const struct tm ∗timp, uint32_t tv_msec, RTCDateTime ∗timespec)

  Convert broken-down time structure to `RTCDateTime`.
- uint32_t rtcConvertDateTimeToFAT (const RTCDateTime ∗timespec)

  Get current time in format suitable for usage in FAT file system.
- void rtc_lld_init (void)

  RTC driver identifier.
- void rtc_lld_set_time (RTCDriver ∗rtcp, const RTCDateTime ∗timespec)

Set current time.
- void rtc_lld_get_time (RTCDriver ∗rtcp, RTCDateTime ∗timespec)

  Get current time.
- void rtc_lld_set_alarm (RTCDriver ∗rtcp, rtcalarm_t alarm, const RTCAlarm ∗alarmspec)

  Set alarm time.
- void rtc_lld_get_alarm (RTCDriver ∗rtcp, rtcalarm_t alarm, RTCAlarm ∗alarmspec)

  Get alarm time.

Enumerations

7.32.2 Macro Definition Documentation

7.32.2.1 #define RTC_BASE_YEAR 1980U

Base year of the calendar.

7.32.2.2 #define RTC_SUPPORTS_CALLBACKS TRUE

Callback support int the driver.

7.32.2.3 #define RTC_ALARMS 2

Number of alarms available.

7.32.2.4 #define RTC_HAS_STORAGE FALSE

Presence of a local persistent storage.

7.32.2.5 #define PLATFORM_RTC_USE_RTC1 FALSE

RTCD1 driver enable switch.

If set to `TRUE` the support for RTC1 is included.

Note

The default is `FALSE`.

7.32.2.6 #define _rtc_driver_methods _file_stream_methods

FileStream specific methods.

7.32.3 Typedef Documentation

7.32.3.1 typedef struct RTCDriver RTCDriver

Type of a structure representing an RTC driver.

7.32.3.2 typedef uint32_t rtcalarm_t

Type of an RTC alarm number.

7.32.3.3 typedef void(∗ rtccb_t) (RTCDriver ∗rtcp, rtcevent_t event)

Type of a generic RTC callback.

7.32.4 Enumeration Type Documentation

7.32.4.1 enum rtcevent_t

Type of an RTC event.

7.32.5 Function Documentation

7.32.5.1 void rtcInit (void)

RTC Driver initialization.

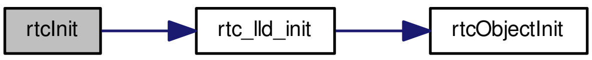
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



7.32.5.2 void rtcObjectInit (RTCDriver ∗ *rtcp*)

Initializes a generic RTC driver object.

The HW dependent part of the initialization has to be performed outside, usually in the hardware initialization code.

Parameters

out	*rtcp*	pointer to RTC driver structure

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.32.5.3 void rtcSetTime (RTCDriver ∗ *rtcp*, const RTCDateTime ∗ *timespec*)

Set current time.

Note

This function can be called from any context but limitations could be imposed by the low level implementation. It is guaranteed that the function can be called from thread context.
The function can be reentrant or not reentrant depending on the low level implementation.

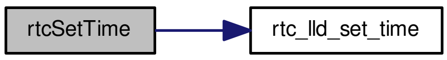
Parameters

`in`	*rtcp*	pointer to RTC driver structure
`in`	*timespec*	pointer to a `RTCDateTime` structure

Function Class:

Special function, this function has special requirements see the notes.

Here is the call graph for this function:



7.32.5.4 void rtcGetTime (**RTCDriver** ∗ *rtcp,* **RTCDateTime** ∗ *timespec*)

Get current time.

Note

This function can be called from any context but limitations could be imposed by the low level implementation. It is guaranteed that the function can be called from thread context.
The function can be reentrant or not reentrant depending on the low level implementation.

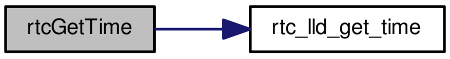
Parameters

`in`	*rtcp*	pointer to RTC driver structure
`out`	*timespec*	pointer to a `RTCDateTime` structure

Function Class:

Special function, this function has special requirements see the notes.

Here is the call graph for this function:



7.32.5.5 void rtcSetAlarm (**RTCDriver** ∗ *rtcp*, **rtcalarm_t** *alarm*, const **RTCAlarm** ∗ *alarmspec*)

Set alarm time.

Note

This function can be called from any context but limitations could be imposed by the low level implementation.
It is guaranteed that the function can be called from thread context.
The function can be reentrant or not reentrant depending on the low level implementation.

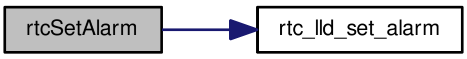
Parameters

`in`	*rtcp*	pointer to RTC driver structure
`in`	*alarm*	alarm identifier
`in`	*alarmspec*	pointer to a `RTCAlarm` structure or `NULL`

Function Class:

Special function, this function has special requirements see the notes.

Here is the call graph for this function:



7.32.5.6 void rtcGetAlarm (**RTCDriver** ∗ *rtcp*, **rtcalarm_t** *alarm*, **RTCAlarm** ∗ *alarmspec*)

Get current alarm.

Note

If an alarm has not been set then the returned alarm specification is not meaningful.
This function can be called from any context but limitations could be imposed by the low level implementation.
It is guaranteed that the function can be called from thread context.
The function can be reentrant or not reentrant depending on the low level implementation.

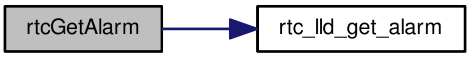
Parameters

in	*rtcp*	pointer to RTC driver structure
in	*alarm*	alarm identifier
out	*alarmspec*	pointer to a RTCAlarm structure

Function Class:

Special function, this function has special requirements see the notes.

Here is the call graph for this function:



7.32.5.7 void rtcSetCallback (**RTCDriver** ∗ *rtcp,* **rtccb_t** *callback*)

Enables or disables RTC callbacks.

This function enables or disables the callback, use a `NULL` pointer in order to disable it.

Note

This function can be called from any context but limitations could be imposed by the low level implementation.
It is guaranteed that the function can be called from thread context.
The function can be reentrant or not reentrant depending on the low level implementation.

Parameters

in	*rtcp*	pointer to RTC driver structure
in	*callback*	callback function pointer or `NULL`

Function Class:

Special function, this function has special requirements see the notes.

7.32.5.8 void rtcConvertDateTimeToStructTm (const **RTCDateTime** ∗ *timespec,* struct tm ∗ *timp,* uint32_t ∗ *tv_msec*)

Convert RTCDateTime to broken-down time structure.

Parameters

in	*timespec*	pointer to a RTCDateTime structure
out	*timp*	pointer to a broken-down time structure
out	*tv_msec*	pointer to milliseconds value or `NULL`

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.32.5.9 void rtcConvertStructTmToDateTime (const struct tm ∗ *timp*, uint32_t *tv_msec*, RTCDateTime ∗ *timespec*)

Convert broken-down time structure to `RTCDateTime`.

Parameters

`in`	*timp*	pointer to a broken-down time structure
`in`	*tv_msec*	milliseconds value
`out`	*timespec*	pointer to a `RTCDateTime` structure

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.32.5.10 uint32_t rtcConvertDateTimeToFAT (const RTCDateTime ∗ *timespec*)

Get current time in format suitable for usage in FAT file system.

Note

The information about day of week and DST is lost in DOS format, the second field loses its least significant bit.

Parameters

`out`	*timespec*	pointer to a `RTCDateTime` structure

Returns

FAT date/time value.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.32.5.11 void rtc_lld_init (void)

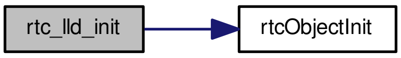
RTC driver identifier.

Enable access to registers.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.32.5.12 void rtc_lld_set_time (**RTCDriver** ∗ *rtcp*, const **RTCDateTime** ∗ *timespec*)

Set current time.

Note

Fractional part will be silently ignored. There is no possibility to set it on PLATFORM platform.
The function can be called from any context.

Parameters

in	*rtcp*	pointer to RTC driver structure
in	*timespec*	pointer to a `RTCDateTime` structure

Function Class:

Not an API, this function is for internal use only.

7.32.5.13 void rtc_lld_get_time (**RTCDriver** ∗ *rtcp*, **RTCDateTime** ∗ *timespec*)

Get current time.

Note

The function can be called from any context.

Parameters

in	*rtcp*	pointer to RTC driver structure
out	*timespec*	pointer to a `RTCDateTime` structure

Function Class:

Not an API, this function is for internal use only.

7.32.5.14 void rtc_lld_set_alarm (**RTCDriver** ∗ *rtcp*, **rtcalarm_t** *alarm*, const **RTCAlarm** ∗ *alarmspec*)

Set alarm time.

Note

Default value after BKP domain reset for both comparators is 0.
Function does not performs any checks of alarm time validity.
The function can be called from any context.

Parameters

in	*rtcp*	pointer to RTC driver structure.
in	*alarm*	alarm identifier. Can be 1 or 2.
in	*alarmspec*	pointer to a `RTCAlarm` structure.

Function Class:

Not an API, this function is for internal use only.

7.32.5.15 void rtc_lld_get_alarm (RTCDriver * *rtcp*, rtcalarm_t *alarm*, RTCAlarm * *alarmspec*)

Get alarm time.

Note

The function can be called from any context.

Parameters

in	*rtcp*	pointer to RTC driver structure
in	*alarm*	alarm identifier
out	*alarmspec*	pointer to a `RTCAlarm` structure

Function Class:

Not an API, this function is for internal use only.

7.33 SDC Driver

Generic SD Card Driver.

7.33.1 Detailed Description

Generic SD Card Driver.

This module implements a generic SDC (Secure Digital Card) driver.

Precondition

In order to use the SDC driver the `HAL_USE_SDC` option must be enabled in `halconf.h`.

7.33.2 Driver State Machine

This driver implements a state machine internally, see the Abstract I/O Block Device module documentation for details.

7.33.3 Driver Operations

This driver allows to read or write single or multiple 512 bytes blocks on a SD Card.

Macros

- #define _sdc_driver_methods _mmcsd_block_device_methods

  `SDCDriver` specific methods.

SD card types

- #define **SDC_MODE_CARDTYPE_MASK** 0xFU
- #define **SDC_MODE_CARDTYPE_SDV11** 0U
- #define **SDC_MODE_CARDTYPE_SDV20** 1U
- #define **SDC_MODE_CARDTYPE_MMC** 2U
- #define **SDC_MODE_HIGH_CAPACITY** 0x10U

SDC bus error conditions

- #define **SDC_NO_ERROR** 0U
- #define **SDC_CMD_CRC_ERROR** 1U
- #define **SDC_DATA_CRC_ERROR** 2U
- #define **SDC_DATA_TIMEOUT** 4U
- #define **SDC_COMMAND_TIMEOUT** 8U
- #define **SDC_TX_UNDERRUN** 16U
- #define **SDC_RX_OVERRUN** 32U
- #define **SDC_STARTBIT_ERROR** 64U
- #define **SDC_OVERFLOW_ERROR** 128U
- #define **SDC_UNHANDLED_ERROR** 0xFFFFFFFFU

SDC configuration options

- #define SDC_INIT_RETRY 100

  Number of initialization attempts before rejecting the card.
- #define SDC_MMC_SUPPORT FALSE

  Include support for MMC cards.
- #define SDC_NICE_WAITING TRUE

  Delays insertions.
- #define SDC_INIT_OCR_V20 0x50FF8000U

  OCR initialization constant for V20 cards.
- #define SDC_INIT_OCR 0x80100000U

  OCR initialization constant for non-V20 cards.

Macro Functions

- #define sdcIsCardInserted(sdcp) (sdc_lld_is_card_inserted(sdcp))

  Returns the card insertion status.
- #define sdcIsWriteProtected(sdcp) (sdc_lld_is_write_protected(sdcp))

  Returns the write protect status.

PLATFORM configuration options

- #define PLATFORM_SDC_USE_SDC1 FALSE

  PWMD1 driver enable switch.

Typedefs

- typedef uint32_t sdcmode_t

  Type of card flags.
- typedef uint32_t sdcflags_t

  SDC Driver condition flags type.
- typedef struct SDCDriver SDCDriver

  Type of a structure representing an SDC driver.

Data Structures

- struct SDCConfig

  Driver configuration structure.
- struct SDCDriverVMT

  `SDCDriver` *virtual methods table.*
- struct SDCDriver

  Structure representing an SDC driver.

Functions

- static bool mode_detect (SDCDriver ∗sdcp)

  Detects card mode.
- static bool mmc_init (SDCDriver ∗sdcp)

  Init procedure for MMC.
- static bool sdc_init (SDCDriver ∗sdcp)

Init procedure for SDC.
- static uint32_t mmc_cmd6_construct (mmc_switch_t access, uint32_t idx, uint32_t value, uint32_t cmd_set)

  Constructs CMD6 argument for MMC.
- static uint32_t sdc_cmd6_construct (sd_switch_t mode, sd_switch_function_t function, uint32_t value)

  Constructs CMD6 argument for SDC.
- static uint16_t sdc_cmd6_extract_info (sd_switch_function_t function, const uint8_t ∗buf)

  Extracts information from CMD6 answer.
- static bool sdc_cmd6_check_status (sd_switch_function_t function, const uint8_t ∗buf)

  Checks status after switching using CMD6.
- static bool sdc_detect_bus_clk (SDCDriver ∗sdcp, sdcbusclk_t ∗clk)

  Reads supported bus clock and switch SDC to appropriate mode.
- static bool mmc_detect_bus_clk (SDCDriver ∗sdcp, sdcbusclk_t ∗clk)

  Reads supported bus clock and switch MMC to appropriate mode.
- static bool detect_bus_clk (SDCDriver ∗sdcp, sdcbusclk_t ∗clk)

  Reads supported bus clock and switch card to appropriate mode.
- static bool sdc_set_bus_width (SDCDriver ∗sdcp)

  Sets bus width for SDC.
- static bool mmc_set_bus_width (SDCDriver ∗sdcp)

  Sets bus width for MMC.
- bool _sdc_wait_for_transfer_state (SDCDriver ∗sdcp)

  Wait for the card to complete pending operations.
- void sdcInit (void)

  SDC Driver initialization.
- void sdcObjectInit (SDCDriver ∗sdcp)

  Initializes the standard part of a `SDCDriver` structure.
- void sdcStart (SDCDriver ∗sdcp, const SDCConfig ∗config)

  Configures and activates the SDC peripheral.
- void sdcStop (SDCDriver ∗sdcp)

  Deactivates the SDC peripheral.
- bool sdcConnect (SDCDriver ∗sdcp)

  Performs the initialization procedure on the inserted card.
- bool sdcDisconnect (SDCDriver ∗sdcp)

  Brings the driver in a state safe for card removal.
- bool sdcRead (SDCDriver ∗sdcp, uint32_t startblk, uint8_t ∗buf, uint32_t n)

  Reads one or more blocks.
- bool sdcWrite (SDCDriver ∗sdcp, uint32_t startblk, const uint8_t ∗buf, uint32_t n)

  Writes one or more blocks.
- sdcflags_t sdcGetAndClearErrors (SDCDriver ∗sdcp)

  Returns the errors mask associated to the previous operation.
- bool sdcSync (SDCDriver ∗sdcp)

  Waits for card idle condition.
- bool sdcGetInfo (SDCDriver ∗sdcp, BlockDeviceInfo ∗bdip)

  Returns the media info.
- bool sdcErase (SDCDriver ∗sdcp, uint32_t startblk, uint32_t endblk)

  Erases the supplied blocks.
- void sdc_lld_init (void)

  Low level SDC driver initialization.
- void sdc_lld_start (SDCDriver ∗sdcp)

  Configures and activates the SDC peripheral.
- void sdc_lld_stop (SDCDriver ∗sdcp)

  Deactivates the SDC peripheral.

- void sdc_lld_start_clk (SDCDriver ∗sdcp)

  Starts the SDIO clock and sets it to init mode (400kHz or less).
- void sdc_lld_set_data_clk (SDCDriver ∗sdcp, sdcbusclk_t clk)

  Sets the SDIO clock to data mode (25MHz or less).
- void sdc_lld_stop_clk (SDCDriver ∗sdcp)

  Stops the SDIO clock.
- void sdc_lld_set_bus_mode (SDCDriver ∗sdcp, sdcbusmode_t mode)

  Switches the bus to 4 bits mode.
- void sdc_lld_send_cmd_none (SDCDriver ∗sdcp, uint8_t cmd, uint32_t arg)

  Sends an SDIO command with no response expected.
- bool sdc_lld_send_cmd_short (SDCDriver ∗sdcp, uint8_t cmd, uint32_t arg, uint32_t ∗resp)

  Sends an SDIO command with a short response expected.
- bool sdc_lld_send_cmd_short_crc (SDCDriver ∗sdcp, uint8_t cmd, uint32_t arg, uint32_t ∗resp)

  Sends an SDIO command with a short response expected and CRC.
- bool sdc_lld_send_cmd_long_crc (SDCDriver ∗sdcp, uint8_t cmd, uint32_t arg, uint32_t ∗resp)

  Sends an SDIO command with a long response expected and CRC.
- bool sdc_lld_read (SDCDriver ∗sdcp, uint32_t startblk, uint8_t ∗buf, uint32_t n)

  Reads one or more blocks.
- bool sdc_lld_write (SDCDriver ∗sdcp, uint32_t startblk, const uint8_t ∗buf, uint32_t n)

  Writes one or more blocks.
- bool sdc_lld_sync (SDCDriver ∗sdcp)

  Waits for card idle condition.

Enumerations

Variables

- static const struct SDCDriverVMT sdc_vmt

  Virtual methods table.
- SDCDriver SDCD1

  SDCD1 driver identifier.

7.33.4 Macro Definition Documentation

7.33.4.1 #define SDC_INIT_RETRY 100

Number of initialization attempts before rejecting the card.

Note

Attempts are performed at 10mS intervals.

7.33.4.2 #define SDC_MMC_SUPPORT FALSE

Include support for MMC cards.

Note

MMC support is not yet implemented so this option must be kept at `FALSE`.

7.33.4.3 #define SDC_NICE_WAITING TRUE

Delays insertions.

If enabled this options inserts delays into the MMC waiting routines releasing some extra CPU time for the threads with lower priority, this may slow down the driver a bit however.

7.33.4.4 #define SDC_INIT_OCR_V20 0x50FF8000U

OCR initialization constant for V20 cards.

7.33.4.5 #define SDC_INIT_OCR 0x80100000U

OCR initialization constant for non-V20 cards.

7.33.4.6 #define sdcIsCardInserted(*sdcp*) (sdc_lld_is_card_inserted(sdcp))

Returns the card insertion status.

Note

This macro wraps a low level function named `sdc_lld_is_card_inserted()`, this function must be provided by the application because it is not part of the SDC driver.

Parameters

`in`	*sdcp*	pointer to the `SDCDriver` object

Returns

The card state.

Return values

FALSE	card not inserted.
TRUE	card inserted.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.33.4.7 #define sdcIsWriteProtected(*sdcp*) (sdc_lld_is_write_protected(sdcp))

Returns the write protect status.

Note

This macro wraps a low level function named `sdc_lld_is_write_protected()`, this function must be provided by the application because it is not part of the SDC driver.

Parameters

in	*sdcp*	pointer to the `SDCDriver` object

Returns

The card state.

Return values

FALSE	not write protected.
TRUE	write protected.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.33.4.8 #define PLATFORM_SDC_USE_SDC1 FALSE

PWMD1 driver enable switch.

If set to `TRUE` the support for PWM1 is included.

Note

The default is `FALSE`.

7.33.4.9 #define _sdc_driver_methods _mmcsd_block_device_methods

`SDCDriver` specific methods.

7.33.5 Typedef Documentation

7.33.5.1 typedef uint32_t sdcmode_t

Type of card flags.

7.33.5.2 typedef uint32_t sdcflags_t

SDC Driver condition flags type.

7.33.5.3 typedef struct SDCDriver SDCDriver

Type of a structure representing an SDC driver.

7.33.6 Enumeration Type Documentation

7.33.6.1 enum mmc_switch_t

MMC switch mode.

7.33.6.2 enum sd_switch_t

SDC switch mode.

7.33.6.3 enum sd_switch_function_t

SDC switch function.

7.33.6.4 enum sdcbusmode_t

Type of SDIO bus mode.

7.33.6.5 enum sdcbusclk_t

Max supported clock.

7.33.7 Function Documentation

7.33.7.1 static bool mode_detect (SDCDriver * *sdcp*) [static]

Detects card mode.

Parameters

in	*sdcp*	pointer to the SDCDriver object

Returns

The operation status.

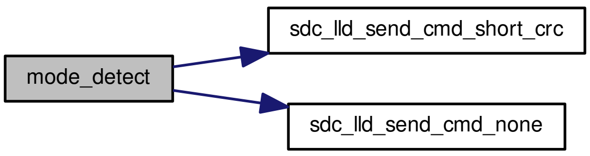
Return values

HAL_SUCCESS	operation succeeded.
HAL_FAILED	operation failed.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.33.7.2 static bool mmc_init (SDCDriver * *sdcp*) [static]

Init procedure for MMC.

Parameters

in	*sdcp*	pointer to the SDCDriver object

Returns

The operation status.

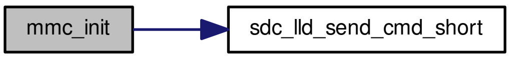
Return values

HAL_SUCCESS	operation succeeded.
HAL_FAILED	operation failed.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.33.7.3 static bool sdc_init (SDCDriver * *sdcp*) [static]

Init procedure for SDC.

Parameters

in	*sdcp*	pointer to the SDCDriver object

Returns

The operation status.

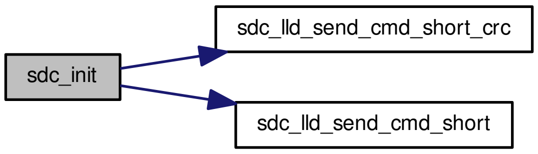
Return values

HAL_SUCCESS	operation succeeded.
HAL_FAILED	operation failed.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.33.7.4 static uint32_t mmc_cmd6_construct (mmc_switch_t *access*, uint32_t *idx*, uint32_t *value*, uint32_t *cmd_set*) [static]

Constructs CMD6 argument for MMC.

Parameters

in	*access*	EXT_CSD access mode
in	*idx*	EXT_CSD byte number
in	*value*	value to be written in target field
in	*cmd_set*	switch current command set

Returns

CMD6 argument.

Function Class:

Not an API, this function is for internal use only.

7.33.7.5 static uint32_t sdc_cmd6_construct (sd_switch_t *mode*, sd_switch_function_t *function*, uint32_t *value*) [static]

Constructs CMD6 argument for SDC.

Parameters

in	*mode*	switch/test mode
in	*function*	function number to be switched
in	*value*	value to be written in target function

Returns

CMD6 argument.

Function Class:

Not an API, this function is for internal use only.

7.33.7.6 static uint16_t sdc_cmd6_extract_info (sd_switch_function_t *function,* const uint8_t ∗ *buf*) [static]

Extracts information from CMD6 answer.

Parameters

in	*function*	function number to be switched
in	*buf*	buffer with answer

Returns

extracted answer.

Function Class:

Not an API, this function is for internal use only.

7.33.7.7 static bool sdc_cmd6_check_status (sd_switch_function_t *function,* const uint8_t ∗ *buf*) [static]

Checks status after switching using CMD6.

Parameters

in	*function*	function number to be switched
in	*buf*	buffer with answer

Returns

The operation status.

Return values

HAL_SUCCESS	operation succeeded.
HAL_FAILED	operation failed.

Function Class:

Not an API, this function is for internal use only.

7.33.7.8 static bool sdc_detect_bus_clk (SDCDriver ∗ *sdcp,* sdcbusclk_t ∗ *clk*) [static]

Reads supported bus clock and switch SDC to appropriate mode.

Parameters

`in`	*sdcp*	pointer to the `SDCDriver` object
`out`	*clk*	pointer to clock enum

Returns

The operation status.

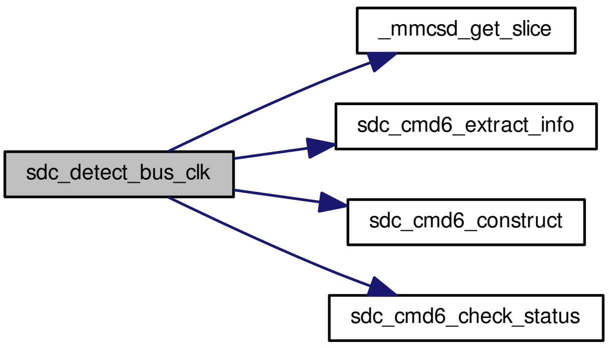
Return values

HAL_SUCCESS	operation succeeded.
HAL_FAILED	operation failed.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.33.7.9 static bool mmc_detect_bus_clk (**SDCDriver** ∗ *sdcp,* **sdcbusclk_t** ∗ *clk*) `[static]`

Reads supported bus clock and switch MMC to appropriate mode.

Parameters

`in`	*sdcp*	pointer to the `SDCDriver` object
`out`	*clk*	pointer to clock enum

Returns

The operation status.

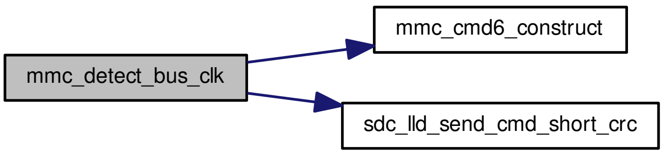
Return values

HAL_SUCCESS	operation succeeded.
HAL_FAILED	operation failed.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.33.7.10 static bool detect_bus_clk (SDCDriver ∗ *sdcp*, sdcbusclk_t ∗ *clk*) [static]

Reads supported bus clock and switch card to appropriate mode.

Parameters

in	*sdcp*	pointer to the SDCDriver object
out	*clk*	pointer to clock enum

Returns

The operation status.

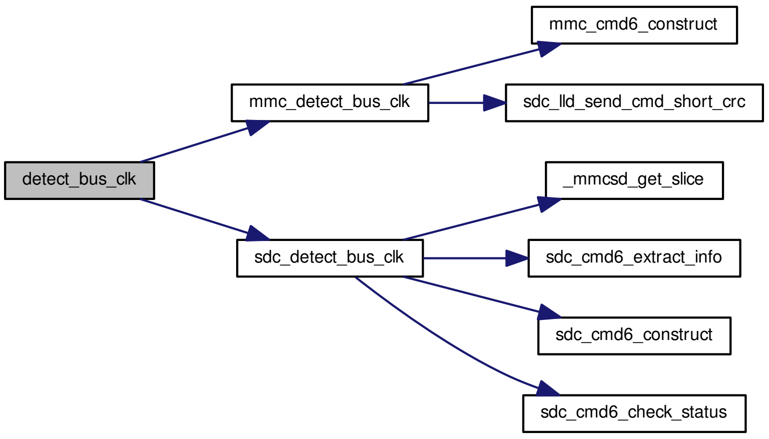
Return values

HAL_SUCCESS	operation succeeded.
HAL_FAILED	operation failed.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.33.7.11 static bool sdc_set_bus_width (**SDCDriver** ∗ *sdcp*) [static]

Sets bus width for SDC.

Parameters

in	*sdcp*	pointer to the SDCDriver object

Returns

The operation status.

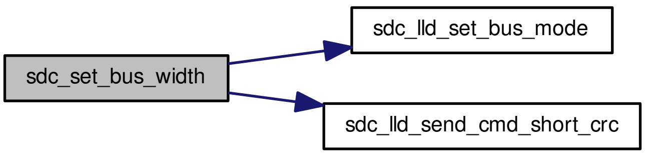
Return values

HAL_SUCCESS	operation succeeded.
HAL_FAILED	operation failed.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.33.7.12 static bool mmc_set_bus_width (SDCDriver * *sdcp*) [static]

Sets bus width for MMC.

Parameters

in	*sdcp*	pointer to the SDCDriver object

Returns

The operation status.

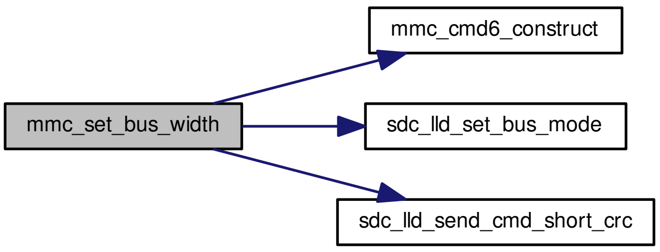
Return values

HAL_SUCCESS	operation succeeded.
HAL_FAILED	operation failed.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.33.7.13 bool _sdc_wait_for_transfer_state (**SDCDriver** * *sdcp*)

Wait for the card to complete pending operations.

Parameters

in	*sdcp*	pointer to the SDCDriver object

Returns

The operation status.

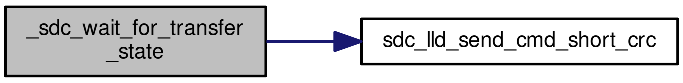
Return values

HAL_SUCCESS	operation succeeded.
HAL_FAILED	operation failed.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.33.7.14 void sdcInit (void)

SDC Driver initialization.

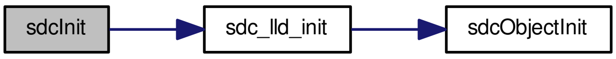
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



7.33.7.15 void sdcObjectInit (SDCDriver ∗ *sdcp*)

Initializes the standard part of a `SDCDriver` structure.

Parameters

`out`	*sdcp*	pointer to the `SDCDriver` object

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.33.7.16 void sdcStart (SDCDriver ∗ *sdcp*, const SDCConfig ∗ *config*)

Configures and activates the SDC peripheral.

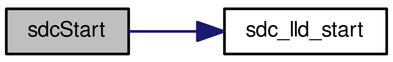
Parameters

`in`	*sdcp*	pointer to the `SDCDriver` object
`in`	*config*	pointer to the `SDCConfig` object, can be `NULL` if the driver supports a default configuration or requires no configuration

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.33.7.17 void sdcStop (SDCDriver ∗ *sdcp*)

Deactivates the SDC peripheral.

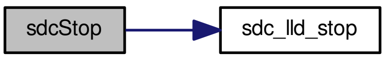
Parameters

`in`	*sdcp*	pointer to the `SDCDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.33.7.18 bool sdcConnect (SDCDriver * *sdcp*)

Performs the initialization procedure on the inserted card.

This function should be invoked when a card is inserted and brings the driver in the `BLK_READY` state where it is possible to perform read and write operations.

Parameters

`in`	*sdcp*	pointer to the `SDCDriver` object

Returns

The operation status.

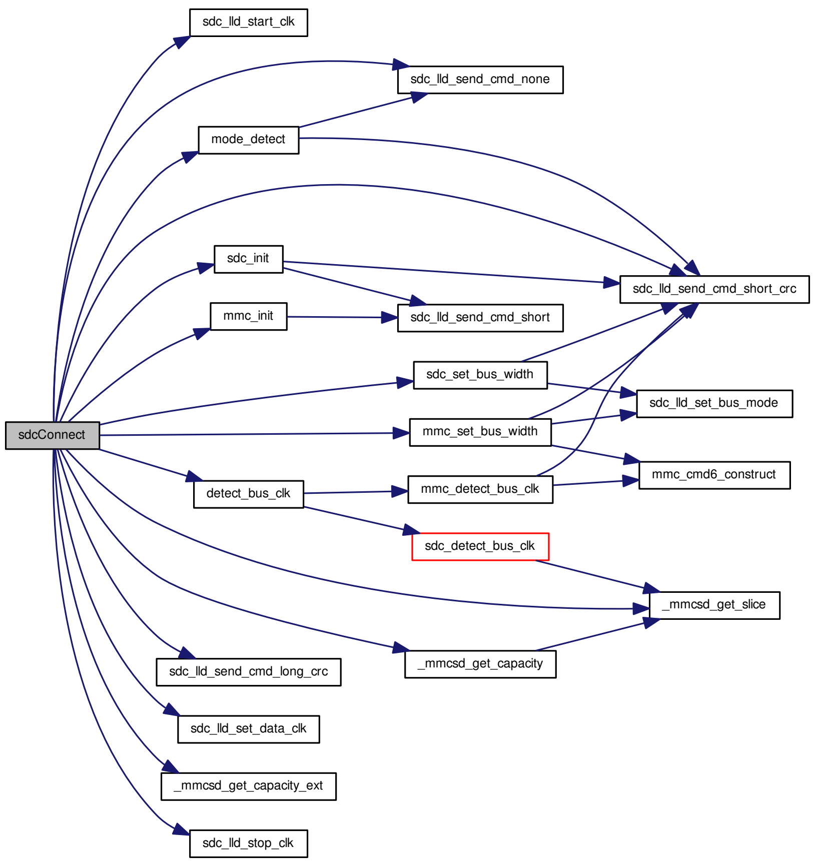
Return values

HAL_SUCCESS	operation succeeded.
HAL_FAILED	operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.33.7.19 bool sdcDisconnect (SDCDriver ∗ *sdcp*)

Brings the driver in a state safe for card removal.

Parameters

in	*sdcp*	pointer to the SDCDriver object

Returns

The operation status.

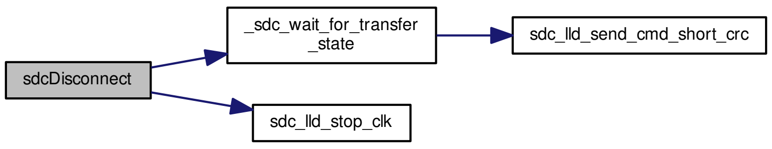
Return values

HAL_SUCCESS	operation succeeded.
HAL_FAILED	operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.33.7.20 bool sdcRead (SDCDriver * *sdcp*, uint32_t *startblk*, uint8_t * *buf*, uint32_t *n*)

Reads one or more blocks.

Precondition

The driver must be in the `BLK_READY` state after a successful sdcConnect() invocation.

Parameters

`in`	*sdcp*	pointer to the `SDCDriver` object
`in`	*startblk*	first block to read
`out`	*buf*	pointer to the read buffer
`in`	*n*	number of blocks to read

Returns

The operation status.

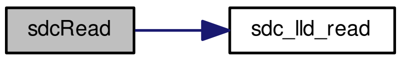
Return values

HAL_SUCCESS	operation succeeded.
HAL_FAILED	operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.33.7.21 bool sdcWrite (**SDCDriver** ∗ *sdcp,* uint32_t *startblk,* const uint8_t ∗ *buf,* uint32_t *n*)

Writes one or more blocks.

Precondition

The driver must be in the `BLK_READY` state after a successful sdcConnect() invocation.

Parameters

`in`	*sdcp*	pointer to the `SDCDriver` object
`in`	*startblk*	first block to write
`out`	*buf*	pointer to the write buffer
`in`	*n*	number of blocks to write

Returns

The operation status.

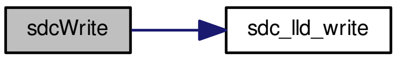
Return values

HAL_SUCCESS	operation succeeded.
HAL_FAILED	operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.33.7.22 sdcflags_t sdcGetAndClearErrors (**SDCDriver** ∗ *sdcp*)

Returns the errors mask associated to the previous operation.

Parameters

`in`	*sdcp*	pointer to the `SDCDriver` object

Returns

The errors mask.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.33.7.23 bool sdcSync (SDCDriver * *sdcp*)

Waits for card idle condition.

Parameters

in	*sdcp*	pointer to the `SDCDriver` object

Returns

The operation status.

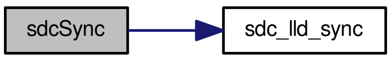
Return values

HAL_SUCCESS	the operation succeeded.
HAL_FAILED	the operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.33.7.24 bool sdcGetInfo (SDCDriver * *sdcp*, BlockDeviceInfo * *bdip*)

Returns the media info.

Parameters

in	*sdcp*	pointer to the `SDCDriver` object
out	*bdip*	pointer to a `BlockDeviceInfo` structure

Returns

The operation status.

Return values

HAL_SUCCESS	the operation succeeded.
HAL_FAILED	the operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.33.7.25 bool sdcErase (SDCDriver ∗ *sdcp*, uint32_t *startblk*, uint32_t *endblk*)

Erases the supplied blocks.

Parameters

`in`	*sdcp*	pointer to the `SDCDriver` object
`in`	*startblk*	starting block number
`in`	*endblk*	ending block number

Returns

The operation status.

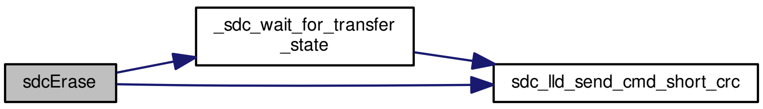
Return values

HAL_SUCCESS	the operation succeeded.
HAL_FAILED	the operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



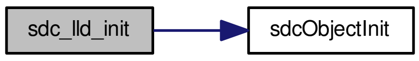
7.33.7.26 void sdc_lld_init (void)

Low level SDC driver initialization.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.33.7.27 void sdc_lld_start (SDCDriver * *sdcp*)

Configures and activates the SDC peripheral.

Parameters

`in`	*sdcp*	pointer to the `SDCDriver` object

Function Class:

Not an API, this function is for internal use only.

7.33.7.28 void sdc_lld_stop (SDCDriver * *sdcp*)

Deactivates the SDC peripheral.

Parameters

`in`	*sdcp*	pointer to the `SDCDriver` object

Function Class:

Not an API, this function is for internal use only.

7.33.7.29 void sdc_lld_start_clk (SDCDriver * *sdcp*)

Starts the SDIO clock and sets it to init mode (400kHz or less).

Parameters

`in`	*sdcp*	pointer to the `SDCDriver` object

Function Class:

Not an API, this function is for internal use only.

7.33.7.30 void sdc_lld_set_data_clk (SDCDriver ∗ *sdcp,* sdcbusclk_t *clk*)

Sets the SDIO clock to data mode (25MHz or less).

Parameters

in	*sdcp*	pointer to the SDCDriver object
in	*clk*	the clock mode

Function Class:

Not an API, this function is for internal use only.

7.33.7.31 void sdc_lld_stop_clk (SDCDriver ∗ *sdcp*)

Stops the SDIO clock.

Parameters

in	*sdcp*	pointer to the SDCDriver object

Function Class:

Not an API, this function is for internal use only.

7.33.7.32 void sdc_lld_set_bus_mode (SDCDriver ∗ *sdcp,* sdcbusmode_t *mode*)

Switches the bus to 4 bits mode.

Parameters

in	*sdcp*	pointer to the SDCDriver object
in	*mode*	bus mode

Function Class:

Not an API, this function is for internal use only.

7.33.7.33 void sdc_lld_send_cmd_none (SDCDriver ∗ *sdcp,* uint8_t *cmd,* uint32_t *arg*)

Sends an SDIO command with no response expected.

Parameters

in	*sdcp*	pointer to the SDCDriver object
in	*cmd*	card command
in	*arg*	command argument

Function Class:

Not an API, this function is for internal use only.

7.33.7.34 bool sdc_lld_send_cmd_short (SDCDriver ∗ *sdcp*, uint8_t *cmd*, uint32_t *arg*, uint32_t ∗ *resp*)

Sends an SDIO command with a short response expected.

Note

The CRC is not verified.

Parameters

`in`	*sdcp*	pointer to the `SDCDriver` object
`in`	*cmd*	card command
`in`	*arg*	command argument
`out`	*resp*	pointer to the response buffer (one word)

Returns

The operation status.

Return values

HAL_SUCCESS	operation succeeded.
HAL_FAILED	operation failed.

Function Class:

Not an API, this function is for internal use only.

7.33.7.35 bool sdc_lld_send_cmd_short_crc (SDCDriver ∗ *sdcp*, uint8_t *cmd*, uint32_t *arg*, uint32_t ∗ *resp*)

Sends an SDIO command with a short response expected and CRC.

Parameters

`in`	*sdcp*	pointer to the `SDCDriver` object
`in`	*cmd*	card command
`in`	*arg*	command argument
`out`	*resp*	pointer to the response buffer (one word)

Returns

The operation status.

Return values

HAL_SUCCESS	operation succeeded.
HAL_FAILED	operation failed.

Function Class:

Not an API, this function is for internal use only.

7.33.7.36 bool sdc_lld_send_cmd_long_crc (SDCDriver ∗ *sdcp,* uint8_t *cmd,* uint32_t *arg,* uint32_t ∗ *resp*)

Sends an SDIO command with a long response expected and CRC.

Parameters

in	*sdcp*	pointer to the SDCDriver object
in	*cmd*	card command
in	*arg*	command argument
out	*resp*	pointer to the response buffer (four words)

Returns

The operation status.

Return values

HAL_SUCCESS	operation succeeded.
HAL_FAILED	operation failed.

Function Class:

Not an API, this function is for internal use only.

7.33.7.37 bool sdc_lld_read (SDCDriver ∗ *sdcp,* uint32_t *startblk,* uint8_t ∗ *buf,* uint32_t *n*)

Reads one or more blocks.

Parameters

in	*sdcp*	pointer to the SDCDriver object
in	*startblk*	first block to read
out	*buf*	pointer to the read buffer
in	*n*	number of blocks to read

Returns

The operation status.

Return values

HAL_SUCCESS	operation succeeded.
HAL_FAILED	operation failed.

Function Class:

Not an API, this function is for internal use only.

7.33.7.38 bool sdc_lld_write (**SDCDriver** ∗ *sdcp,* uint32_t *startblk,* const uint8_t ∗ *buf,* uint32_t *n*)

Writes one or more blocks.

Parameters

`in`	*sdcp*	pointer to the `SDCDriver` object
`in`	*startblk*	first block to write
`out`	*buf*	pointer to the write buffer
`in`	*n*	number of blocks to write

Returns

The operation status.

Return values

HAL_SUCCESS	operation succeeded.
HAL_FAILED	operation failed.

Function Class:

Not an API, this function is for internal use only.

7.33.7.39 bool sdc_lld_sync (**SDCDriver** ∗ *sdcp*)

Waits for card idle condition.

Parameters

`in`	*sdcp*	pointer to the `SDCDriver` object

Returns

The operation status.

Return values

HAL_SUCCESS	the operation succeeded.
HAL_FAILED	the operation failed.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.33.8 Variable Documentation

7.33.8.1 const struct SDCDriverVMT sdc_vmt [static]

Initial value:

```
= {
  (bool (*)(void *))sdc_lld_is_card_inserted,
  (bool (*)(void *))sdc_lld_is_write_protected,
  (bool (*)(void *))sdcConnect,
  (bool (*)(void *))sdcDisconnect,
  (bool (*)(void *, uint32_t, uint8_t *, uint32_t))sdcRead,
  (bool (*)(void *, uint32_t, const uint8_t *, uint32_t))sdcWrite,
  (bool (*)(void *))sdcSync,
  (bool (*)(void *, BlockDeviceInfo *))sdcGetInfo
}
```

Virtual methods table.

7.33.8.2 SDCDriver SDCD1

SDCD1 driver identifier.

7.34 Serial Driver

Generic Serial Driver.

7.34.1 Detailed Description

Generic Serial Driver.

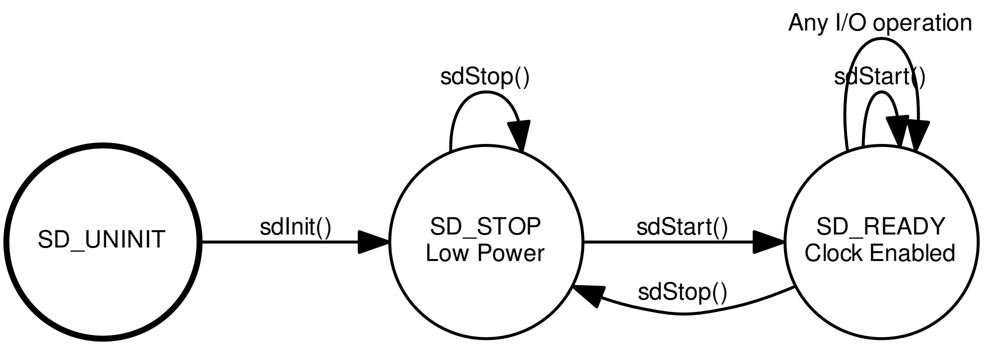
This module implements a generic full duplex serial driver. The driver implements a `SerialDriver` interface and uses I/O Queues for communication between the upper and the lower driver. Event flags are used to notify the application about incoming data, outgoing data and other I/O events.
The module also contains functions that make the implementation of the interrupt service routines much easier.

Precondition

In order to use the SERIAL driver the `HAL_USE_SERIAL` option must be enabled in `halconf.h`.

7.34.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



Macros

- #define _serial_driver_methods _base_asynchronous_channel_methods

  `SerialDriver` specific methods.
- #define _serial_driver_data

  `SerialDriver` specific data.

Serial status flags

- #define SD_PARITY_ERROR (eventflags_t)32

Parity.
- #define SD_FRAMING_ERROR (eventflags_t)64

  Framing.
- #define SD_OVERRUN_ERROR (eventflags_t)128

  Overflow.
- #define SD_NOISE_ERROR (eventflags_t)256

  Line noise.
- #define SD_BREAK_DETECTED (eventflags_t)512

  LIN Break.
- #define SD_QUEUE_FULL_ERROR (eventflags_t)1024

  Queue full.

Serial configuration options

- #define SERIAL_DEFAULT_BITRATE 38400

  Default bit rate.
- #define SERIAL_BUFFERS_SIZE 16

  Serial buffers size.

Macro Functions

- #define sdPut(sdp, b) oqPut(&(sdp)->oqueue, b)

  Direct write to a `SerialDriver`.
- #define sdPutTimeout(sdp, b, t) oqPutTimeout(&(sdp)->oqueue, b, t)

  Direct write to a `SerialDriver` with timeout specification.
- #define sdGet(sdp) iqGet(&(sdp)->iqueue)

  Direct read from a `SerialDriver`.
- #define sdGetTimeout(sdp, t) iqGetTimeout(&(sdp)->iqueue, t)

  Direct read from a `SerialDriver` with timeout specification.
- #define sdWrite(sdp, b, n) oqWriteTimeout(&(sdp)->oqueue, b, n, TIME_INFINITE)

  Direct blocking write to a `SerialDriver`.
- #define sdWriteTimeout(sdp, b, n, t) oqWriteTimeout(&(sdp)->oqueue, b, n, t)

  Direct blocking write to a `SerialDriver` with timeout specification.
- #define sdAsynchronousWrite(sdp, b, n) oqWriteTimeout(&(sdp)->oqueue, b, n, TIME_IMMEDIATE)

  Direct non-blocking write to a `SerialDriver`.
- #define sdRead(sdp, b, n) iqReadTimeout(&(sdp)->iqueue, b, n, TIME_INFINITE)

  Direct blocking read from a `SerialDriver`.
- #define sdReadTimeout(sdp, b, n, t) iqReadTimeout(&(sdp)->iqueue, b, n, t)

  Direct blocking read from a `SerialDriver` with timeout specification.
- #define sdAsynchronousRead(sdp, b, n) iqReadTimeout(&(sdp)->iqueue, b, n, TIME_IMMEDIATE)

  Direct non-blocking read from a `SerialDriver`.

PLATFORM configuration options

- #define PLATFORM_SERIAL_USE_USART1 FALSE

  USART1 driver enable switch.

Typedefs

- typedef struct SerialDriver SerialDriver

  Structure representing a serial driver.

Data Structures

- struct SerialDriverVMT

  `SerialDriver` virtual methods table.
- struct SerialDriver

  Full duplex serial driver class.
- struct SerialConfig

  PLATFORM Serial Driver configuration structure.

Functions

- void sdInit (void)

  Serial Driver initialization.
- void sdObjectInit (SerialDriver ∗sdp, qnotify_t inotify, qnotify_t onotify)

  Initializes a generic full duplex driver object.
- void sdStart (SerialDriver ∗sdp, const SerialConfig ∗config)

  Configures and starts the driver.
- void sdStop (SerialDriver ∗sdp)

  Stops the driver.
- void sdIncomingDataI (SerialDriver ∗sdp, uint8_t b)

  Handles incoming data.
- msg_t sdRequestDataI (SerialDriver ∗sdp)

  Handles outgoing data.
- bool sdPutWouldBlock (SerialDriver ∗sdp)

  Direct output check on a `SerialDriver`.
- bool sdGetWouldBlock (SerialDriver ∗sdp)

  Direct input check on a `SerialDriver`.
- void sd_lld_init (void)

  Low level serial driver initialization.
- void sd_lld_start (SerialDriver ∗sdp, const SerialConfig ∗config)

  Low level serial driver configuration and (re)start.
- void sd_lld_stop (SerialDriver ∗sdp)

  Low level serial driver stop.

Enumerations

Variables

- SerialDriver SD1

  USART1 serial driver identifier.
- static const SerialConfig default_config

  Driver default configuration.

7.34.3 Macro Definition Documentation

7.34.3.1 #define SD_PARITY_ERROR (eventflags_t)32

Parity.

7.34.3.2 #define SD_FRAMING_ERROR (eventflags_t)64

Framing.

7.34.3.3 #define SD_OVERRUN_ERROR (eventflags_t)128

Overflow.

7.34.3.4 #define SD_NOISE_ERROR (eventflags_t)256

Line noise.

7.34.3.5 #define SD_BREAK_DETECTED (eventflags_t)512

LIN Break.

7.34.3.6 #define SD_QUEUE_FULL_ERROR (eventflags_t)1024

Queue full.

7.34.3.7 #define SERIAL_DEFAULT_BITRATE 38400

Default bit rate.

Configuration parameter, this is the baud rate selected for the default configuration.

7.34.3.8 #define SERIAL_BUFFERS_SIZE 16

Serial buffers size.

Configuration parameter, you can change the depth of the queue buffers depending on the requirements of your application.

Note

> The default is 16 bytes for both the transmission and receive buffers.
> This is a global setting and it can be overridden by low level driver specific settings.

7.34.3.9 #define _serial_driver_methods _base_asynchronous_channel_methods

`SerialDriver` specific methods.

7.34.3.10 #define sdPut(*sdp, b*) oqPut(&(sdp)->oqueue, b)

Direct write to a `SerialDriver`.

Note

> This function bypasses the indirect access to the channel and writes directly on the output queue. This is faster but cannot be used to write to different channels implementations.

See also

chnPutTimeout()

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.34.3.11 #define sdPutTimeout(*sdp*, *b*, *t*) oqPutTimeout(&(sdp)->oqueue, b, t)

Direct write to a `SerialDriver` with timeout specification.

Note

This function bypasses the indirect access to the channel and writes directly on the output queue. This is faster but cannot be used to write to different channels implementations.

See also

chnPutTimeout()

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.34.3.12 #define sdGet(*sdp*) iqGet(&(sdp)->iqueue)

Direct read from a `SerialDriver`.

Note

This function bypasses the indirect access to the channel and reads directly from the input queue. This is faster but cannot be used to read from different channels implementations.

See also

chnGetTimeout()

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.34.3.13 #define sdGetTimeout(*sdp*, *t*) iqGetTimeout(&(sdp)->iqueue, t)

Direct read from a `SerialDriver` with timeout specification.

Note

This function bypasses the indirect access to the channel and reads directly from the input queue. This is faster but cannot be used to read from different channels implementations.

See also

chnGetTimeout()

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.34.3.14 #define sdWrite(*sdp,* *b,* *n*) **oqWriteTimeout(&(sdp)->oqueue, b, n, TIME_INFINITE)**

Direct blocking write to a `SerialDriver`.

Note

This function bypasses the indirect access to the channel and writes directly to the output queue. This is faster but cannot be used to write from different channels implementations.

See also

chnWrite()

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.34.3.15 #define sdWriteTimeout(*sdp,* *b,* *n,* *t*) **oqWriteTimeout(&(sdp)->oqueue, b, n, t)**

Direct blocking write to a `SerialDriver` with timeout specification.

Note

This function bypasses the indirect access to the channel and writes directly to the output queue. This is faster but cannot be used to write to different channels implementations.

See also

chnWriteTimeout()

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.34.3.16 #define sdAsynchronousWrite(*sdp,* *b,* *n*) **oqWriteTimeout(&(sdp)->oqueue, b, n, TIME_IMMEDIATE)**

Direct non-blocking write to a `SerialDriver`.

Note

This function bypasses the indirect access to the channel and writes directly to the output queue. This is faster but cannot be used to write to different channels implementations.

See also

chnWriteTimeout()

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.34.3.17 #define sdRead(*sdp,* *b,* *n*) iqReadTimeout(&(sdp)->iqueue, b, n, TIME_INFINITE)

Direct blocking read from a `SerialDriver`.

Note

This function bypasses the indirect access to the channel and reads directly from the input queue. This is faster but cannot be used to read from different channels implementations.

See also

chnRead()

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.34.3.18 #define sdReadTimeout(*sdp,* *b,* *n,* *t*) iqReadTimeout(&(sdp)->iqueue, b, n, t)

Direct blocking read from a `SerialDriver` with timeout specification.

Note

This function bypasses the indirect access to the channel and reads directly from the input queue. This is faster but cannot be used to read from different channels implementations.

See also

chnReadTimeout()

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.34.3.19 #define sdAsynchronousRead(*sdp,* *b,* *n*) iqReadTimeout(&(sdp)->iqueue, b, n, TIME_IMMEDIATE)

Direct non-blocking read from a `SerialDriver`.

Note

This function bypasses the indirect access to the channel and reads directly from the input queue. This is faster but cannot be used to read from different channels implementations.

See also

chnReadTimeout()

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.34.3.20 #define PLATFORM_SERIAL_USE_USART1 FALSE

USART1 driver enable switch.

If set to `TRUE` the support for USART1 is included.

Note

The default is `FALSE`.

7.34.3.21 #define _serial_driver_data

Value:

```
_base_asynchronous_channel_data                                        \
  /* Driver state.*/                                                     \
  sdstate_t                 state;                                       \
  /* Input queue.*/                                                      \
  input_queue_t             iqueue;                                      \
  /* Output queue.*/                                                     \
  output_queue_t            oqueue;                                      \
  /* Input circular buffer.*/                                            \
  uint8_t                   ib[SERIAL_BUFFERS_SIZE];                     \
  /* Output circular buffer.*/                                           \
  uint8_t                   ob[SERIAL_BUFFERS_SIZE];                     \
  /* End of the mandatory fields.*/
```

`SerialDriver` specific data.

7.34.4 Typedef Documentation

7.34.4.1 typedef struct SerialDriver SerialDriver

Structure representing a serial driver.

7.34.5 Enumeration Type Documentation

7.34.5.1 enum sdstate_t

Driver state machine possible states.

Enumerator

SD_UNINIT Not initialized.

SD_STOP Stopped.

SD_READY Ready.

7.34.6 Function Documentation

7.34.6.1 void sdInit (void)

Serial Driver initialization.

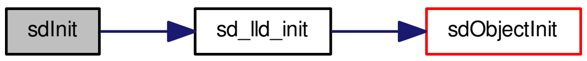
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



7.34.6.2 void sdObjectInit (SerialDriver ∗ *sdp*, qnotify_t *inotify*, qnotify_t *onotify*)

Initializes a generic full duplex driver object.

The HW dependent part of the initialization has to be performed outside, usually in the hardware initialization code.

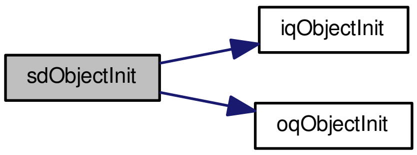
Parameters

`out`	*sdp*	pointer to a `SerialDriver` structure
`in`	*inotify*	pointer to a callback function that is invoked when some data is read from the Queue. The value can be `NULL`.
`in`	*onotify*	pointer to a callback function that is invoked when some data is written in the Queue. The value can be `NULL`.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



7.34.6.3 void sdStart (SerialDriver ∗ *sdp*, const SerialConfig ∗ *config*)

Configures and starts the driver.

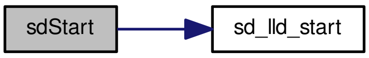
Parameters

`in`	*sdp*	pointer to a `SerialDriver` object
`in`	*config*	the architecture-dependent serial driver configuration. If this parameter is set to `NULL` then a default configuration is used.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.34.6.4 void sdStop (**SerialDriver** ∗ *sdp*)

Stops the driver.

Any thread waiting on the driver's queues will be awakened with the message `MSG_RESET`.

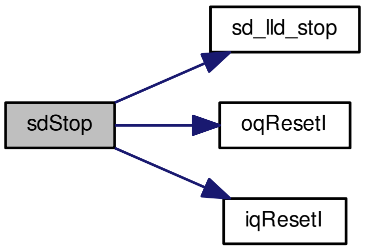
Parameters

`in`	*sdp*	pointer to a `SerialDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.34.6.5 void sdIncomingDataI (**SerialDriver** ∗ *sdp,* uint8_t *b*)

Handles incoming data.

This function must be called from the input interrupt service routine in order to enqueue incoming data and generate the related events.

Note

The incoming data event is only generated when the input queue becomes non-empty.
In order to gain some performance it is suggested to not use this function directly but copy this code directly into the interrupt service routine.

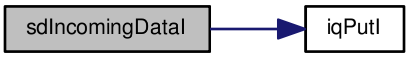
Parameters

`in`	*sdp*	pointer to a `SerialDriver` structure
`in`	*b*	the byte to be written in the driver's Input Queue

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



7.34.6.6 msg_t sdRequestDataI (**SerialDriver** ∗ *sdp*)

Handles outgoing data.

Must be called from the output interrupt service routine in order to get the next byte to be transmitted.

Note

In order to gain some performance it is suggested to not use this function directly but copy this code directly into the interrupt service routine.

Parameters

`in`	*sdp*	pointer to a `SerialDriver` structure

Returns

The byte value read from the driver's output queue.

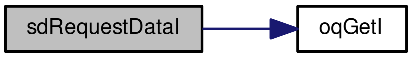
Return values

MSG_TIMEOUT	if the queue is empty (the lower driver usually disables the interrupt source when this happens).

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



7.34.6.7 bool sdPutWouldBlock (**SerialDriver** ∗ *sdp*)

Direct output check on a SerialDriver.

Note

This function bypasses the indirect access to the channel and checks directly the output queue. This is faster but cannot be used to check different channels implementations.

Parameters

in	*sdp*	pointer to a SerialDriver structure

Returns

The queue status.

Return values

false	if the next write operation would not block.
true	if the next write operation would block.

Deprecated

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.34.6.8 bool sdGetWouldBlock (**SerialDriver** ∗ *sdp*)

Direct input check on a SerialDriver.

Note

This function bypasses the indirect access to the channel and checks directly the input queue. This is faster but cannot be used to check different channels implementations.

Parameters

in	*sdp*	pointer to a `SerialDriver` structure

Returns

The queue status.

Return values

false	if the next write operation would not block.
true	if the next write operation would block.

Deprecated

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

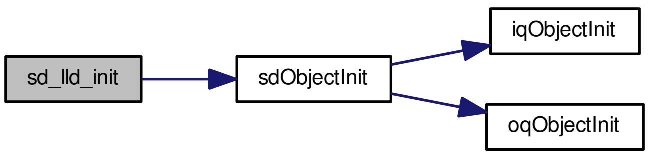
7.34.6.9 void sd_lld_init (void)

Low level serial driver initialization.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.34.6.10 void sd_lld_start (SerialDriver ∗ *sdp*, const SerialConfig ∗ *config*)

Low level serial driver configuration and (re)start.

Parameters

in	*sdp*	pointer to a `SerialDriver` object
in	*config*	the architecture-dependent serial driver configuration. If this parameter is set to `NULL` then a default configuration is used.

Function Class:

Not an API, this function is for internal use only.

7.34.6.11 void sd_lld_stop (SerialDriver ∗ *sdp*)

Low level serial driver stop.

De-initializes the USART, stops the associated clock, resets the interrupt vector.

Parameters

in	*sdp*	pointer to a `SerialDriver` object

Function Class:

Not an API, this function is for internal use only.

7.34.7 Variable Documentation

7.34.7.1 SerialDriver SD1

USART1 serial driver identifier.

7.34.7.2 const SerialConfig default_config [static]

Initial value:

```
= {
  38400
}
```

Driver default configuration.

7.35 Serial over USB Driver

Serial over USB Driver.

7.35.1 Detailed Description

Serial over USB Driver.

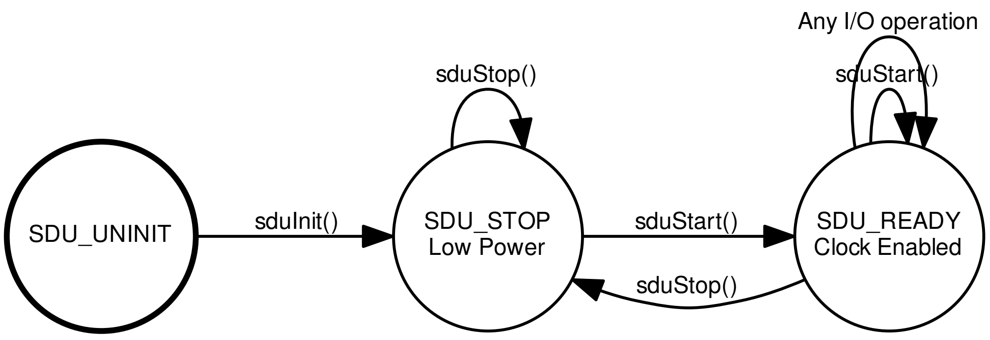
This module implements an USB Communication Device Class (CDC) as a normal serial communication port accessible from the device application.

Precondition

In order to use the USB over Serial driver the `HAL_USE_SERIAL_USB` option must be enabled in `halconf.h`.

7.35.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



Macros

- #define _serial_usb_driver_data

    `SerialDriver` specific data.
- #define _serial_usb_driver_methods _base_asynchronous_channel_methods

    `SerialUSBDriver` specific methods.

SERIAL_USB configuration options

- #define SERIAL_USB_BUFFERS_SIZE 256

    Serial over USB buffers size.

- #define SERIAL_USB_BUFFERS_NUMBER 2

  Serial over USB number of buffers.

Typedefs

- typedef struct SerialUSBDriver SerialUSBDriver

  Structure representing a serial over USB driver.

Data Structures

- struct SerialUSBConfig

  Serial over USB Driver configuration structure.
- struct SerialUSBDriverVMT

  `SerialDriver` *virtual methods table.*
- struct SerialUSBDriver

  Full duplex serial driver class.

Functions

- static void ibnotify (io_buffers_queue_t ∗bqp)

  Notification of empty buffer released into the input buffers queue.
- static void obnotify (io_buffers_queue_t ∗bqp)

  Notification of filled buffer inserted into the output buffers queue.
- void sduInit (void)

  Serial Driver initialization.
- void sduObjectInit (SerialUSBDriver ∗sdup)

  Initializes a generic full duplex driver object.
- void sduStart (SerialUSBDriver ∗sdup, const SerialUSBConfig ∗config)

  Configures and starts the driver.
- void sduStop (SerialUSBDriver ∗sdup)

  Stops the driver.
- void sduSuspendHookI (SerialUSBDriver ∗sdup)

  USB device suspend handler.
- void sduWakeupHookI (SerialUSBDriver ∗sdup)

  USB device wakeup handler.
- void sduConfigureHookI (SerialUSBDriver ∗sdup)

  USB device configured handler.
- bool sduRequestsHook (USBDriver ∗usbp)

  Default requests hook.
- void sduSOFHookI (SerialUSBDriver ∗sdup)

  SOF handler.
- void sduDataTransmitted (USBDriver ∗usbp, usbep_t ep)

  Default data transmitted callback.
- void sduDataReceived (USBDriver ∗usbp, usbep_t ep)

  Default data received callback.
- void sduInterruptTransmitted (USBDriver ∗usbp, usbep_t ep)

  Default data received callback.

Enumerations

7.35.3 Macro Definition Documentation

7.35.3.1 #define SERIAL_USB_BUFFERS_SIZE 256

Serial over USB buffers size.

Configuration parameter, the buffer size must be a multiple of the USB data endpoint maximum packet size.

Note

The default is 256 bytes for both the transmission and receive buffers.

7.35.3.2 #define SERIAL_USB_BUFFERS_NUMBER 2

Serial over USB number of buffers.

Note

The default is 2 buffers.

7.35.3.3 #define _serial_usb_driver_data

Value:

```
_base_asynchronous_channel_data                                            \
  /* Driver state.*/                                                         \
  sdustate_t                state;                                           \
  /* Input buffers queue.*/                                                  \
  input_buffers_queue_t     ibqueue;                                         \
  /* Output queue.*/                                                         \
  output_buffers_queue_t    obqueue;                                         \
  /* Input buffer.*/                                                         \
  uint8_t                   ib[BQ_BUFFER_SIZE(
      SERIAL_USB_BUFFERS_NUMBER,     \
                                              SERIAL_USB_BUFFERS_SIZE)];     \
  /* Output buffer.*/                                                        \
  uint8_t                   ob[BQ_BUFFER_SIZE(
      SERIAL_USB_BUFFERS_NUMBER,     \
                                              SERIAL_USB_BUFFERS_SIZE)];     \
  /* End of the mandatory fields.*/                                          \
  /* Current configuration data.*/                                           \
  const SerialUSBConfig     *config;
```

SerialDriver specific data.

7.35.3.4 #define _serial_usb_driver_methods _base_asynchronous_channel_methods

SerialUSBDriver specific methods.

7.35.4 Typedef Documentation

7.35.4.1 typedef struct SerialUSBDriver SerialUSBDriver

Structure representing a serial over USB driver.

7.35.5 Enumeration Type Documentation

7.35.5.1 enum sdustate_t

Driver state machine possible states.

Enumerator

SDU_UNINIT	Not initialized.
SDU_STOP	Stopped.
SDU_READY	Ready.

7.35.6 Function Documentation

7.35.6.1 static void ibnotify (io_buffers_queue_t * *bqp*) [static]

Notification of empty buffer released into the input buffers queue.

Parameters

in	*bqp*	the buffers queue pointer.

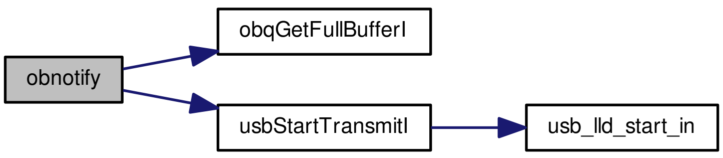
7.35.6.2 static void obnotify (io_buffers_queue_t * *bqp*) [static]

Notification of filled buffer inserted into the output buffers queue.

Parameters

in	*bqp*	the buffers queue pointer.

Here is the call graph for this function:



7.35.6.3 void sduInit (void)

Serial Driver initialization.

Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.35.6.4 void sduObjectInit (SerialUSBDriver ∗ *sdup*)

Initializes a generic full duplex driver object.

The HW dependent part of the initialization has to be performed outside, usually in the hardware initialization code.

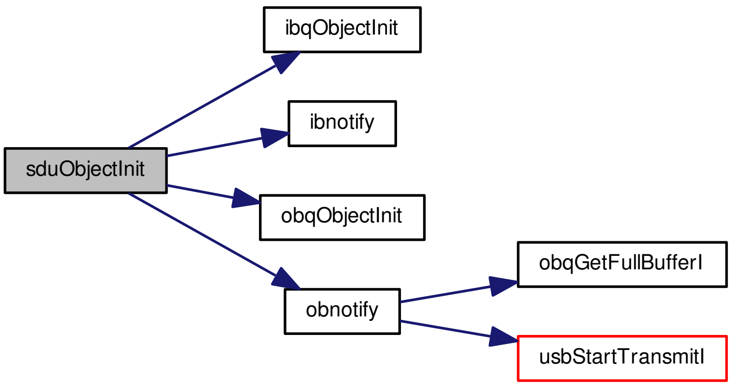
Parameters

`out`	*sdup*	pointer to a `SerialUSBDriver` structure

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



7.35.6.5 void sduStart (SerialUSBDriver ∗ *sdup*, const SerialUSBConfig ∗ *config*)

Configures and starts the driver.

Parameters

`in`	*sdup*	pointer to a `SerialUSBDriver` object
`in`	*config*	the serial over USB driver configuration

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.35.6.6 void sduStop (**SerialUSBDriver** ∗ *sdup*)

Stops the driver.

Any thread waiting on the driver's queues will be awakened with the message `MSG_RESET`.

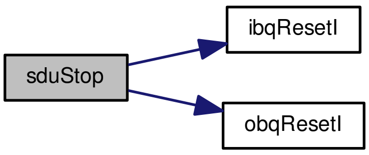
Parameters

in	*sdup*	pointer to a `SerialUSBDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.35.6.7 void sduSuspendHookI (**SerialUSBDriver** ∗ *sdup*)

USB device suspend handler.

Generates a `CHN_DISCONNECT` event and puts queues in non-blocking mode, this way the application cannot get stuck in the middle of an I/O operations.

Note

If this function is not called from an ISR then an explicit call to `osalOsRescheduleS()` in necessary afterward.

Parameters

in	*sdup*	pointer to a `SerialUSBDriver` object

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.35.6.8 void sduWakeupHookI (SerialUSBDriver ∗ *sdup*)

USB device wakeup handler.

Generates a `CHN_CONNECT` event and resumes normal queues operations.

Note

If this function is not called from an ISR then an explicit call to `osalOsRescheduleS()` in necessary afterward.

Parameters

`in`	*sdup*	pointer to a `SerialUSBDriver` object

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.35.6.9 void sduConfigureHookI (SerialUSBDriver ∗ *sdup*)

USB device configured handler.

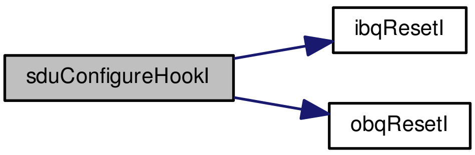
Parameters

`in`	*sdup*	pointer to a `SerialUSBDriver` object

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



7.35.6.10 bool sduRequestsHook (USBDriver ∗ *usbp*)

Default requests hook.

Applications wanting to use the Serial over USB driver can use this function as requests hook in the USB configuration. The following requests are emulated:

- CDC_GET_LINE_CODING.
- CDC_SET_LINE_CODING.
- CDC_SET_CONTROL_LINE_STATE.

Parameters

in	*usbp*	pointer to the `USBDriver` object

Returns

The hook status.

Return values

true	Message handled internally.
false	Message not handled.

7.35.6.11 void sduSOFHookI (SerialUSBDriver ∗ *sdup*)

SOF handler.

The SOF interrupt is used for automatic flushing of incomplete buffers pending in the output queue.

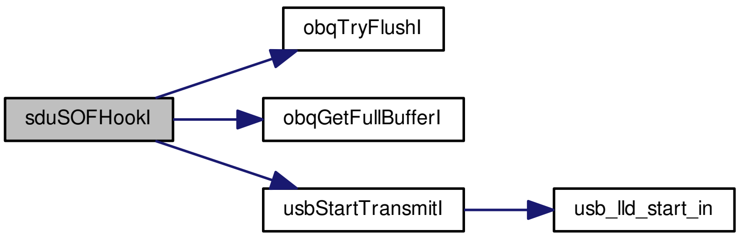
Parameters

in	*sdup*	pointer to a `SerialUSBDriver` object

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



7.35.6.12 void sduDataTransmitted (USBDriver ∗ *usbp*, usbep_t *ep*)

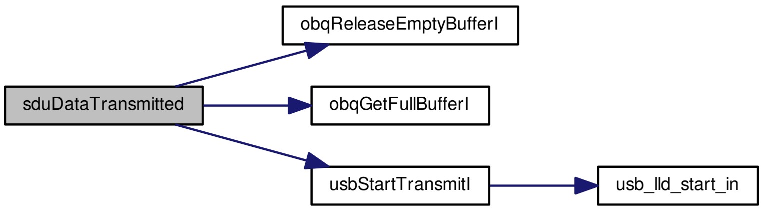
Default data transmitted callback.

The application must use this function as callback for the IN data endpoint.

Parameters

in	*usbp*	pointer to the `USBDriver` object
in	*ep*	IN endpoint number

Here is the call graph for this function:



7.35.6.13 void sduDataReceived (USBDriver ∗ *usbp*, usbep_t *ep*)

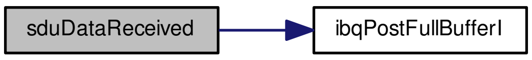
Default data received callback.

The application must use this function as callback for the OUT data endpoint.

Parameters

in	*usbp*	pointer to the `USBDriver` object
in	*ep*	OUT endpoint number

Here is the call graph for this function:



7.35.6.14 void sduInterruptTransmitted (USBDriver ∗ *usbp*, usbep_t *ep*)

Default data received callback.

The application must use this function as callback for the IN interrupt endpoint.

Parameters

in	*usbp*	pointer to the `USBDriver` object
in	*ep*	endpoint number

7.36 SPI Driver

Generic SPI Driver.

7.36.1 Detailed Description

Generic SPI Driver.

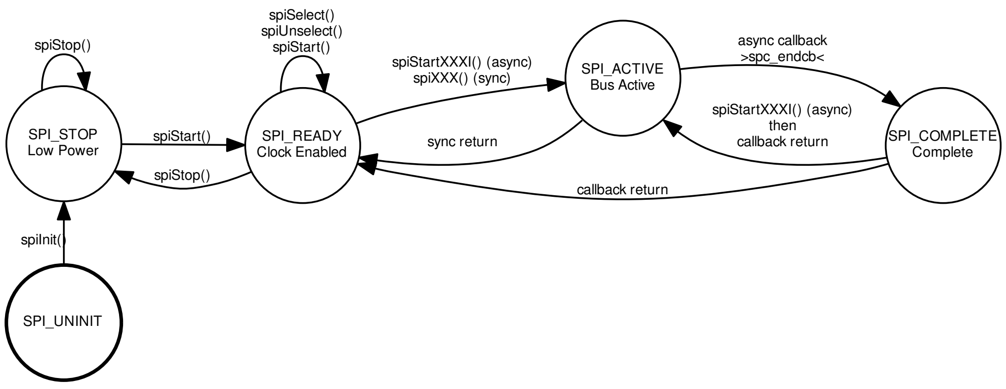
This module implements a generic SPI (Serial Peripheral Interface) driver allowing bidirectional and monodirectional transfers, complex atomic transactions are supported as well.

Precondition

In order to use the SPI driver the `HAL_USE_SPI` option must be enabled in `halconf.h`.

7.36.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



The driver is not thread safe for performance reasons, if you need to access the SPI bus from multiple threads then use the `spiAcquireBus()` and `spiReleaseBus()` APIs in order to gain exclusive access.

SPI configuration options

- #define SPI_USE_WAIT TRUE

  Enables synchronous APIs.
- #define SPI_USE_MUTUAL_EXCLUSION TRUE

  Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.

Macro Functions

- #define spiSelectI(spip)

Asserts the slave select signal and prepares for transfers.
- #define spiUnselectI(spip)

  Deasserts the slave select signal.
- #define spiStartIgnoreI(spip, n)

  Ignores data on the SPI bus.
- #define spiStartExchangeI(spip, n, txbuf, rxbuf)

  Exchanges data on the SPI bus.
- #define spiStartSendI(spip, n, txbuf)

  Sends data over the SPI bus.
- #define spiStartReceiveI(spip, n, rxbuf)

  Receives data from the SPI bus.
- #define spiPolledExchange(spip, frame) spi_lld_polled_exchange(spip, frame)

  Exchanges one frame using a polled wait.

Low level driver helper macros

- #define _spi_wakeup_isr(spip)

  Wakes up the waiting thread.
- #define _spi_isr_code(spip)

  Common ISR code.

PLATFORM configuration options

- #define PLATFORM_SPI_USE_SPI1 FALSE

  SPI1 driver enable switch.

Typedefs

- typedef struct SPIDriver SPIDriver

  Type of a structure representing an SPI driver.
- typedef void(∗ spicallback_t) (SPIDriver ∗spip)

  SPI notification callback type.

Data Structures

- struct SPIConfig

  Driver configuration structure.
- struct SPIDriver

  Structure representing an SPI driver.

Functions

- void spiInit (void)

  SPI Driver initialization.
- void spiObjectInit (SPIDriver ∗spip)

  Initializes the standard part of a `SPIDriver` structure.
- void spiStart (SPIDriver ∗spip, const SPIConfig ∗config)

  Configures and activates the SPI peripheral.
- void spiStop (SPIDriver ∗spip)

Deactivates the SPI peripheral.
- void spiSelect (SPIDriver ∗spip)

  Asserts the slave select signal and prepares for transfers.
- void spiUnselect (SPIDriver ∗spip)

  Deasserts the slave select signal.
- void spiStartIgnore (SPIDriver ∗spip, size_t n)

  Ignores data on the SPI bus.
- void spiStartExchange (SPIDriver ∗spip, size_t n, const void ∗txbuf, void ∗rxbuf)

  Exchanges data on the SPI bus.
- void spiStartSend (SPIDriver ∗spip, size_t n, const void ∗txbuf)

  Sends data over the SPI bus.
- void spiStartReceive (SPIDriver ∗spip, size_t n, void ∗rxbuf)

  Receives data from the SPI bus.
- void spiIgnore (SPIDriver ∗spip, size_t n)

  Ignores data on the SPI bus.
- void spiExchange (SPIDriver ∗spip, size_t n, const void ∗txbuf, void ∗rxbuf)

  Exchanges data on the SPI bus.
- void spiSend (SPIDriver ∗spip, size_t n, const void ∗txbuf)

  Sends data over the SPI bus.
- void spiReceive (SPIDriver ∗spip, size_t n, void ∗rxbuf)

  Receives data from the SPI bus.
- void spiAcquireBus (SPIDriver ∗spip)

  Gains exclusive access to the SPI bus.
- void spiReleaseBus (SPIDriver ∗spip)

  Releases exclusive access to the SPI bus.
- void spi_lld_init (void)

  Low level SPI driver initialization.
- void spi_lld_start (SPIDriver ∗spip)

  Configures and activates the SPI peripheral.
- void spi_lld_stop (SPIDriver ∗spip)

  Deactivates the SPI peripheral.
- void spi_lld_select (SPIDriver ∗spip)

  Asserts the slave select signal and prepares for transfers.
- void spi_lld_unselect (SPIDriver ∗spip)

  Deasserts the slave select signal.
- void spi_lld_ignore (SPIDriver ∗spip, size_t n)

  Ignores data on the SPI bus.
- void spi_lld_exchange (SPIDriver ∗spip, size_t n, const void ∗txbuf, void ∗rxbuf)

  Exchanges data on the SPI bus.
- void spi_lld_send (SPIDriver ∗spip, size_t n, const void ∗txbuf)

  Sends data over the SPI bus.
- void spi_lld_receive (SPIDriver ∗spip, size_t n, void ∗rxbuf)

  Receives data from the SPI bus.
- uint16_t spi_lld_polled_exchange (SPIDriver ∗spip, uint16_t frame)

  Exchanges one frame using a polled wait.

Enumerations

Variables

- SPIDriver SPID1

  SPI1 driver identifier.

7.36.3 Macro Definition Documentation

7.36.3.1 #define SPI_USE_WAIT TRUE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.36.3.2 #define SPI_USE_MUTUAL_EXCLUSION TRUE

Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.

Note

Disabling this option saves both code and data space.

7.36.3.3 #define spiSelectI(*spip*)

Value:

```
{                                                                         \
  spi_lld_select(spip);                                                   \
}
```

Asserts the slave select signal and prepares for transfers.

Parameters

in	*spip*	pointer to the `SPIDriver` object

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.36.3.4 #define spiUnselectI(*spip*)

Value:

```
{                                                                         \
  spi_lld_unselect(spip);                                                 \
}
```

Deasserts the slave select signal.

The previously selected peripheral is unselected.

Parameters

in	*spip*	pointer to the `SPIDriver` object

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.36.3.5 #define spiStartIgnoreI(*spip*, *n*)

Value:

```
{                                                    \
  (spip)->state = SPI_ACTIVE;
       \
  spi_lld_ignore(spip, n);                                                          \
}
```

Ignores data on the SPI bus.

This asynchronous function starts the transmission of a series of idle words on the SPI bus and ignores the received data.

Precondition

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	*spip*	pointer to the `SPIDriver` object
in	*n*	number of words to be ignored

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.36.3.6 #define spiStartExchangeI(*spip*, *n*, *txbuf*, *rxbuf*)

Value:

```
{                                  \
  (spip)->state = SPI_ACTIVE;
       \
  spi_lld_exchange(spip, n, txbuf, rxbuf);                                          \
}
```

Exchanges data on the SPI bus.

This asynchronous function starts a simultaneous transmit/receive operation.

Precondition

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

Postcondition

At the end of the operation the configured callback is invoked.

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

Parameters

in	*spip*	pointer to the `SPIDriver` object
in	*n*	number of words to be exchanged
in	*txbuf*	the pointer to the transmit buffer
out	*rxbuf*	the pointer to the receive buffer

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.36.3.7 #define spiStartSendI(*spip, n, txbuf*)

Value:

```
{                                              \
  (spip)->state = SPI_ACTIVE;
      \
  spi_lld_send(spip, n, txbuf);                                              \
}
```

Sends data over the SPI bus.

This asynchronous function starts a transmit operation.

Precondition

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

Postcondition

At the end of the operation the configured callback is invoked.

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

Parameters

in	*spip*	pointer to the `SPIDriver` object
in	*n*	number of words to send
in	*txbuf*	the pointer to the transmit buffer

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.36.3.8 #define spiStartReceiveI(*spip,* *n,* *rxbuf*)

Value:

```
{                                                     \
  (spip)->state = SPI_ACTIVE;
      \
  spi_lld_receive(spip, n, rxbuf);                                \
}
```

Receives data from the SPI bus.

This asynchronous function starts a receive operation.

Precondition

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

Postcondition

At the end of the operation the configured callback is invoked.

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

Parameters

`in`	*spip*	pointer to the `SPIDriver` object
`in`	*n*	number of words to receive
`out`	*rxbuf*	the pointer to the receive buffer

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.36.3.9 #define spiPolledExchange(*spip,* *frame*) spi_lld_polled_exchange(spip, frame)

Exchanges one frame using a polled wait.

This synchronous function exchanges one frame using a polled synchronization method. This function is useful when exchanging small amount of data on high speed channels, usually in this situation is much more efficient just wait for completion using polling than suspending the thread waiting for an interrupt.

Note

This API is implemented as a macro in order to minimize latency.

Parameters

in	*spip*	pointer to the SPIDriver object
in	*frame*	the data frame to send over the SPI bus

Returns

The received data frame from the SPI bus.

7.36.3.10 #define _spi_wakeup_isr(*spip*)

Value:

```
{                                        \
  osalSysLockFromISR();                                                    \
  osalThreadResumeI(&(spip)->thread, MSG_OK);                              \
  osalSysUnlockFromISR();                                                  \
}
```

Wakes up the waiting thread.

Parameters

in	*spip*	pointer to the SPIDriver object

Function Class:

Not an API, this function is for internal use only.

7.36.3.11 #define _spi_isr_code(*spip*)

Value:

```
{                                          \
  if ((spip)->config->end_cb) {                                            \
    (spip)->state = SPI_COMPLETE;                                          \
    (spip)->config->end_cb(spip);                                          \
    if ((spip)->state == SPI_COMPLETE)                                     \
      (spip)->state = SPI_READY;                                           \
  }                                                                        \
  else                                                                     \
    (spip)->state = SPI_READY;
      \
  _spi_wakeup_isr(spip);                                                   \
}
```

Common ISR code.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

`in`	*spip*	pointer to the `SPIDriver` object

Function Class:

Not an API, this function is for internal use only.

7.36.3.12 #define PLATFORM_SPI_USE_SPI1 FALSE

SPI1 driver enable switch.

If set to `TRUE` the support for SPI1 is included.

Note

The default is `FALSE`.

7.36.4 Typedef Documentation

7.36.4.1 typedef struct SPIDriver SPIDriver

Type of a structure representing an SPI driver.

7.36.4.2 typedef void(∗ spicallback_t) (SPIDriver ∗spip)

SPI notification callback type.

Parameters

`in`	*spip*	pointer to the `SPIDriver` object triggering the callback

7.36.5 Enumeration Type Documentation

7.36.5.1 enum spistate_t

Driver state machine possible states.

Enumerator

SPI_UNINIT	Not initialized.
SPI_STOP	Stopped.
SPI_READY	Ready.
SPI_ACTIVE	Exchanging data.
SPI_COMPLETE	Asynchronous operation complete.

7.36.6 Function Documentation

7.36.6.1 void spiInit (void)

SPI Driver initialization.

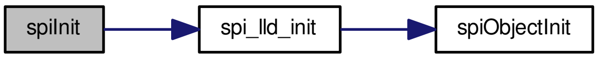
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



7.36.6.2 void spiObjectInit (SPIDriver ∗ *spip*)

Initializes the standard part of a `SPIDriver` structure.

Parameters

`out`	*spip*	pointer to the `SPIDriver` object

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.36.6.3 void spiStart (SPIDriver ∗ *spip*, const SPIConfig ∗ *config*)

Configures and activates the SPI peripheral.

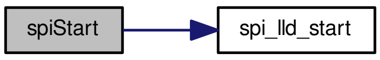
Parameters

`in`	*spip*	pointer to the `SPIDriver` object
`in`	*config*	pointer to the `SPIConfig` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.36.6.4 void spiStop (**SPIDriver** * *spip*)

Deactivates the SPI peripheral.

Note

Deactivating the peripheral also enforces a release of the slave select line.

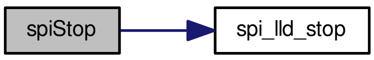
Parameters

in	*spip*	pointer to the `SPIDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.36.6.5 void spiSelect (**SPIDriver** * *spip*)

Asserts the slave select signal and prepares for transfers.

Parameters

in	*spip*	pointer to the `SPIDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.36.6.6 void spiUnselect (SPIDriver ∗ *spip*)

Deasserts the slave select signal.

The previously selected peripheral is unselected.

Parameters

in	*spip*	pointer to the `SPIDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.36.6.7 void spiStartIgnore (SPIDriver ∗ *spip*, size_t *n*)

Ignores data on the SPI bus.

This asynchronous function starts the transmission of a series of idle words on the SPI bus and ignores the received data.

Precondition

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	*spip*	pointer to the `SPIDriver` object
in	*n*	number of words to be ignored

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.36.6.8 void spiStartExchange (SPIDriver ∗ *spip*, size_t *n*, const void ∗ *txbuf*, void ∗ *rxbuf*)

Exchanges data on the SPI bus.

This asynchronous function starts a simultaneous transmit/receive operation.

Precondition

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

Postcondition

At the end of the operation the configured callback is invoked.

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

Parameters

`in`	*spip*	pointer to the `SPIDriver` object
`in`	*n*	number of words to be exchanged
`in`	*txbuf*	the pointer to the transmit buffer
`out`	*rxbuf*	the pointer to the receive buffer

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.36.6.9 void spiStartSend (SPIDriver ∗ *spip*, size_t *n*, const void ∗ *txbuf*)

Sends data over the SPI bus.

This asynchronous function starts a transmit operation.

Precondition

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

Postcondition

At the end of the operation the configured callback is invoked.

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

Parameters

`in`	*spip*	pointer to the `SPIDriver` object
`in`	*n*	number of words to send
`in`	*txbuf*	the pointer to the transmit buffer

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.36.6.10 void spiStartReceive (SPIDriver ∗ *spip*, size_t *n*, void ∗ *rxbuf*)

Receives data from the SPI bus.

This asynchronous function starts a receive operation.

Precondition

A slave must have been selected using `spiSelect()` or `spiSelectI()`.

Postcondition

At the end of the operation the configured callback is invoked.

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

Parameters

`in`	*spip*	pointer to the `SPIDriver` object
`in`	*n*	number of words to receive
`out`	*rxbuf*	the pointer to the receive buffer

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.36.6.11 void spiIgnore (SPIDriver ∗ *spip*, size_t *n*)

Ignores data on the SPI bus.

This synchronous function performs the transmission of a series of idle words on the SPI bus and ignores the received data.

Precondition

In order to use this function the option `SPI_USE_WAIT` must be enabled.
In order to use this function the driver must have been configured without callbacks (`end_cb` = `NULL`).

Parameters

`in`	*spip*	pointer to the `SPIDriver` object
`in`	*n*	number of words to be ignored

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.36.6.12 void spiExchange (SPIDriver ∗ *spip*, size_t *n*, const void ∗ *txbuf*, void ∗ *rxbuf*)

Exchanges data on the SPI bus.

This synchronous function performs a simultaneous transmit/receive operation.

Precondition

In order to use this function the option `SPI_USE_WAIT` must be enabled.
In order to use this function the driver must have been configured without callbacks (`end_cb` = `NULL`).

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

Parameters

in	*spip*	pointer to the SPIDriver object
in	*n*	number of words to be exchanged
in	*txbuf*	the pointer to the transmit buffer
out	*rxbuf*	the pointer to the receive buffer

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.36.6.13 void spiSend (SPIDriver * *spip*, size_t *n*, const void * *txbuf*)

Sends data over the SPI bus.

This synchronous function performs a transmit operation.

Precondition

In order to use this function the option `SPI_USE_WAIT` must be enabled.
In order to use this function the driver must have been configured without callbacks (`end_cb` = `NULL`).

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

Parameters

in	*spip*	pointer to the SPIDriver object
in	*n*	number of words to send
in	*txbuf*	the pointer to the transmit buffer

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.36.6.14 void spiReceive (SPIDriver * *spip*, size_t *n*, void * *rxbuf*)

Receives data from the SPI bus.

This synchronous function performs a receive operation.

Precondition

In order to use this function the option `SPI_USE_WAIT` must be enabled.
In order to use this function the driver must have been configured without callbacks (`end_cb` = `NULL`).

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

Parameters

in	*spip*	pointer to the `SPIDriver` object
in	*n*	number of words to receive
out	*rxbuf*	the pointer to the receive buffer

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.36.6.15 void spiAcquireBus (SPIDriver ∗ *spip*)

Gains exclusive access to the SPI bus.

This function tries to gain ownership to the SPI bus, if the bus is already being used then the invoking thread is queued.

Precondition

In order to use this function the option `SPI_USE_MUTUAL_EXCLUSION` must be enabled.

Parameters

in	*spip*	pointer to the `SPIDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.36.6.16 void spiReleaseBus (SPIDriver ∗ *spip*)

Releases exclusive access to the SPI bus.

Precondition

In order to use this function the option `SPI_USE_MUTUAL_EXCLUSION` must be enabled.

Parameters

in	*spip*	pointer to the `SPIDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

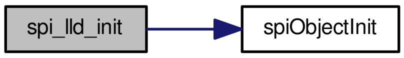
7.36.6.17 void spi_lld_init (void)

Low level SPI driver initialization.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.36.6.18 void spi_lld_start (SPIDriver * *spip*)

Configures and activates the SPI peripheral.

Parameters

in	*spip*	pointer to the SPIDriver object

Function Class:

Not an API, this function is for internal use only.

7.36.6.19 void spi_lld_stop (SPIDriver * *spip*)

Deactivates the SPI peripheral.

Parameters

in	*spip*	pointer to the SPIDriver object

Function Class:

Not an API, this function is for internal use only.

7.36.6.20 void spi_lld_select (SPIDriver * *spip*)

Asserts the slave select signal and prepares for transfers.

Parameters

in	*spip*	pointer to the SPIDriver object

Function Class:

Not an API, this function is for internal use only.

7.36.6.21 void spi_lld_unselect (SPIDriver ∗ *spip*)

Deasserts the slave select signal.

The previously selected peripheral is unselected.

Parameters

in	*spip*	pointer to the SPIDriver object

Function Class:

Not an API, this function is for internal use only.

7.36.6.22 void spi_lld_ignore (SPIDriver ∗ *spip*, size_t *n*)

Ignores data on the SPI bus.

This asynchronous function starts the transmission of a series of idle words on the SPI bus and ignores the received data.

Postcondition

At the end of the operation the configured callback is invoked.

Parameters

in	*spip*	pointer to the SPIDriver object
in	*n*	number of words to be ignored

Function Class:

Not an API, this function is for internal use only.

7.36.6.23 void spi_lld_exchange (SPIDriver ∗ *spip*, size_t *n*, const void ∗ *txbuf*, void ∗ *rxbuf*)

Exchanges data on the SPI bus.

This asynchronous function starts a simultaneous transmit/receive operation.

Postcondition

At the end of the operation the configured callback is invoked.

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

Parameters

in	*spip*	pointer to the SPIDriver object
in	*n*	number of words to be exchanged
in	*txbuf*	the pointer to the transmit buffer
out	*rxbuf*	the pointer to the receive buffer

Function Class:

Not an API, this function is for internal use only.

7.36.6.24 void spi_lld_send (SPIDriver ∗ *spip*, size_t *n*, const void ∗ *txbuf*)

Sends data over the SPI bus.

This asynchronous function starts a transmit operation.

Postcondition

At the end of the operation the configured callback is invoked.

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

Parameters

`in`	*spip*	pointer to the `SPIDriver` object
`in`	*n*	number of words to send
`in`	*txbuf*	the pointer to the transmit buffer

Function Class:

Not an API, this function is for internal use only.

7.36.6.25 void spi_lld_receive (SPIDriver ∗ *spip*, size_t *n*, void ∗ *rxbuf*)

Receives data from the SPI bus.

This asynchronous function starts a receive operation.

Postcondition

At the end of the operation the configured callback is invoked.

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

Parameters

`in`	*spip*	pointer to the `SPIDriver` object
`in`	*n*	number of words to receive
`out`	*rxbuf*	the pointer to the receive buffer

Function Class:

Not an API, this function is for internal use only.

7.36.6.26 uint16_t spi_lld_polled_exchange (SPIDriver * *spip*, uint16_t *frame*)

Exchanges one frame using a polled wait.

This synchronous function exchanges one frame using a polled synchronization method. This function is useful when exchanging small amount of data on high speed channels, usually in this situation is much more efficient just wait for completion using polling than suspending the thread waiting for an interrupt.

Parameters

`in`	*spip*	pointer to the `SPIDriver` object
`in`	*frame*	the data frame to send over the SPI bus

Returns

The received data frame from the SPI bus.

7.36.7 Variable Documentation

7.36.7.1 SPIDriver SPID1

SPI1 driver identifier.

7.37 ST Driver

Generic System Tick Driver.

7.37.1 Detailed Description

Generic System Tick Driver.

This module implements a system tick timer in order to support the underlying operating system.

Macro Functions

- #define stGetCounter() st_lld_get_counter()

  Returns the time counter value.
- #define stIsAlarmActive() st_lld_is_alarm_active()

  Determines if the alarm is active.

Functions

- void stInit (void)

  ST Driver initialization.
- void stStartAlarm (systime_t abstime)

  Starts the alarm.
- void stStopAlarm (void)

  Stops the alarm interrupt.
- void stSetAlarm (systime_t abstime)

  Sets the alarm time.
- systime_t stGetAlarm (void)

  Returns the current alarm time.
- void st_lld_init (void)

  Low level ST driver initialization.
- static systime_t st_lld_get_counter (void)

  Returns the time counter value.
- static void st_lld_start_alarm (systime_t abstime)

  Starts the alarm.
- static void st_lld_stop_alarm (void)

  Stops the alarm interrupt.
- static void st_lld_set_alarm (systime_t abstime)

  Sets the alarm time.
- static systime_t st_lld_get_alarm (void)

  Returns the current alarm time.
- static bool st_lld_is_alarm_active (void)

  Determines if the alarm is active.

7.37.2 Macro Definition Documentation

7.37.2.1 #define stGetCounter( ) st_lld_get_counter()

Returns the time counter value.

Note

This functionality is only available in free running mode, the behaviour in periodic mode is undefined.

Returns

The counter value.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.37.2.2 #define stIsAlarmActive( ) st_lld_is_alarm_active()

Determines if the alarm is active.

Returns

The alarm status.

Return values

false	if the alarm is not active.
true	is the alarm is active

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.37.3 Function Documentation

7.37.3.1 void stInit (void)

ST Driver initialization.

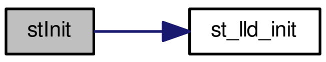
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



7.37.3.2 void stStartAlarm (systime_t *abstime*)

Starts the alarm.

Note

Makes sure that no spurious alarms are triggered after this call.
This functionality is only available in free running mode, the behavior in periodic mode is undefined.

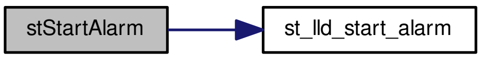
Parameters

`in`	*abstime*	the time to be set for the first alarm

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.37.3.3 void stStopAlarm (void)

Stops the alarm interrupt.

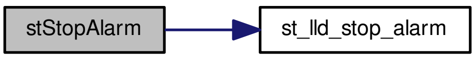
Note

This functionality is only available in free running mode, the behavior in periodic mode is undefined.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.37.3.4 void stSetAlarm (systime_t *abstime*)

Sets the alarm time.

Note

This functionality is only available in free running mode, the behavior in periodic mode is undefined.

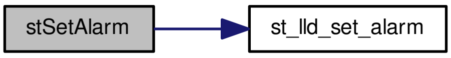
Parameters

`in`	*abstime*	the time to be set for the next alarm

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.37.3.5 systime_t stGetAlarm (void)

Returns the current alarm time.

Note

This functionality is only available in free running mode, the behavior in periodic mode is undefined.

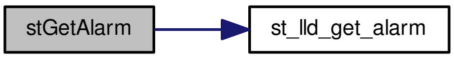
Returns

The currently set alarm time.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.37.3.6 void st_lld_init (void)

Low level ST driver initialization.

Function Class:

Not an API, this function is for internal use only.

7.37.3.7 static systime_t st_lld_get_counter (void) [inline], [static]

Returns the time counter value.

Returns

The counter value.

Function Class:

Not an API, this function is for internal use only.

7.37.3.8 static void st_lld_start_alarm (systime_t *abstime*) [inline], [static]

Starts the alarm.

Note

Makes sure that no spurious alarms are triggered after this call.

Parameters

in	*abstime*	the time to be set for the first alarm

Function Class:

Not an API, this function is for internal use only.

7.37.3.9 static void st_lld_stop_alarm (void) [inline], [static]

Stops the alarm interrupt.

Function Class:

Not an API, this function is for internal use only.

7.37.3.10 static void st_lld_set_alarm (systime_t *abstime*) [inline], [static]

Sets the alarm time.

Parameters

in	*abstime*	the time to be set for the next alarm

Function Class:

Not an API, this function is for internal use only.

7.37.3.11 static systime_t st_lld_get_alarm (void) [inline], [static]

Returns the current alarm time.

Returns

The currently set alarm time.

Function Class:

Not an API, this function is for internal use only.

7.37.3.12 static bool st_lld_is_alarm_active (void) [inline], [static]

Determines if the alarm is active.

Returns

The alarm status.

Return values

false	if the alarm is not active.
true	is the alarm is active

Function Class:

Not an API, this function is for internal use only.

7.38 UART Driver

Generic UART Driver.

7.38.1 Detailed Description

Generic UART Driver.

This driver abstracts a generic UART (Universal Asynchronous Receiver Transmitter) peripheral, the API is designed to be:

- Unbuffered and copy-less, transfers are always directly performed from/to the application-level buffers without extra copy operations.
- Asynchronous, the API is always non blocking.
- Callbacks capable, operations completion and other events are notified using callbacks.

Special hardware features like deep hardware buffers, DMA transfers are hidden to the user but fully supportable by the low level implementations.
This driver model is best used where communication events are meant to drive an higher level state machine, as example:

- RS485 drivers.
- Multipoint network drivers.
- Serial protocol decoders.

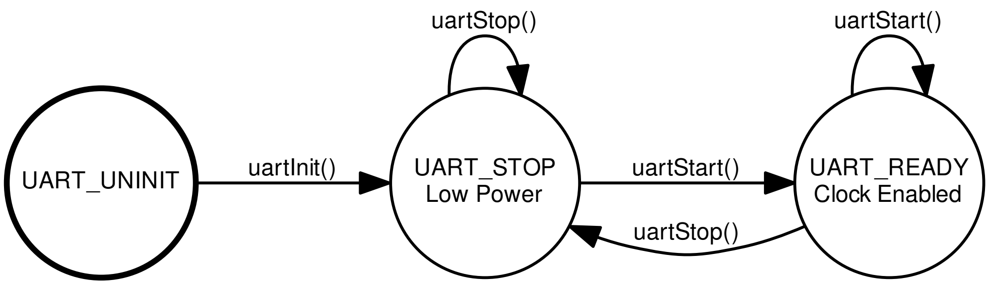
If your application requires a synchronous buffered driver then the Serial Driver should be used instead.

Precondition

In order to use the UART driver the `HAL_USE_UART` option must be enabled in `halconf.h`.

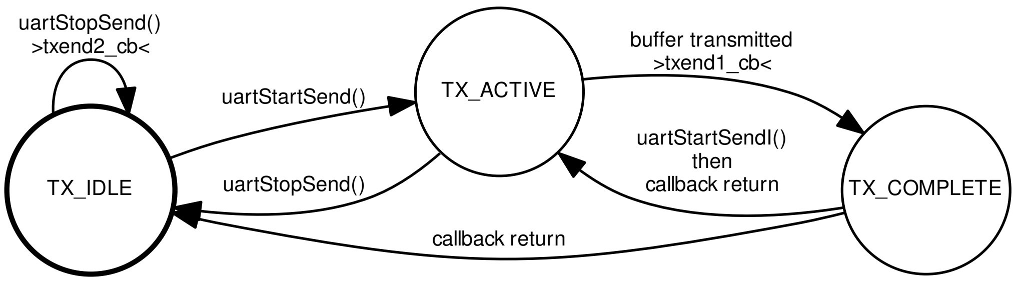
7.38.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



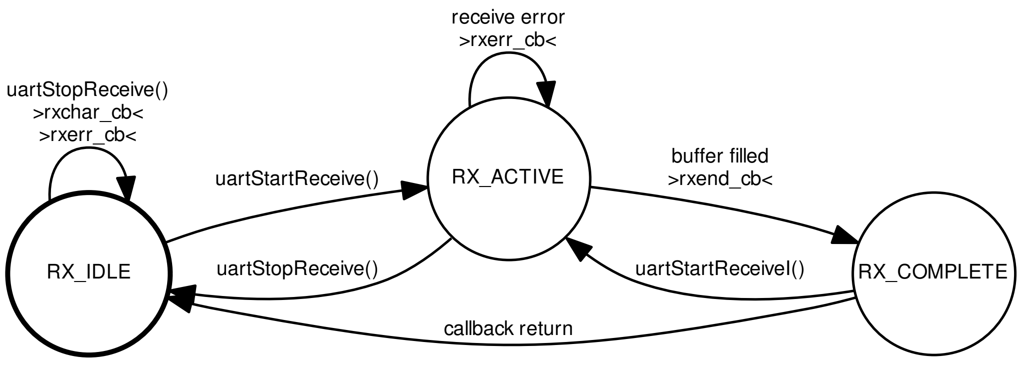
7.38.2.1 Transmitter sub State Machine

The follow diagram describes the transmitter state machine, this diagram is valid while the driver is in the `UAR←T_READY` state. This state machine is automatically reset to the `TX_IDLE` state each time the driver enters the `UART_READY` state.



7.38.2.2 Receiver sub State Machine

The follow diagram describes the receiver state machine, this diagram is valid while the driver is in the `UART←_READY` state. This state machine is automatically reset to the `RX_IDLE` state each time the driver enters the `UART_READY` state.



UART status flags

- #define UART_NO_ERROR 0

  No pending conditions.
- #define UART_PARITY_ERROR 4

  Parity error happened.
- #define UART_FRAMING_ERROR 8

Framing error happened.
- #define UART_OVERRUN_ERROR 16

  Overflow happened.
- #define UART_NOISE_ERROR 32

  Noise on the line.
- #define UART_BREAK_DETECTED 64

  Break detected.

UART configuration options

- #define UART_USE_WAIT FALSE

  Enables synchronous APIs.
- #define UART_USE_MUTUAL_EXCLUSION FALSE

  Enables the `uartAcquireBus()` and `uartReleaseBus()` APIs.

Low level driver helper macros

- #define _uart_wakeup_tx1_isr(uartp)

  Wakes up the waiting thread in case of early TX complete.
- #define _uart_wakeup_tx2_isr(uartp)

  Wakes up the waiting thread in case of late TX complete.
- #define _uart_wakeup_rx_complete_isr(uartp)

  Wakes up the waiting thread in case of RX complete.
- #define _uart_wakeup_rx_error_isr(uartp)

  Wakes up the waiting thread in case of RX error.
- #define _uart_wakeup_rx_timeout_isr(uartp)

  Wakes up the waiting thread in case of RX timeout.
- #define _uart_tx1_isr_code(uartp)

  Common ISR code for early TX.
- #define _uart_tx2_isr_code(uartp)

  Common ISR code for late TX.
- #define _uart_rx_complete_isr_code(uartp)

  Common ISR code for RX complete.
- #define _uart_rx_error_isr_code(uartp, errors)

  Common ISR code for RX error.
- #define _uart_rx_idle_code(uartp)

  Common ISR code for RX on idle.
- #define _uart_timeout_isr_code(uartp)

  Timeout ISR code for receiver.

PLATFORM configuration options

- #define PLATFORM_UART_USE_UART1 FALSE

  UART driver enable switch.

Typedefs

- typedef uint32_t uartflags_t

  UART driver condition flags type.
- typedef struct UARTDriver UARTDriver

  Type of structure representing an UART driver.
- typedef void(∗ uartcb_t) (UARTDriver ∗uartp)

  Generic UART notification callback type.
- typedef void(∗ uartccb_t) (UARTDriver ∗uartp, uint16_t c)

  Character received UART notification callback type.
- typedef void(∗ uartecb_t) (UARTDriver ∗uartp, uartflags_t e)

  Receive error UART notification callback type.

Data Structures

- struct UARTConfig

  Driver configuration structure.
- struct UARTDriver

  Structure representing an UART driver.

Functions

- void uartInit (void)

  UART Driver initialization.
- void uartObjectInit (UARTDriver ∗uartp)

  Initializes the standard part of a `UARTDriver` structure.
- void uartStart (UARTDriver ∗uartp, const UARTConfig ∗config)

  Configures and activates the UART peripheral.
- void uartStop (UARTDriver ∗uartp)

  Deactivates the UART peripheral.
- void uartStartSend (UARTDriver ∗uartp, size_t n, const void ∗txbuf)

  Starts a transmission on the UART peripheral.
- void uartStartSendI (UARTDriver ∗uartp, size_t n, const void ∗txbuf)

  Starts a transmission on the UART peripheral.
- size_t uartStopSend (UARTDriver ∗uartp)

  Stops any ongoing transmission.
- size_t uartStopSendI (UARTDriver ∗uartp)

  Stops any ongoing transmission.
- void uartStartReceive (UARTDriver ∗uartp, size_t n, void ∗rxbuf)

  Starts a receive operation on the UART peripheral.
- void uartStartReceiveI (UARTDriver ∗uartp, size_t n, void ∗rxbuf)

  Starts a receive operation on the UART peripheral.
- size_t uartStopReceive (UARTDriver ∗uartp)

  Stops any ongoing receive operation.
- size_t uartStopReceiveI (UARTDriver ∗uartp)

  Stops any ongoing receive operation.
- msg_t uartSendTimeout (UARTDriver ∗uartp, size_t ∗np, const void ∗txbuf, systime_t timeout)

  Performs a transmission on the UART peripheral.
- msg_t uartSendFullTimeout (UARTDriver ∗uartp, size_t ∗np, const void ∗txbuf, systime_t timeout)

  Performs a transmission on the UART peripheral.

- msg_t uartReceiveTimeout (UARTDriver ∗uartp, size_t ∗np, void ∗rxbuf, systime_t timeout)

    Performs a receive operation on the UART peripheral.
- void uartAcquireBus (UARTDriver ∗uartp)

    Gains exclusive access to the UART bus.
- void uartReleaseBus (UARTDriver ∗uartp)

    Releases exclusive access to the UART bus.
- void uart_lld_init (void)

    Low level UART driver initialization.
- void uart_lld_start (UARTDriver ∗uartp)

    Configures and activates the UART peripheral.
- void uart_lld_stop (UARTDriver ∗uartp)

    Deactivates the UART peripheral.
- void uart_lld_start_send (UARTDriver ∗uartp, size_t n, const void ∗txbuf)

    Starts a transmission on the UART peripheral.
- size_t uart_lld_stop_send (UARTDriver ∗uartp)

    Stops any ongoing transmission.
- void uart_lld_start_receive (UARTDriver ∗uartp, size_t n, void ∗rxbuf)

    Starts a receive operation on the UART peripheral.
- size_t uart_lld_stop_receive (UARTDriver ∗uartp)

    Stops any ongoing receive operation.

Enumerations

Variables

- UARTDriver UARTD1

    UART1 driver identifier.

7.38.3 Macro Definition Documentation

7.38.3.1 #define UART_NO_ERROR 0

No pending conditions.

7.38.3.2 #define UART_PARITY_ERROR 4

Parity error happened.

7.38.3.3 #define UART_FRAMING_ERROR 8

Framing error happened.

7.38.3.4 #define UART_OVERRUN_ERROR 16

Overflow happened.

7.38.3.5 #define UART_NOISE_ERROR 32

Noise on the line.

7.38.3.6 #define UART_BREAK_DETECTED 64

Break detected.

7.38.3.7 #define UART_USE_WAIT FALSE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.38.3.8 #define UART_USE_MUTUAL_EXCLUSION FALSE

Enables the `uartAcquireBus()` and `uartReleaseBus()` APIs.

Note

Disabling this option saves both code and data space.

7.38.3.9 #define _uart_wakeup_tx1_isr(*uartp*)

Value:

```
{                                        \
  if ((uartp)->early == true) {                                        \
    osalSysLockFromISR();                                              \
    osalThreadResumeI(&(uartp)->threadtx, MSG_OK);                     \
    osalSysUnlockFromISR();                                            \
  }                                                                    \
}
```

Wakes up the waiting thread in case of early TX complete.

Parameters

in	*uartp*	pointer to the `UARTDriver` object

Function Class:

Not an API, this function is for internal use only.

7.38.3.10 #define _uart_wakeup_tx2_isr(*uartp*)

Value:

```
{                                        \
  if ((uartp)->early == false) {                                       \
    osalSysLockFromISR();                                              \
    osalThreadResumeI(&(uartp)->threadtx, MSG_OK);                     \
    osalSysUnlockFromISR();                                            \
  }                                                                    \
}
```

Wakes up the waiting thread in case of late TX complete.

Parameters

in	*uartp*	pointer to the UARTDriver object

Function Class:

Not an API, this function is for internal use only.

7.38.3.11 #define _uart_wakeup_rx_complete_isr(*uartp*)

Value:

```
{                                \
  osalSysLockFromISR();                                              \
  osalThreadResumeI(&(uartp)->threadrx, MSG_OK);                     \
  osalSysUnlockFromISR();                                            \
}
```

Wakes up the waiting thread in case of RX complete.

Parameters

in	*uartp*	pointer to the UARTDriver object

Function Class:

Not an API, this function is for internal use only.

7.38.3.12 #define _uart_wakeup_rx_error_isr(*uartp*)

Value:

```
{                                   \
  osalSysLockFromISR();                                              \
  osalThreadResumeI(&(uartp)->threadrx, MSG_RESET);                  \
  osalSysUnlockFromISR();                                            \
}
```

Wakes up the waiting thread in case of RX error.

Parameters

in	*uartp*	pointer to the UARTDriver object

Function Class:

Not an API, this function is for internal use only.

7.38.3.13 #define _uart_wakeup_rx_timeout_isr(*uartp*)

Value:

```
{                                 \
  osalSysLockFromISR();                                              \
  osalThreadResumeI(&(uartp)->threadrx, MSG_TIMEOUT);                             \
  osalSysUnlockFromISR();                                                         \
}
```

Wakes up the waiting thread in case of RX timeout.

Parameters

in	*uartp*	pointer to the `UARTDriver` object

Function Class:

Not an API, this function is for internal use only.

7.38.3.14 #define _uart_tx1_isr_code(*uartp*)

Value:

```
{                                          \
  (uartp)->txstate = UART_TX_COMPLETE;                                            \
  if ((uartp)->config->txend1_cb != NULL) {                                       \
    (uartp)->config->txend1_cb(uartp);                                            \
  }                                                                               \
  if ((uartp)->txstate == UART_TX_COMPLETE) {                                     \
    (uartp)->txstate = UART_TX_IDLE;                                              \
  }                                                                               \
  _uart_wakeup_tx1_isr(uartp);                                                    \
}
```

Common ISR code for early TX.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

in	*uartp*	pointer to the `UARTDriver` object

Function Class:

Not an API, this function is for internal use only.

7.38.3.15 #define _uart_tx2_isr_code(*uartp*)

Value:

```
{                                          \
  if ((uartp)->config->txend2_cb != NULL) {                                       \
    (uartp)->config->txend2_cb(uartp);                                            \
  }                                                                      \
  _uart_wakeup_tx2_isr(uartp);                                           \
}
```

Common ISR code for late TX.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

in	*uartp*	pointer to the UARTDriver object

Function Class:

Not an API, this function is for internal use only.

7.38.3.16 #define _uart_rx_complete_isr_code(*uartp*)

Value:

```
{                                  \
  (uartp)->rxstate = UART_RX_COMPLETE;                                   \
  if ((uartp)->config->rxend_cb != NULL) {                               \
    (uartp)->config->rxend_cb(uartp);                                    \
  }                                                                      \
  if ((uartp)->rxstate == UART_RX_COMPLETE) {                            \
    (uartp)->rxstate = UART_RX_IDLE;                                     \
    uart_enter_rx_idle_loop(uartp);                                      \
  }                                                                      \
  _uart_wakeup_rx_complete_isr(uartp);                                   \
}
```

Common ISR code for RX complete.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

in	*uartp*	pointer to the UARTDriver object

Function Class:

Not an API, this function is for internal use only.

7.38.3.17 #define _uart_rx_error_isr_code(*uartp, errors*)

Value:

```
{                             \
  if ((uartp)->config->rxerr_cb != NULL) {                                \
    (uartp)->config->rxerr_cb(uartp, errors);                             \
  }                                                                       \
  _uart_wakeup_rx_error_isr(uartp);                                       \
}
```

Common ISR code for RX error.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

`in`	*uartp*	pointer to the `UARTDriver` object
`in`	*errors*	mask of errors to be reported

Function Class:

Not an API, this function is for internal use only.

7.38.3.18 #define _uart_rx_idle_code(*uartp*)

Value:

```
{                                          \
  if ((uartp)->config->rxchar_cb != NULL)                                 \
    (uartp)->config->rxchar_cb(uartp, (uartp)->rxbuf);                    \
}
```

Common ISR code for RX on idle.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

in	*uartp*	pointer to the `UARTDriver` object

Function Class:

Not an API, this function is for internal use only.

7.38.3.19 #define _uart_timeout_isr_code(*uartp*)

Value:

```
{                                                     \
  if ((uartp)->config->timeout_cb != NULL) {                                \
    (uartp)->config->timeout_cb(uartp);                                     \
  }                                                                         \
  _uart_wakeup_rx_timeout_isr(uartp);                                       \
}
```

Timeout ISR code for receiver.

This code handles the portable part of the ISR code:

- Callback invocation.
- Waiting thread wakeup, if any.
- Driver state transitions.

Note

This macro is meant to be used in the low level drivers implementation only.

Parameters

in	*uartp*	pointer to the `UARTDriver` object

Function Class:

Not an API, this function is for internal use only.

7.38.3.20 #define PLATFORM_UART_USE_UART1 FALSE

UART driver enable switch.

If set to `TRUE` the support for UART1 is included.

Note

The default is `FALSE`.

7.38.4 Typedef Documentation

7.38.4.1 typedef uint32_t **uartflags_t**

UART driver condition flags type.

7.38.4.2 typedef struct UARTDriver UARTDriver

Type of structure representing an UART driver.

7.38.4.3 typedef void(∗ uartcb_t) (UARTDriver ∗uartp)

Generic UART notification callback type.

Parameters

in	*uartp*	pointer to the `UARTDriver` object

7.38.4.4 typedef void(∗ uartccb_t) (UARTDriver ∗uartp, uint16_t c)

Character received UART notification callback type.

Parameters

in	*uartp*	pointer to the `UARTDriver` object triggering the callback
in	*c*	received character

7.38.4.5 typedef void(∗ uartecb_t) (UARTDriver ∗uartp, uartflags_t e)

Receive error UART notification callback type.

Parameters

in	*uartp*	pointer to the `UARTDriver` object triggering the callback
in	*e*	receive error mask

7.38.5 Enumeration Type Documentation

7.38.5.1 enum uartstate_t

Driver state machine possible states.

Enumerator

UART_UNINIT Not initialized.

UART_STOP Stopped.

UART_READY Ready.

7.38.5.2 enum uarttxstate_t

Transmitter state machine states.

Enumerator

UART_TX_IDLE Not transmitting.

UART_TX_ACTIVE	Transmitting.
UART_TX_COMPLETE	Buffer complete.

7.38.5.3 enum uartrxstate_t

Receiver state machine states.

Enumerator

UART_RX_IDLE	Not receiving.
UART_RX_ACTIVE	Receiving.
UART_RX_COMPLETE	Buffer complete.

7.38.6 Function Documentation

7.38.6.1 void uartInit (void)

UART Driver initialization.

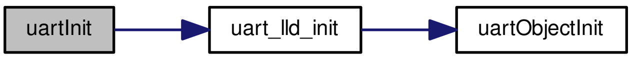
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



7.38.6.2 void uartObjectInit (UARTDriver * *uartp*)

Initializes the standard part of a `UARTDriver` structure.

Parameters

`out`	*uartp*	pointer to the `UARTDriver` object

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.38.6.3 void uartStart (**UARTDriver** ∗ *uartp*, const **UARTConfig** ∗ *config*)

Configures and activates the UART peripheral.

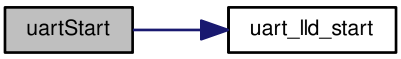
Parameters

in	*uartp*	pointer to the `UARTDriver` object
in	*config*	pointer to the `UARTConfig` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.38.6.4 void uartStop (**UARTDriver** ∗ *uartp*)

Deactivates the UART peripheral.

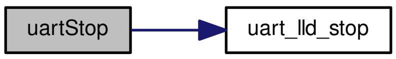
Parameters

in	*uartp*	pointer to the `UARTDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.38.6.5 void uartStartSend (**UARTDriver** ∗ *uartp*, size_t *n*, const void ∗ *txbuf*)

Starts a transmission on the UART peripheral.

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

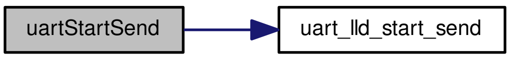
Parameters

in	*uartp*	pointer to the `UARTDriver` object
in	*n*	number of data frames to send
in	*txbuf*	the pointer to the transmit buffer

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.38.6.6 void uartStartSendI (**UARTDriver** ∗ *uartp,* size_t *n,* const void ∗ *txbuf*)

Starts a transmission on the UART peripheral.

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.
This function has to be invoked from a lock zone.

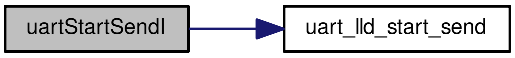
Parameters

in	*uartp*	pointer to the `UARTDriver` object
in	*n*	number of data frames to send
in	*txbuf*	the pointer to the transmit buffer

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



7.38.6.7 size_t uartStopSend (UARTDriver * *uartp*)

Stops any ongoing transmission.

Note

Stopping a transmission also suppresses the transmission callbacks.

Parameters

in	*uartp*	pointer to the UARTDriver object

Returns

The number of data frames not transmitted by the stopped transmit operation.

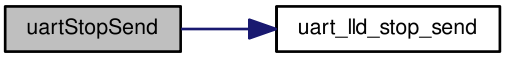
Return values

0	There was no transmit operation in progress.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.38.6.8 size_t uartStopSendI (UARTDriver * *uartp*)

Stops any ongoing transmission.

Note

Stopping a transmission also suppresses the transmission callbacks.
This function has to be invoked from a lock zone.

Parameters

`in`	*uartp*	pointer to the `UARTDriver` object

Returns

The number of data frames not transmitted by the stopped transmit operation.

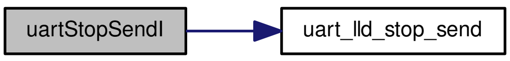
Return values

0	There was no transmit operation in progress.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



7.38.6.9 void uartStartReceive (**UARTDriver** ∗ *uartp,* size_t *n,* void ∗ *rxbuf*)

Starts a receive operation on the UART peripheral.

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

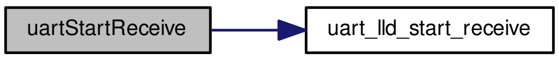
Parameters

`in`	*uartp*	pointer to the `UARTDriver` object
`in`	*n*	number of data frames to receive
`in`	*rxbuf*	the pointer to the receive buffer

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.38.6.10 void uartStartReceiveI (UARTDriver ∗ *uartp*, size_t *n*, void ∗ *rxbuf*)

Starts a receive operation on the UART peripheral.

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.
This function has to be invoked from a lock zone.

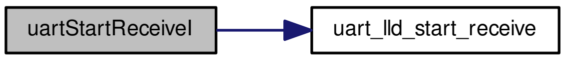
Parameters

`in`	*uartp*	pointer to the `UARTDriver` object
`in`	*n*	number of data frames to receive
`out`	*rxbuf*	the pointer to the receive buffer

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



7.38.6.11 size_t uartStopReceive (UARTDriver ∗ *uartp*)

Stops any ongoing receive operation.

Note

Stopping a receive operation also suppresses the receive callbacks.

Parameters

`in`	*uartp*	pointer to the `UARTDriver` object

Returns

The number of data frames not received by the stopped receive operation.

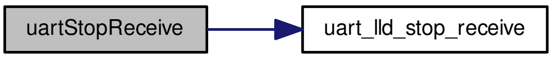
Return values

0	There was no receive operation in progress.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.38.6.12 size_t uartStopReceiveI (UARTDriver * *uartp*)

Stops any ongoing receive operation.

Note

Stopping a receive operation also suppresses the receive callbacks.
This function has to be invoked from a lock zone.

Parameters

`in`	*uartp*	pointer to the `UARTDriver` object

Returns

The number of data frames not received by the stopped receive operation.

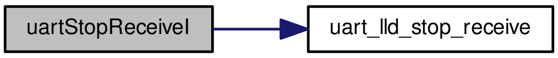
Return values

0	There was no receive operation in progress.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



7.38.6.13 msg_t uartSendTimeout (UARTDriver ∗ *uartp,* size_t ∗ *np,* const void ∗ *txbuf,* systime_t *timeout*)

Performs a transmission on the UART peripheral.

Note

The function returns when the specified number of frames have been sent to the UART or on timeout.
The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.
This function implements a software timeout, it does not use any underlying HW timeout mechanism.

Parameters

`in`	*uartp*	pointer to the `UARTDriver` object
`in,out`	*np*	number of data frames to transmit, on exit the number of frames actually transmitted
`in`	*txbuf*	the pointer to the transmit buffer
`in`	*timeout*	operation timeout

Returns

The operation status.

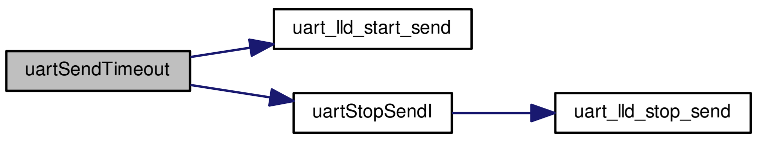
Return values

MSG_OK	if the operation completed successfully.
MSG_TIMEOUT	if the operation timed out.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.38.6.14 msg_t uartSendFullTimeout (UARTDriver ∗ *uartp*, size_t ∗ *np*, const void ∗ *txbuf*, systime_t *timeout*)

Performs a transmission on the UART peripheral.

Note

The function returns when the specified number of frames have been physically transmitted or on timeout.
The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.
This function implements a software timeout, it does not use any underlying HW timeout mechanism.

Parameters

`in`	*uartp*	pointer to the `UARTDriver` object
`in,out`	*np*	number of data frames to transmit, on exit the number of frames actually transmitted
`in`	*txbuf*	the pointer to the transmit buffer
`in`	*timeout*	operation timeout

Returns

The operation status.

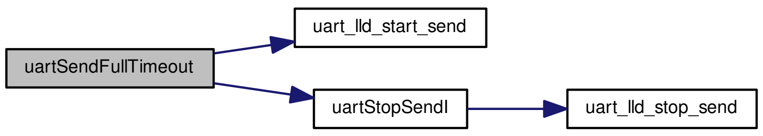
Return values

MSG_OK	if the operation completed successfully.
MSG_TIMEOUT	if the operation timed out.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.38.6.15 msg_t uartReceiveTimeout (UARTDriver * *uartp,* size_t * *np,* void * *rxbuf,* systime_t *timeout*)

Performs a receive operation on the UART peripheral.

Note

The function returns when the specified number of frames have been received or on error/timeout.
The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.
This function implements a software timeout, it does not use any underlying HW timeout mechanism.

Parameters

`in`	*uartp*	pointer to the `UARTDriver` object
`in,out`	*np*	number of data frames to receive, on exit the number of frames actually received
`in`	*rxbuf*	the pointer to the receive buffer
`in`	*timeout*	operation timeout

Returns

The operation status.

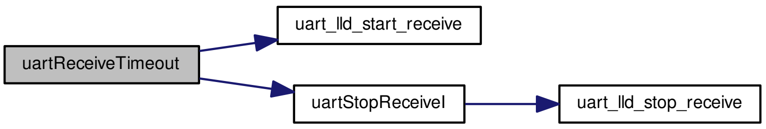
Return values

MSG_OK	if the operation completed successfully.
MSG_TIMEOUT	if the operation timed out.
MSG_RESET	in case of a receive error.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.38.6.16 void uartAcquireBus (**UARTDriver** ∗ *uartp*)

Gains exclusive access to the UART bus.

This function tries to gain ownership to the UART bus, if the bus is already being used then the invoking thread is queued.

Precondition

In order to use this function the option `UART_USE_MUTUAL_EXCLUSION` must be enabled.

Parameters

in	*uartp*	pointer to the `UARTDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.38.6.17 void uartReleaseBus (**UARTDriver** ∗ *uartp*)

Releases exclusive access to the UART bus.

Precondition

In order to use this function the option `UART_USE_MUTUAL_EXCLUSION` must be enabled.

Parameters

in	*uartp*	pointer to the `UARTDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

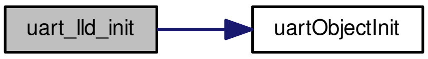
7.38.6.18 void uart_lld_init (void)

Low level UART driver initialization.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.38.6.19 void uart_lld_start (**UARTDriver** ∗ *uartp*)

Configures and activates the UART peripheral.

Parameters

in	*uartp*	pointer to the UARTDriver object

Function Class:

Not an API, this function is for internal use only.

7.38.6.20 void uart_lld_stop (**UARTDriver** ∗ *uartp*)

Deactivates the UART peripheral.

Parameters

in	*uartp*	pointer to the UARTDriver object

Function Class:

Not an API, this function is for internal use only.

7.38.6.21 void uart_lld_start_send (**UARTDriver** ∗ *uartp,* size_t *n,* const void ∗ *txbuf*)

Starts a transmission on the UART peripheral.

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

Parameters

in	*uartp*	pointer to the UARTDriver object
in	*n*	number of data frames to send
in	*txbuf*	the pointer to the transmit buffer

Function Class:

Not an API, this function is for internal use only.

7.38.6.22 size_t uart_lld_stop_send (UARTDriver ∗ *uartp*)

Stops any ongoing transmission.

Note

Stopping a transmission also suppresses the transmission callbacks.

Parameters

in	*uartp*	pointer to the UARTDriver object

Returns

The number of data frames not transmitted by the stopped transmit operation.

Function Class:

Not an API, this function is for internal use only.

7.38.6.23 void uart_lld_start_receive (UARTDriver ∗ *uartp*, size_t *n*, void ∗ *rxbuf*)

Starts a receive operation on the UART peripheral.

Note

The buffers are organized as uint8_t arrays for data sizes below or equal to 8 bits else it is organized as uint16_t arrays.

Parameters

in	*uartp*	pointer to the UARTDriver object
in	*n*	number of data frames to send
out	*rxbuf*	the pointer to the receive buffer

Function Class:

Not an API, this function is for internal use only.

7.38.6.24 size_t uart_lld_stop_receive (UARTDriver ∗ *uartp*)

Stops any ongoing receive operation.

Note

Stopping a receive operation also suppresses the receive callbacks.

Parameters

`in`	*uartp*	pointer to the `UARTDriver` object

Returns

The number of data frames not received by the stopped receive operation.

Function Class:

Not an API, this function is for internal use only.

7.38.7 Variable Documentation

7.38.7.1 UARTDriver UARTD1

UART1 driver identifier.

7.39 USB Driver

Generic USB Driver.

7.39.1 Detailed Description

Generic USB Driver.

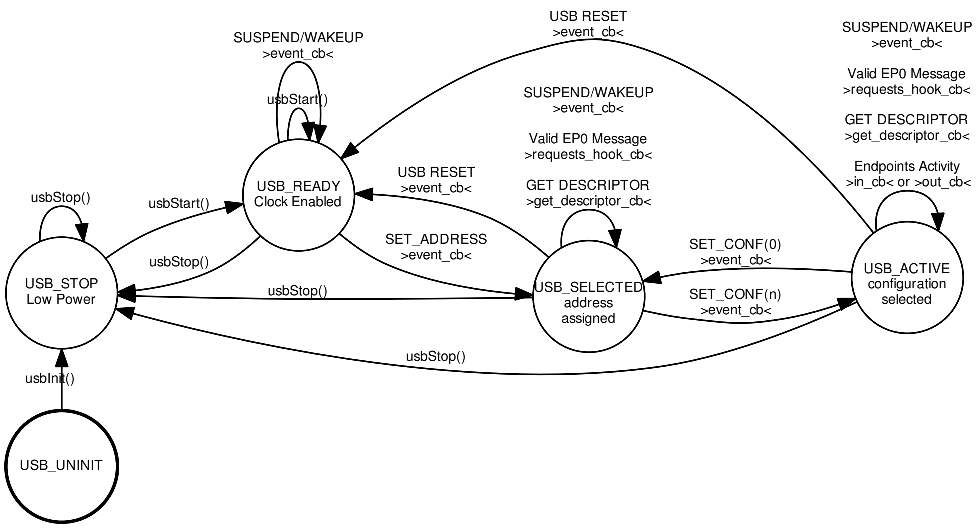
This module implements a generic USB (Universal Serial Bus) driver supporting device-mode operations.

Precondition

In order to use the USB driver the `HAL_USE_USB` option must be enabled in `halconf.h`.

7.39.2 Driver State Machine

The driver implements a state machine internally, not all the driver functionalities can be used in any moment, any transition not explicitly shown in the following diagram has to be considered an error and shall be captured by an assertion (if enabled).



7.39.3 USB Operations

The USB driver is quite complex and USB is complex in itself, it is recommended to study the USB specification before trying to use the driver.

7.39.3.1 USB Implementation

The USB driver abstracts the inner details of the underlying USB hardware. The driver works asynchronously and communicates with the application using callbacks. The application is responsible of the descriptors and strings required by the USB device class to be implemented and of the handling of the specific messages sent over the endpoint zero. Standard messages are handled internally to the driver. The application can use hooks in order to handle custom messages or override the handling of the default handling of standard messages.

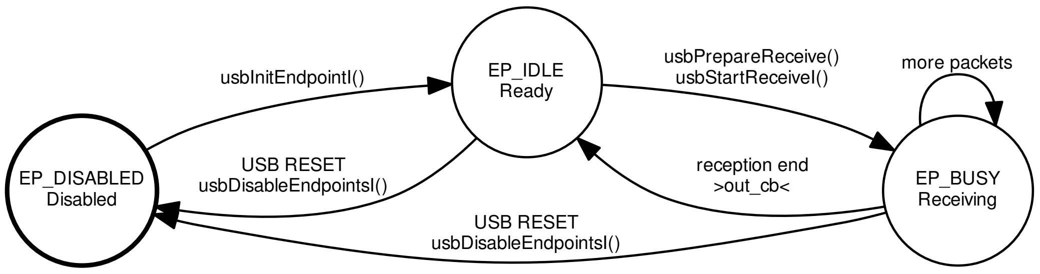
7.39.3.2 USB Endpoints

USB endpoints are the objects that the application uses to exchange data with the host. There are two kind of endpoints:

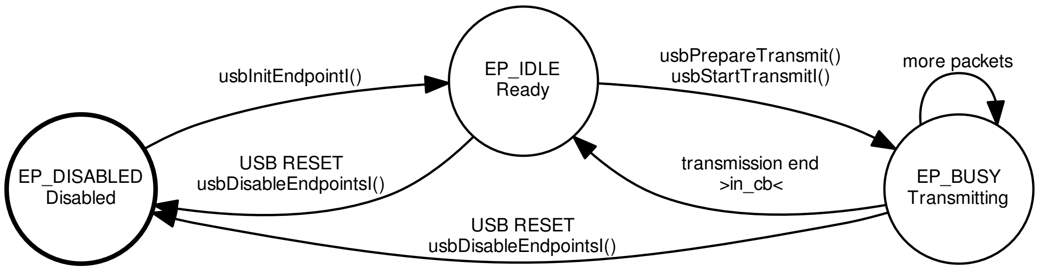
- **IN** endpoints are used by the application to transmit data to the host.

- **OUT** endpoints are used by the application to receive data from the host.

The driver invokes a callback after finishing an IN or OUT transaction. States diagram for OUT endpoints in transaction mode:



States diagram for IN endpoints in transaction mode:



7.39.3.3 USB Callbacks

The USB driver uses callbacks in order to interact with the application. There are several kinds of callbacks to be handled:

- Driver events callback. As example errors, suspend event, reset event etc.
- Messages Hook callback. This hook allows the application to implement handling of custom messages or to override the default handling of standard messages on endpoint zero.
- Descriptor Requested callback. When the driver endpoint zero handler receives a GET DESCRIPTOR message and needs to send a descriptor to the host it queries the application using this callback.
- Start of Frame callback. This callback is invoked each time a SOF packet is received.
- Endpoint callbacks. Each endpoint informs the application about I/O conditions using those callbacks.

Macros

- #define USB_USE_WAIT FALSE

  Enables synchronous APIs.
- #define USB_MAX_ENDPOINTS 4

  Maximum endpoint address.
- #define USB_EP0_STATUS_STAGE USB_EP0_STATUS_STAGE_SW

  Status stage handling method.
- #define USB_SET_ADDRESS_MODE USB_LATE_SET_ADDRESS

  The address can be changed immediately upon packet reception.
- #define USB_SET_ADDRESS_ACK_HANDLING USB_SET_ADDRESS_ACK_SW

  Method for set address acknowledge.
- #define usb_lld_get_frame_number(usbp) 0

  Returns the current frame number.
- #define usb_lld_get_transaction_size(usbp, ep) ((usbp)->epc[ep]->out_state->rxcnt)

  Returns the exact size of a receive transaction.
- #define usb_lld_connect_bus(usbp)

  Connects the USB device.
- #define usb_lld_disconnect_bus(usbp)

  Disconnect the USB device.

Helper macros for USB descriptors

- #define USB_DESC_INDEX(i) ((uint8_t)(i))

  Helper macro for index values into descriptor strings.
- #define USB_DESC_BYTE(b) ((uint8_t)(b))

  Helper macro for byte values into descriptor strings.
- #define USB_DESC_WORD(w)

  Helper macro for word values into descriptor strings.
- #define USB_DESC_BCD(bcd)

  Helper macro for BCD values into descriptor strings.
- #define **USB_DESC_DEVICE_SIZE** 18U
- #define USB_DESC_DEVICE(bcdUSB, bDeviceClass, bDeviceSubClass, bDeviceProtocol, bMaxPacket←Size, idVendor, idProduct, bcdDevice, iManufacturer, iProduct, iSerialNumber, bNumConfigurations)

  Device Descriptor helper macro.
- #define USB_DESC_CONFIGURATION_SIZE 9U

Configuration Descriptor size.

- #define USB_DESC_CONFIGURATION(wTotalLength, bNumInterfaces, bConfigurationValue, iConfiguration, bmAttributes, bMaxPower)

  Configuration Descriptor helper macro.
- #define USB_DESC_INTERFACE_SIZE 9U

  Interface Descriptor size.
- #define USB_DESC_INTERFACE(bInterfaceNumber, bAlternateSetting, bNumEndpoints, bInterfaceClass, bInterfaceSubClass, bInterfaceProtocol, iInterface)

  Interface Descriptor helper macro.
- #define USB_DESC_INTERFACE_ASSOCIATION_SIZE 8U

  Interface Association Descriptor size.
- #define USB_DESC_INTERFACE_ASSOCIATION(bFirstInterface, bInterfaceCount, bFunctionClass, b↩ FunctionSubClass, bFunctionProcotol, iInterface)

  Interface Association Descriptor helper macro.
- #define USB_DESC_ENDPOINT_SIZE 7U

  Endpoint Descriptor size.
- #define USB_DESC_ENDPOINT(bEndpointAddress, bmAttributes, wMaxPacketSize, bInterval)

  Endpoint Descriptor helper macro.

Endpoint types and settings

- #define USB_EP_MODE_TYPE 0x0003U
- #define USB_EP_MODE_TYPE_CTRL 0x0000U
- #define USB_EP_MODE_TYPE_ISOC 0x0001U
- #define USB_EP_MODE_TYPE_BULK 0x0002U
- #define USB_EP_MODE_TYPE_INTR 0x0003U

Macro Functions

- #define usbGetDriverStateI(usbp) ((usbp)->state)

  Returns the driver state.
- #define usbConnectBus(usbp) usb_lld_connect_bus(usbp)

  Connects the USB device.
- #define usbDisconnectBus(usbp) usb_lld_disconnect_bus(usbp)

  Disconnect the USB device.
- #define usbGetFrameNumberX(usbp) usb_lld_get_frame_number(usbp)

  Returns the current frame number.
- #define usbGetTransmitStatusI(usbp, ep) (((usbp)->transmitting & (uint16_t)((unsigned)1U << (unsigned)(ep))) != 0U)

  Returns the status of an IN endpoint.
- #define usbGetReceiveStatusI(usbp, ep) (((usbp)->receiving & (uint16_t)((unsigned)1U << (unsigned)(ep))) != 0U)

  Returns the status of an OUT endpoint.
- #define usbGetReceiveTransactionSizeX(usbp, ep) usb_lld_get_transaction_size(usbp, ep)

  Returns the exact size of a receive transaction.
- #define usbSetupTransfer(usbp, buf, n, endcb)

  Request transfer setup.
- #define usbReadSetup(usbp, ep, buf) usb_lld_read_setup(usbp, ep, buf)

  Reads a setup packet from the dedicated packet buffer.

Low level driver helper macros

- #define _usb_isr_invoke_event_cb(usbp, evt)

  Common ISR code, usb event callback.
- #define _usb_isr_invoke_sof_cb(usbp)

  Common ISR code, SOF callback.
- #define _usb_isr_invoke_setup_cb(usbp, ep)

  Common ISR code, setup packet callback.
- #define _usb_isr_invoke_in_cb(usbp, ep)

  Common ISR code, IN endpoint callback.
- #define _usb_isr_invoke_out_cb(usbp, ep)

  Common ISR code, OUT endpoint event.

PLATFORM configuration options

- #define PLATFORM_USB_USE_USB1 FALSE

  USB driver enable switch.

Typedefs

- typedef struct USBDriver USBDriver

  Type of a structure representing an USB driver.
- typedef uint8_t usbep_t

  Type of an endpoint identifier.
- typedef void(∗ usbcallback_t) (USBDriver ∗usbp)

  Type of an USB generic notification callback.
- typedef void(∗ usbepcallback_t) (USBDriver ∗usbp, usbep_t ep)

  Type of an USB endpoint callback.
- typedef void(∗ usbeventcb_t) (USBDriver ∗usbp, usbevent_t event)

  Type of an USB event notification callback.
- typedef bool(∗ usbreqhandler_t) (USBDriver ∗usbp)

  Type of a requests handler callback.
- typedef const USBDescriptor ∗(∗ usbgetdescriptor_t) (USBDriver ∗usbp, uint8_t dtype, uint8_t dindex, uint16_t lang)

  Type of an USB descriptor-retrieving callback.

Data Structures

- struct USBDescriptor

  Type of an USB descriptor.
- struct USBInEndpointState

  Type of an IN endpoint state structure.
- struct USBOutEndpointState

  Type of an OUT endpoint state structure.
- struct USBEndpointConfig

  Type of an USB endpoint configuration structure.
- struct USBConfig

  Type of an USB driver configuration structure.
- struct USBDriver

  Structure representing an USB driver.

Functions

- static void set_address (USBDriver ∗usbp)

  SET ADDRESS transaction callback.
- static bool default_handler (USBDriver ∗usbp)

  Standard requests handler.
- void usbInit (void)

  USB Driver initialization.
- void usbObjectInit (USBDriver ∗usbp)

  Initializes the standard part of a `USBDriver` structure.
- void usbStart (USBDriver ∗usbp, const USBConfig ∗config)

  Configures and activates the USB peripheral.
- void usbStop (USBDriver ∗usbp)

  Deactivates the USB peripheral.
- void usbInitEndpointI (USBDriver ∗usbp, usbep_t ep, const USBEndpointConfig ∗epcp)

  Enables an endpoint.
- void usbDisableEndpointsI (USBDriver ∗usbp)

  Disables all the active endpoints.
- void usbStartReceiveI (USBDriver ∗usbp, usbep_t ep, uint8_t ∗buf, size_t n)

  Starts a receive transaction on an OUT endpoint.
- void usbStartTransmitI (USBDriver ∗usbp, usbep_t ep, const uint8_t ∗buf, size_t n)

  Starts a transmit transaction on an IN endpoint.
- msg_t usbReceive (USBDriver ∗usbp, usbep_t ep, uint8_t ∗buf, size_t n)

  Performs a receive transaction on an OUT endpoint.
- msg_t usbTransmit (USBDriver ∗usbp, usbep_t ep, const uint8_t ∗buf, size_t n)

  Performs a transmit transaction on an IN endpoint.
- bool usbStallReceiveI (USBDriver ∗usbp, usbep_t ep)

  Stalls an OUT endpoint.
- bool usbStallTransmitI (USBDriver ∗usbp, usbep_t ep)

  Stalls an IN endpoint.
- void _usb_reset (USBDriver ∗usbp)

  USB reset routine.
- void _usb_suspend (USBDriver ∗usbp)

  USB suspend routine.
- void _usb_wakeup (USBDriver ∗usbp)

  USB wake-up routine.
- void _usb_ep0setup (USBDriver ∗usbp, usbep_t ep)

  Default EP0 SETUP callback.
- void _usb_ep0in (USBDriver ∗usbp, usbep_t ep)

  Default EP0 IN callback.
- void _usb_ep0out (USBDriver ∗usbp, usbep_t ep)

  Default EP0 OUT callback.
- void usb_lld_init (void)

  Low level USB driver initialization.
- void usb_lld_start (USBDriver ∗usbp)

  Configures and activates the USB peripheral.
- void usb_lld_stop (USBDriver ∗usbp)

  Deactivates the USB peripheral.
- void usb_lld_reset (USBDriver ∗usbp)

  USB low level reset routine.
- void usb_lld_set_address (USBDriver ∗usbp)

Sets the USB address.

- void usb_lld_init_endpoint (USBDriver ∗usbp, usbep_t ep)

  Enables an endpoint.
- void usb_lld_disable_endpoints (USBDriver ∗usbp)

  Disables all the active endpoints except the endpoint zero.
- usbepstatus_t usb_lld_get_status_out (USBDriver ∗usbp, usbep_t ep)

  Returns the status of an OUT endpoint.
- usbepstatus_t usb_lld_get_status_in (USBDriver ∗usbp, usbep_t ep)

  Returns the status of an IN endpoint.
- void usb_lld_read_setup (USBDriver ∗usbp, usbep_t ep, uint8_t ∗buf)

  Reads a setup packet from the dedicated packet buffer.
- void usb_lld_prepare_receive (USBDriver ∗usbp, usbep_t ep)

  Prepares for a receive operation.
- void usb_lld_prepare_transmit (USBDriver ∗usbp, usbep_t ep)

  Prepares for a transmit operation.
- void usb_lld_start_out (USBDriver ∗usbp, usbep_t ep)

  Starts a receive operation on an OUT endpoint.
- void usb_lld_start_in (USBDriver ∗usbp, usbep_t ep)

  Starts a transmit operation on an IN endpoint.
- void usb_lld_stall_out (USBDriver ∗usbp, usbep_t ep)

  Brings an OUT endpoint in the stalled state.
- void usb_lld_stall_in (USBDriver ∗usbp, usbep_t ep)

  Brings an IN endpoint in the stalled state.
- void usb_lld_clear_out (USBDriver ∗usbp, usbep_t ep)

  Brings an OUT endpoint in the active state.
- void usb_lld_clear_in (USBDriver ∗usbp, usbep_t ep)

  Brings an IN endpoint in the active state.

Enumerations

Variables

- USBDriver USBD1

  USB1 driver identifier.
- union {
  USBInEndpointState in
  IN EP0 state.
  USBOutEndpointState out
  OUT EP0 state.
  } ep0_state

  EP0 state.
- static const USBEndpointConfig ep0config

  EP0 initialization structure.

7.39.4 Macro Definition Documentation

7.39.4.1 #define USB_DESC_INDEX(*i*) ((uint8_t)(i))

Helper macro for index values into descriptor strings.

7.39.4.2 #define USB_DESC_BYTE(*b*) ((uint8_t)(b))

Helper macro for byte values into descriptor strings.

7.39.4.3 #define USB_DESC_WORD(*w*)

Value:

```
(uint8_t)((w) & 255U),                                                    \
  (uint8_t)(((w) >> 8) & 255U)
```

Helper macro for word values into descriptor strings.

7.39.4.4 #define USB_DESC_BCD(*bcd*)

Value:

```
(uint8_t)((bcd) & 255U),                                                  \
  (uint8_t)(((bcd) >> 8) & 255)
```

Helper macro for BCD values into descriptor strings.

7.39.4.5 #define USB_DESC_DEVICE(*bcdUSB, bDeviceClass, bDeviceSubClass, bDeviceProtocol, bMaxPacketSize, idVendor, idProduct, bcdDevice, iManufacturer, iProduct, iSerialNumber, bNumConfigurations*)

Value:

```
USB_DESC_BYTE(USB_DESC_DEVICE_SIZE),
      \
  USB_DESC_BYTE(USB_DESCRIPTOR_DEVICE),
      \
  USB_DESC_BCD(bcdUSB),
      \
  USB_DESC_BYTE(bDeviceClass),
      \
  USB_DESC_BYTE(bDeviceSubClass),
      \
  USB_DESC_BYTE(bDeviceProtocol),
      \
  USB_DESC_BYTE(bMaxPacketSize),
      \
  USB_DESC_WORD(idVendor),
      \
  USB_DESC_WORD(idProduct),
      \
  USB_DESC_BCD(bcdDevice),
      \
  USB_DESC_INDEX(iManufacturer),
      \
  USB_DESC_INDEX(iProduct),
      \
  USB_DESC_INDEX(iSerialNumber),
      \
  USB_DESC_BYTE(bNumConfigurations)
```

Device Descriptor helper macro.

7.39.4.6 #define USB_DESC_CONFIGURATION_SIZE 9U

Configuration Descriptor size.

7.39.4.7 #define USB_DESC_CONFIGURATION(*wTotalLength,* *bNumInterfaces,* *bConfigurationValue,* *iConfiguration,* *bmAttributes,* *bMaxPower*)

Value:

```
USB_DESC_BYTE(USB_DESC_CONFIGURATION_SIZE),
             \
  USB_DESC_BYTE(USB_DESCRIPTOR_CONFIGURATION),
      \
  USB_DESC_WORD(wTotalLength),
      \
  USB_DESC_BYTE(bNumInterfaces),
      \
  USB_DESC_BYTE(bConfigurationValue),
      \
  USB_DESC_INDEX(iConfiguration),
      \
  USB_DESC_BYTE(bmAttributes),
      \
  USB_DESC_BYTE(bMaxPower)
```

Configuration Descriptor helper macro.

7.39.4.8 #define USB_DESC_INTERFACE_SIZE 9U

Interface Descriptor size.

7.39.4.9 #define USB_DESC_INTERFACE(*bInterfaceNumber,* *bAlternateSetting,* *bNumEndpoints,* *bInterfaceClass,* *bInterfaceSubClass,* *bInterfaceProtocol,* *iInterface*)

Value:

```
USB_DESC_BYTE(USB_DESC_INTERFACE_SIZE),
         \
  USB_DESC_BYTE(USB_DESCRIPTOR_INTERFACE),
      \
  USB_DESC_BYTE(bInterfaceNumber),
      \
  USB_DESC_BYTE(bAlternateSetting),
      \
  USB_DESC_BYTE(bNumEndpoints),
      \
  USB_DESC_BYTE(bInterfaceClass),
      \
  USB_DESC_BYTE(bInterfaceSubClass),
      \
  USB_DESC_BYTE(bInterfaceProtocol),
      \
  USB_DESC_INDEX(iInterface)
```

Interface Descriptor helper macro.

7.39.4.10 #define USB_DESC_INTERFACE_ASSOCIATION_SIZE 8U

Interface Association Descriptor size.

7.39.4.11 #define USB_DESC_INTERFACE_ASSOCIATION(*bFirstInterface,* *bInterfaceCount,* *bFunctionClass,* *bFunctionSubClass,* *bFunctionProcotol,* *iInterface*)

Value:

```
USB_DESC_BYTE(USB_DESC_INTERFACE_ASSOCIATION_SIZE),
                     \
  USB_DESC_BYTE(USB_DESCRIPTOR_INTERFACE_ASSOCIATION),
      \
  USB_DESC_BYTE(bFirstInterface),
      \
  USB_DESC_BYTE(bInterfaceCount),
      \
  USB_DESC_BYTE(bFunctionClass),
      \
  USB_DESC_BYTE(bFunctionSubClass),
      \
  USB_DESC_BYTE(bFunctionProcotol),
      \
  USB_DESC_INDEX(iInterface)
```

Interface Association Descriptor helper macro.

7.39.4.12 #define USB_DESC_ENDPOINT_SIZE 7U

Endpoint Descriptor size.

7.39.4.13 #define USB_DESC_ENDPOINT(*bEndpointAddress, bmAttributes, wMaxPacketSize, bInterval*)

Value:

```
USB_DESC_BYTE(USB_DESC_ENDPOINT_SIZE),
        \
  USB_DESC_BYTE(USB_DESCRIPTOR_ENDPOINT),
      \
  USB_DESC_BYTE(bEndpointAddress),
      \
  USB_DESC_BYTE(bmAttributes),
      \
  USB_DESC_WORD(wMaxPacketSize),
      \
  USB_DESC_BYTE(bInterval)
```

Endpoint Descriptor helper macro.

7.39.4.14 #define USB_EP_MODE_TYPE 0x0003U

Endpoint type mask.

7.39.4.15 #define USB_EP_MODE_TYPE_CTRL 0x0000U

Control endpoint.

7.39.4.16 #define USB_EP_MODE_TYPE_ISOC 0x0001U

Isochronous endpoint.

7.39.4.17 #define USB_EP_MODE_TYPE_BULK 0x0002U

Bulk endpoint.

7.39.4.18 #define USB_EP_MODE_TYPE_INTR 0x0003U

Interrupt endpoint.

7.39.4.19 #define USB_USE_WAIT FALSE

Enables synchronous APIs.

Note

Disabling this option saves both code and data space.

7.39.4.20 #define usbGetDriverStateI(*usbp*) ((usbp)->state)

Returns the driver state.

Parameters

`in`	*usbp*	pointer to the `USBDriver` object

Returns

The driver state.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.39.4.21 #define usbConnectBus(*usbp*) usb_lld_connect_bus(usbp)

Connects the USB device.

Parameters

`in`	*usbp*	pointer to the `USBDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.39.4.22 #define usbDisconnectBus(*usbp*) usb_lld_disconnect_bus(usbp)

Disconnect the USB device.

Parameters

`in`	*usbp*	pointer to the `USBDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.39.4.23 #define usbGetFrameNumberX(*usbp*) usb_lld_get_frame_number(usbp)

Returns the current frame number.

Parameters

in	*usbp*	pointer to the USBDriver object

Returns

The current frame number.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

7.39.4.24 #define usbGetTransmitStatusI(*usbp*, *ep*) (((usbp)->transmitting & (uint16_t)((unsigned)1U << (unsigned)(ep))) != 0U)

Returns the status of an IN endpoint.

Parameters

in	*usbp*	pointer to the USBDriver object
in	*ep*	endpoint number

Returns

The operation status.

Return values

false	Endpoint ready.
true	Endpoint transmitting.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.39.4.25 #define usbGetReceiveStatusI(*usbp*, *ep*) (((usbp)->receiving & (uint16_t)((unsigned)1U << (unsigned)(ep))) != 0U)

Returns the status of an OUT endpoint.

Parameters

in	*usbp*	pointer to the USBDriver object
in	*ep*	endpoint number

Returns

The operation status.

Return values

false	Endpoint ready.
true	Endpoint receiving.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.39.4.26 #define usbGetReceiveTransactionSizeX(*usbp*, *ep*) usb_lld_get_transaction_size(usbp, ep)

Returns the exact size of a receive transaction.

The received size can be different from the size specified in `usbStartReceiveI()` because the last packet could have a size different from the expected one.

Parameters

`in`	*usbp*	pointer to the `USBDriver` object
`in`	*ep*	endpoint number

Returns

Received data size.

Function Class:

This is an **X-Class** API, this function can be invoked from any context.

7.39.4.27 #define usbSetupTransfer(*usbp*, *buf*, *n*, *endcb*)

Value:

```
{                                                     \
    (usbp)->ep0next   = (buf);                                                \
    (usbp)->ep0n      = (n);                                                  \
    (usbp)->ep0endcb = (endcb);                                               \
}
```

Request transfer setup.

This macro is used by the request handling callbacks in order to prepare a transaction over the endpoint zero.

Parameters

`in`	*usbp*	pointer to the `USBDriver` object
`in`	*buf*	pointer to a buffer for the transaction data
`in`	*n*	number of bytes to be transferred
`in`	*endcb*	callback to be invoked after the transfer or `NULL`

Function Class:

Special function, this function has special requirements see the notes.

7.39.4.28 #define usbReadSetup(*usbp,* *ep,* *buf*) usb_lld_read_setup(usbp, ep, buf)

Reads a setup packet from the dedicated packet buffer.

This function must be invoked in the context of the `setup_cb` callback in order to read the received setup packet.

Precondition

In order to use this function the endpoint must have been initialized as a control endpoint.

Note

This function can be invoked both in thread and IRQ context.

Parameters

in	*usbp*	pointer to the `USBDriver` object
in	*ep*	endpoint number
out	*buf*	buffer where to copy the packet data

Function Class:

Special function, this function has special requirements see the notes.

7.39.4.29 #define _usb_isr_invoke_event_cb(*usbp,* *evt*)

Value:

```
{                                                         \
  if (((usbp)->config->event_cb) != NULL) {                                 \
    (usbp)->config->event_cb(usbp, evt);                                    \
  }                                                                         \
}
```

Common ISR code, usb event callback.

Parameters

in	*usbp*	pointer to the `USBDriver` object
in	*evt*	USB event code

Function Class:

Not an API, this function is for internal use only.

7.39.4.30 #define _usb_isr_invoke_sof_cb(*usbp*)

Value:

```
{                                                                    \
  if (((usbp)->config->sof_cb) != NULL) {                                   \
    (usbp)->config->sof_cb(usbp);                                           \
  }                                                                         \
}
```

Common ISR code, SOF callback.

Parameters

in	*usbp*	pointer to the USBDriver object

Function Class:

Not an API, this function is for internal use only.

7.39.4.31 #define _usb_isr_invoke_setup_cb(*usbp*, *ep*)

Value:

```
{                                              \
  (usbp)->epc[ep]->setup_cb(usbp, ep);                                              \
}
```

Common ISR code, setup packet callback.

Parameters

in	*usbp*	pointer to the USBDriver object
in	*ep*	endpoint number

Function Class:

Not an API, this function is for internal use only.

7.39.4.32 #define _usb_isr_invoke_in_cb(*usbp*, *ep*)

Value:

```
{                                                  \
  (usbp)->transmitting &= ~(1 << (ep));                                            \
  if ((usbp)->epc[ep]->in_cb != NULL) {                                            \
    (usbp)->epc[ep]->in_cb(usbp, ep);                                              \
  }                                                                                \
  osalSysLockFromISR();                                                            \
  osalThreadResumeI(&(usbp)->epc[ep]->in_state->thread, MSG_OK);                   \
  osalSysUnlockFromISR();                                                          \
}
```

Common ISR code, IN endpoint callback.

Parameters

in	*usbp*	pointer to the USBDriver object
in	*ep*	endpoint number

Function Class:

Not an API, this function is for internal use only.

7.39.4.33 #define _usb_isr_invoke_out_cb(*usbp*, *ep*)

Value:

```
{                                   \
  (usbp)->receiving &= ~(1 << (ep));                                        \
  if ((usbp)->epc[ep]->out_cb != NULL) {                                    \
    (usbp)->epc[ep]->out_cb(usbp, ep);                                      \
  }                                                                         \
  osalSysLockFromISR();                                                     \
  osalThreadResumeI(&(usbp)->epc[ep]->out_state->thread,                    \
                    usbGetReceiveTransactionSizeX(usbp, ep));               \
  osalSysUnlockFromISR();                                                   \
}
```

Common ISR code, OUT endpoint event.

Parameters

in	*usbp*	pointer to the `USBDriver` object
in	*ep*	endpoint number

Function Class:

Not an API, this function is for internal use only.

7.39.4.34 #define USB_MAX_ENDPOINTS 4

Maximum endpoint address.

7.39.4.35 #define USB_EP0_STATUS_STAGE USB_EP0_STATUS_STAGE_SW

Status stage handling method.

7.39.4.36 #define USB_SET_ADDRESS_MODE USB_LATE_SET_ADDRESS

The address can be changed immediately upon packet reception.

7.39.4.37 #define USB_SET_ADDRESS_ACK_HANDLING USB_SET_ADDRESS_ACK_SW

Method for set address acknowledge.

7.39.4.38 #define PLATFORM_USB_USE_USB1 FALSE

USB driver enable switch.

If set to `TRUE` the support for USB1 is included.

Note

The default is `FALSE`.

7.39.4.39 #define usb_lld_get_frame_number(*usbp*) 0

Returns the current frame number.

Parameters

in	*usbp*	pointer to the `USBDriver` object

Returns

The current frame number.

Function Class:

Not an API, this function is for internal use only.

7.39.4.40 #define usb_lld_get_transaction_size(*usbp*, *ep*) ((usbp)->epc[ep]->out_state->rxcnt)

Returns the exact size of a receive transaction.

The received size can be different from the size specified in `usbStartReceiveI()` because the last packet could have a size different from the expected one.

Precondition

The OUT endpoint must have been configured in transaction mode in order to use this function.

Parameters

`in`	*usbp*	pointer to the `USBDriver` object
`in`	*ep*	endpoint number

Returns

Received data size.

Function Class:

Not an API, this function is for internal use only.

7.39.4.41 #define usb_lld_connect_bus(*usbp*)

Connects the USB device.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.39.4.42 #define usb_lld_disconnect_bus(*usbp*)

Disconnect the USB device.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.39.5 Typedef Documentation

7.39.5.1 typedef struct USBDriver USBDriver

Type of a structure representing an USB driver.

7.39.5.2 typedef uint8_t usbep_t

Type of an endpoint identifier.

7.39.5.3 typedef void(∗ usbcallback_t) (USBDriver ∗usbp)

Type of an USB generic notification callback.

Parameters

`in`	*usbp*	pointer to the `USBDriver` object triggering the callback

7.39.5.4 typedef void(∗ usbepcallback_t) (USBDriver ∗usbp, usbep_t ep)

Type of an USB endpoint callback.

Parameters

`in`	*usbp*	pointer to the `USBDriver` object triggering the callback
`in`	*ep*	endpoint number

7.39.5.5 typedef void(∗ usbeventcb_t) (USBDriver ∗usbp, usbevent_t event)

Type of an USB event notification callback.

Parameters

`in`	*usbp*	pointer to the `USBDriver` object triggering the callback
`in`	*event*	event type

7.39.5.6 typedef bool(∗ usbreqhandler_t) (USBDriver ∗usbp)

Type of a requests handler callback.

The request is encoded in the `usb_setup` buffer.

Parameters

`in`	*usbp*	pointer to the `USBDriver` object triggering the callback

Returns

The request handling exit code.

Return values

false	Request not recognized by the handler.
true	Request handled.

7.39.5.7 typedef const USBDescriptor∗(∗ usbgetdescriptor_t) (USBDriver ∗usbp, uint8_t dtype, uint8_t dindex, uint16_t lang)

Type of an USB descriptor-retrieving callback.

7.39.6 Enumeration Type Documentation

7.39.6.1 enum usbstate_t

Type of a driver state machine possible states.

Enumerator

USB_UNINIT	Not initialized.
USB_STOP	Stopped.
USB_READY	Ready, after bus reset.
USB_SELECTED	Address assigned.
USB_ACTIVE	Active, configuration selected.
USB_SUSPENDED	Suspended, low power mode.

7.39.6.2 enum usbepstatus_t

Type of an endpoint status.

Enumerator

EP_STATUS_DISABLED	Endpoint not active.
EP_STATUS_STALLED	Endpoint opened but stalled.
EP_STATUS_ACTIVE	Active endpoint.

7.39.6.3 enum usbep0state_t

Type of an endpoint zero state machine states.

Enumerator

USB_EP0_WAITING_SETUP	Waiting for SETUP data.
USB_EP0_TX	Transmitting.
USB_EP0_WAITING_TX0	Waiting transmit 0.
USB_EP0_WAITING_STS	Waiting status.
USB_EP0_RX	Receiving.
USB_EP0_SENDING_STS	Sending status.
USB_EP0_ERROR	Error, EP0 stalled.

7.39.6.4 enum usbevent_t

Type of an enumeration of the possible USB events.

Enumerator

USB_EVENT_RESET	Driver has been reset by host.

USB_EVENT_ADDRESS Address assigned.

USB_EVENT_CONFIGURED Configuration selected.

USB_EVENT_UNCONFIGURED Configuration removed.

USB_EVENT_SUSPEND Entering suspend mode.

USB_EVENT_WAKEUP Leaving suspend mode.

USB_EVENT_STALLED Endpoint 0 error, stalled.

7.39.7 Function Documentation

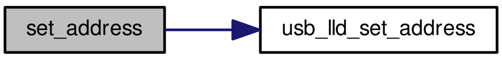
7.39.7.1 static void set_address (**USBDriver** * *usbp*) `[static]`

SET ADDRESS transaction callback.

Parameters

`in`	*usbp*	pointer to the `USBDriver` object

Here is the call graph for this function:



7.39.7.2 static bool default_handler (**USBDriver** * *usbp*) `[static]`

Standard requests handler.

This is the standard requests default handler, most standard requests are handled here, the user can override the standard handling using the `requests_hook_cb` hook in the `USBConfig` structure.

Parameters

`in`	*usbp*	pointer to the `USBDriver` object

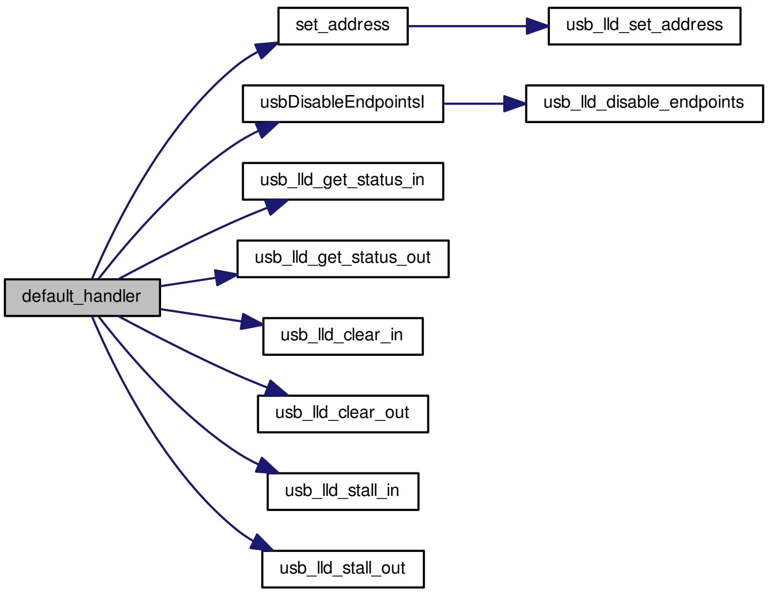
Returns

The request handling exit code.

Return values

false	Request not recognized by the handler or error.
true	Request handled.

Here is the call graph for this function:



7.39.7.3 void usbInit (void)

USB Driver initialization.

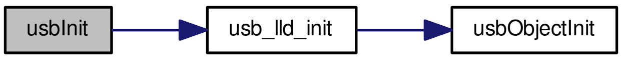
Note

This function is implicitly invoked by halInit(), there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



7.39.7.4 void usbObjectInit (USBDriver * *usbp*)

Initializes the standard part of a USBDriver structure.

Parameters

out	*usbp*	pointer to the USBDriver object

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

7.39.7.5 void usbStart (**USBDriver** ∗ *usbp*, const **USBConfig** ∗ *config*)

Configures and activates the USB peripheral.

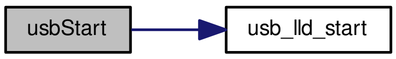
Parameters

in	*usbp*	pointer to the USBDriver object
in	*config*	pointer to the USBConfig object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.39.7.6 void usbStop (**USBDriver** ∗ *usbp*)

Deactivates the USB peripheral.

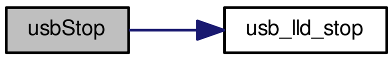
Parameters

in	*usbp*	pointer to the USBDriver object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.39.7.7 void usbInitEndpointI (USBDriver * *usbp*, usbep_t *ep*, const USBEndpointConfig * *epcp*)

Enables an endpoint.

This function enables an endpoint, both IN and/or OUT directions depending on the configuration structure.

Note

This function must be invoked in response of a SET_CONFIGURATION or SET_INTERFACE message.

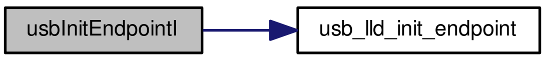
Parameters

in	*usbp*	pointer to the USBDriver object
in	*ep*	endpoint number
in	*epcp*	the endpoint configuration

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



7.39.7.8 void usbDisableEndpointsI (USBDriver * *usbp*)

Disables all the active endpoints.

This function disables all the active endpoints except the endpoint zero.

Note

This function must be invoked in response of a SET_CONFIGURATION message with configuration number zero.

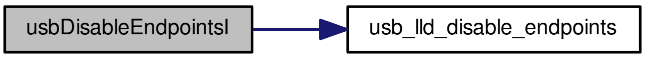
Parameters

`in`	*usbp*	pointer to the `USBDriver` object

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



7.39.7.9 void usbStartReceiveI (USBDriver ∗ *usbp*, usbep_t *ep*, uint8_t ∗ *buf*, size_t *n*)

Starts a receive transaction on an OUT endpoint.

Note

This function is meant to be called from ISR context outside critical zones because there is a potentially slow operation inside.

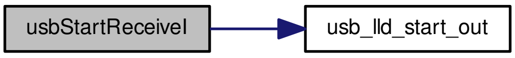
Parameters

`in`	*usbp*	pointer to the `USBDriver` object
`in`	*ep*	endpoint number
`out`	*buf*	buffer where to copy the received data
`in`	*n*	transaction size. It is recommended a multiple of the packet size because the excess is discarded.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



7.39.7.10 void usbStartTransmitI (USBDriver * *usbp*, usbep_t *ep*, const uint8_t * *buf*, size_t *n*)

Starts a transmit transaction on an IN endpoint.

Note

This function is meant to be called from ISR context outside critical zones because there is a potentially slow operation inside.

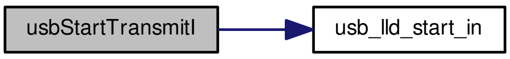
Parameters

in	*usbp*	pointer to the `USBDriver` object
in	*ep*	endpoint number
in	*buf*	buffer where to fetch the data to be transmitted
in	*n*	transaction size

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



7.39.7.11 msg_t usbReceive (USBDriver * *usbp*, usbep_t *ep*, uint8_t * *buf*, size_t *n*)

Performs a receive transaction on an OUT endpoint.

Parameters

in	*usbp*	pointer to the `USBDriver` object
in	*ep*	endpoint number
out	*buf*	buffer where to copy the received data
in	*n*	transaction size. It is recommended a multiple of the packet size because the excess is discarded.

Returns

The received effective data size, it can be less than the amount specified.

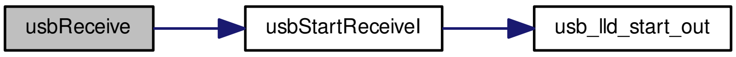
Return values

MSG_RESET	driver not in `USB_ACTIVE` state or the operation has been aborted by an USB reset or a transition to the `USB_SUSPENDED` state.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.39.7.12 msg_t usbTransmit (USBDriver ∗ *usbp*, usbep_t *ep*, const uint8_t ∗ *buf*, size_t *n*)

Performs a transmit transaction on an IN endpoint.

Parameters

`in`	*usbp*	pointer to the `USBDriver` object
`in`	*ep*	endpoint number
`in`	*buf*	buffer where to fetch the data to be transmitted
`in`	*n*	transaction size

Returns

The operation status.

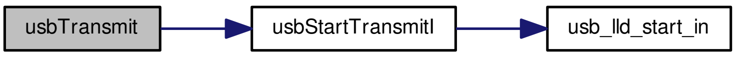
Return values

MSG_OK	operation performed successfully.
MSG_RESET	driver not in `USB_ACTIVE` state or the operation has been aborted by an USB reset or a transition to the `USB_SUSPENDED` state.

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.39.7.13 bool usbStallReceiveI (USBDriver ∗ *usbp*, usbep_t *ep*)

Stalls an OUT endpoint.

Parameters

in	*usbp*	pointer to the USBDriver object
in	*ep*	endpoint number

Returns

The operation status.

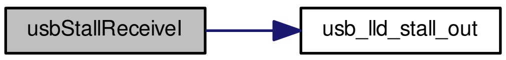
Return values

false	Endpoint stalled.
true	Endpoint busy, not stalled.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



7.39.7.14 bool usbStallTransmitI (USBDriver ∗ *usbp*, usbep_t *ep*)

Stalls an IN endpoint.

Parameters

in	*usbp*	pointer to the USBDriver object
in	*ep*	endpoint number

Returns

The operation status.

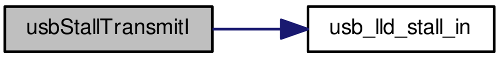
Return values

false	Endpoint stalled.
true	Endpoint busy, not stalled.

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

Here is the call graph for this function:



7.39.7.15 void _usb_reset (**USBDriver** ∗ *usbp*)

USB reset routine.

This function must be invoked when an USB bus reset condition is detected.

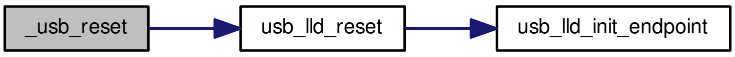
Parameters

in	*usbp*	pointer to the USBDriver object

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.39.7.16 void _usb_suspend (**USBDriver** ∗ *usbp*)

USB suspend routine.

This function must be invoked when an USB bus suspend condition is detected.

Parameters

in	*usbp*	pointer to the USBDriver object

Function Class:

Not an API, this function is for internal use only.

7.39.7.17 void _usb_wakeup (USBDriver * *usbp*)

USB wake-up routine.

This function must be invoked when an USB bus wake-up condition is detected.

Parameters

in	*usbp*	pointer to the `USBDriver` object

Function Class:

Not an API, this function is for internal use only.

7.39.7.18 void _usb_ep0setup (USBDriver * *usbp*, usbep_t *ep*)

Default EP0 SETUP callback.

This function is used by the low level driver as default handler for EP0 SETUP events.

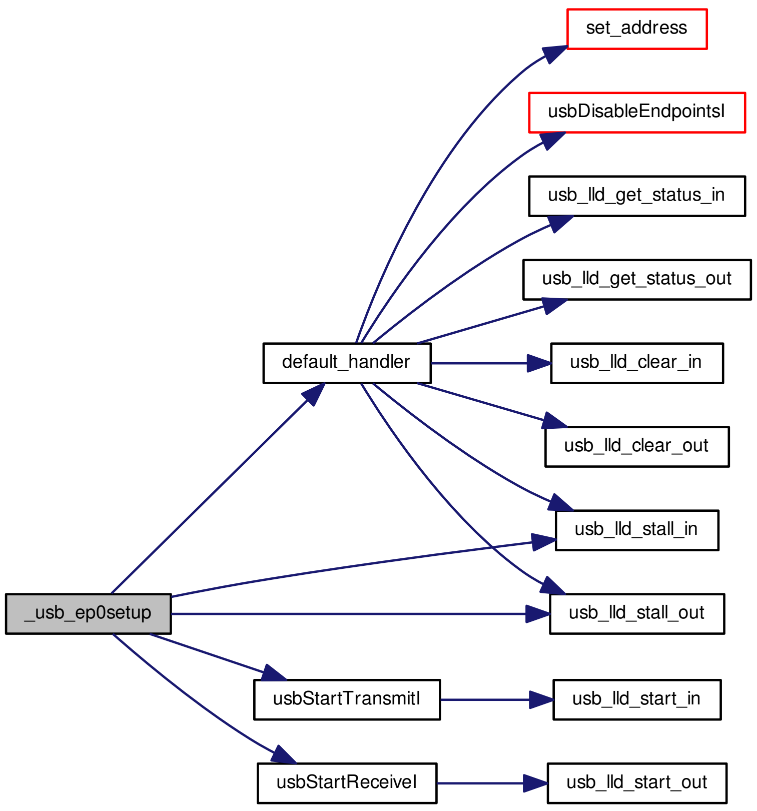
Parameters

in	*usbp*	pointer to the `USBDriver` object
in	*ep*	endpoint number, always zero

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.39.7.19 void _usb_ep0in (**USBDriver** ∗ *usbp,* **usbep_t** *ep*)

Default EP0 IN callback.

This function is used by the low level driver as default handler for EP0 IN events.

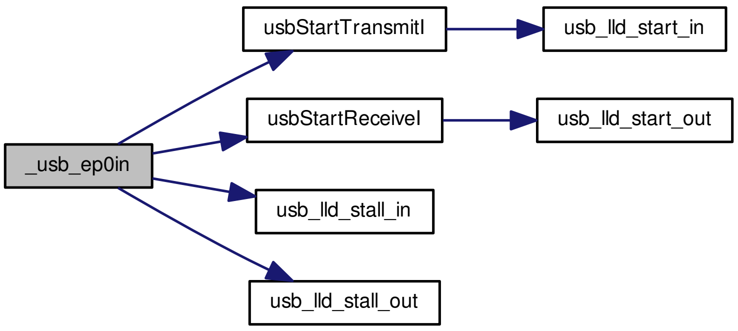
Parameters

`in`	*usbp*	pointer to the `USBDriver` object
`in`	*ep*	endpoint number, always zero

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.39.7.20 void _usb_ep0out (USBDriver * *usbp*, usbep_t *ep*)

Default EP0 OUT callback.

This function is used by the low level driver as default handler for EP0 OUT events.

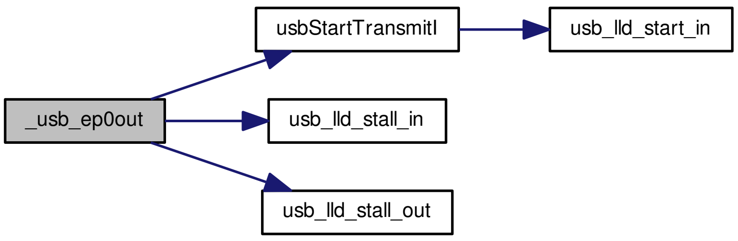
Parameters

in	*usbp*	pointer to the `USBDriver` object
in	*ep*	endpoint number, always zero

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



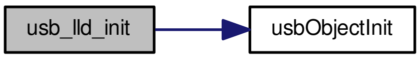
7.39.7.21 void usb_lld_init (void)

Low level USB driver initialization.

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.39.7.22 void usb_lld_start (USBDriver * *usbp*)

Configures and activates the USB peripheral.

Parameters

in	*usbp*	pointer to the `USBDriver` object

Function Class:

Not an API, this function is for internal use only.

7.39.7.23 void usb_lld_stop (USBDriver * *usbp*)

Deactivates the USB peripheral.

Parameters

in	*usbp*	pointer to the `USBDriver` object

Function Class:

Not an API, this function is for internal use only.

7.39.7.24 void usb_lld_reset (USBDriver * *usbp*)

USB low level reset routine.

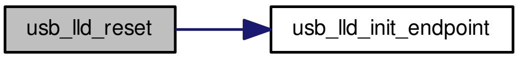
Parameters

in	*usbp*	pointer to the `USBDriver` object

Function Class:

Not an API, this function is for internal use only.

Here is the call graph for this function:



7.39.7.25 void usb_lld_set_address (**USBDriver** ∗ *usbp*)

Sets the USB address.

Parameters

in	*usbp*	pointer to the `USBDriver` object

Function Class:

Not an API, this function is for internal use only.

7.39.7.26 void usb_lld_init_endpoint (**USBDriver** ∗ *usbp,* **usbep_t** *ep*)

Enables an endpoint.

Parameters

in	*usbp*	pointer to the `USBDriver` object
in	*ep*	endpoint number

Function Class:

Not an API, this function is for internal use only.

7.39.7.27 void usb_lld_disable_endpoints (**USBDriver** ∗ *usbp*)

Disables all the active endpoints except the endpoint zero.

Parameters

in	*usbp*	pointer to the `USBDriver` object

Function Class:

Not an API, this function is for internal use only.

7.39.7.28 usbepstatus_t usb_lld_get_status_out (USBDriver ∗ *usbp*, usbep_t *ep*)

Returns the status of an OUT endpoint.

Parameters

in	*usbp*	pointer to the `USBDriver` object
in	*ep*	endpoint number

Returns

The endpoint status.

Return values

EP_STATUS_DISABLED	The endpoint is not active.
EP_STATUS_STALLED	The endpoint is stalled.
EP_STATUS_ACTIVE	The endpoint is active.

Function Class:

Not an API, this function is for internal use only.

7.39.7.29 usbepstatus_t usb_lld_get_status_in (USBDriver ∗ *usbp*, usbep_t *ep*)

Returns the status of an IN endpoint.

Parameters

in	*usbp*	pointer to the `USBDriver` object
in	*ep*	endpoint number

Returns

The endpoint status.

Return values

EP_STATUS_DISABLED	The endpoint is not active.
EP_STATUS_STALLED	The endpoint is stalled.
EP_STATUS_ACTIVE	The endpoint is active.

Function Class:

Not an API, this function is for internal use only.

7.39.7.30 void usb_lld_read_setup (USBDriver ∗ *usbp*, usbep_t *ep*, uint8_t ∗ *buf*)

Reads a setup packet from the dedicated packet buffer.

This function must be invoked in the context of the `setup_cb` callback in order to read the received setup packet.

Precondition

In order to use this function the endpoint must have been initialized as a control endpoint.

Postcondition

The endpoint is ready to accept another packet.

Parameters

`in`	*usbp*	pointer to the `USBDriver` object
`in`	*ep*	endpoint number
`out`	*buf*	buffer where to copy the packet data

Function Class:

Not an API, this function is for internal use only.

7.39.7.31 void usb_lld_prepare_receive (USBDriver ∗ *usbp*, usbep_t *ep*)

Prepares for a receive operation.

Parameters

`in`	*usbp*	pointer to the `USBDriver` object
`in`	*ep*	endpoint number

Function Class:

Not an API, this function is for internal use only.

7.39.7.32 void usb_lld_prepare_transmit (USBDriver ∗ *usbp*, usbep_t *ep*)

Prepares for a transmit operation.

Parameters

`in`	*usbp*	pointer to the `USBDriver` object
`in`	*ep*	endpoint number

Function Class:

Not an API, this function is for internal use only.

7.39.7.33 void usb_lld_start_out (USBDriver ∗ *usbp*, usbep_t *ep*)

Starts a receive operation on an OUT endpoint.

Parameters

`in`	*usbp*	pointer to the `USBDriver` object
`in`	*ep*	endpoint number

Function Class:

Not an API, this function is for internal use only.

7.39.7.34 void usb_lld_start_in (USBDriver ∗ *usbp,* usbep_t *ep*)

Starts a transmit operation on an IN endpoint.

Parameters

`in`	*usbp*	pointer to the `USBDriver` object
`in`	*ep*	endpoint number

Function Class:

Not an API, this function is for internal use only.

7.39.7.35 void usb_lld_stall_out (USBDriver ∗ *usbp,* usbep_t *ep*)

Brings an OUT endpoint in the stalled state.

Parameters

`in`	*usbp*	pointer to the `USBDriver` object
`in`	*ep*	endpoint number

Function Class:

Not an API, this function is for internal use only.

7.39.7.36 void usb_lld_stall_in (USBDriver ∗ *usbp,* usbep_t *ep*)

Brings an IN endpoint in the stalled state.

Parameters

`in`	*usbp*	pointer to the `USBDriver` object
`in`	*ep*	endpoint number

Function Class:

Not an API, this function is for internal use only.

7.39.7.37 void usb_lld_clear_out (USBDriver ∗ *usbp,* usbep_t *ep*)

Brings an OUT endpoint in the active state.

Parameters

in	*usbp*	pointer to the USBDriver object
in	*ep*	endpoint number

Function Class:

Not an API, this function is for internal use only.

7.39.7.38 void usb_lld_clear_in (USBDriver ∗ *usbp,* usbep_t *ep*)

Brings an IN endpoint in the active state.

Parameters

in	*usbp*	pointer to the USBDriver object
in	*ep*	endpoint number

Function Class:

Not an API, this function is for internal use only.

7.39.8 Variable Documentation

7.39.8.1 USBDriver USBD1

USB1 driver identifier.

7.39.8.2 union { ... } ep0_state [static]

EP0 state.

Note

It is an union because IN and OUT endpoints are never used at the same time for EP0.

7.39.8.3 USBInEndpointState { ... } in

IN EP0 state.

7.39.8.4 USBOutEndpointState { ... } out

OUT EP0 state.

7.39.8.5 const USBEndpointConfig ep0config [static]

Initial value:

```
= {
  USB_EP_MODE_TYPE_CTRL,
  _usb_ep0setup,
  _usb_ep0in,
  _usb_ep0out,
  0x40,
  0x40,
  &ep0_state.in,
  &ep0_state.out
}
```

EP0 initialization structure.

7.40 USB CDC Header

USB CDC Support Header.

7.40.1 Detailed Description

USB CDC Support Header.

This header contains definitions and types related to USB CDC.

CDC specific messages.

- #define **CDC_SEND_ENCAPSULATED_COMMAND** 0x00U
- #define **CDC_GET_ENCAPSULATED_RESPONSE** 0x01U
- #define **CDC_SET_COMM_FEATURE** 0x02U
- #define **CDC_GET_COMM_FEATURE** 0x03U
- #define **CDC_CLEAR_COMM_FEATURE** 0x04U
- #define **CDC_SET_AUX_LINE_STATE** 0x10U
- #define **CDC_SET_HOOK_STATE** 0x11U
- #define **CDC_PULSE_SETUP** 0x12U
- #define **CDC_SEND_PULSE** 0x13U
- #define **CDC_SET_PULSE_TIME** 0x14U
- #define **CDC_RING_AUX_JACK** 0x15U
- #define **CDC_SET_LINE_CODING** 0x20U
- #define **CDC_GET_LINE_CODING** 0x21U
- #define **CDC_SET_CONTROL_LINE_STATE** 0x22U
- #define **CDC_SEND_BREAK** 0x23U
- #define **CDC_SET_RINGER_PARMS** 0x30U
- #define **CDC_GET_RINGER_PARMS** 0x31U
- #define **CDC_SET_OPERATION_PARMS** 0x32U
- #define **CDC_GET_OPERATION_PARMS** 0x33U

CDC classes

- #define **CDC_COMMUNICATION_INTERFACE_CLASS** 0x02U
- #define **CDC_DATA_INTERFACE_CLASS** 0x0AU

CDC subclasses

- #define **CDC_ABSTRACT_CONTROL_MODEL** 0x02U

CDC descriptors

- #define **CDC_CS_INTERFACE** 0x24U

CDC subdescriptors

- #define **CDC_HEADER** 0x00U
- #define **CDC_CALL_MANAGEMENT** 0x01U
- #define **CDC_ABSTRACT_CONTROL_MANAGEMENT** 0x02U
- #define **CDC_UNION** 0x06U

Line Control bit definitions.

- #define **LC_STOP_1** 0U
- #define **LC_STOP_1P5** 1U
- #define **LC_STOP_2** 2U
- #define **LC_PARITY_NONE** 0U
- #define **LC_PARITY_ODD** 1U
- #define **LC_PARITY_EVEN** 2U
- #define **LC_PARITY_MARK** 3U
- #define **LC_PARITY_SPACE** 4U

Data Structures

- struct cdc_linecoding_t

  Type of Line Coding structure.

7.41 WDG Driver

Generic WDG Driver.

7.41.1 Detailed Description

Generic WDG Driver.

This module defines an abstract interface for a watchdog timer.

Precondition

In order to use the WDG driver the `HAL_USE_WDG` option must be enabled in `halconf.h`.

Macros

- #define wdgResetI(wdgp) wdg_lld_reset(wdgp)

    Resets WDG's counter.

Configuration options

- #define PLATFORM_WDG_USE_WDG1 FALSE

    WDG1 driver enable switch.

Typedefs

- typedef struct WDGDriver WDGDriver

    Type of a structure representing an WDG driver.

Data Structures

- struct WDGConfig

    Driver configuration structure.
- struct WDGDriver

    Structure representing an WDG driver.

Functions

- void wdgInit (void)

    WDG Driver initialization.
- void wdgStart (WDGDriver ∗wdgp, const WDGConfig ∗config)

    Configures and activates the WDG peripheral.
- void wdgStop (WDGDriver ∗wdgp)

    Deactivates the WDG peripheral.
- void wdgReset (WDGDriver ∗wdgp)

    Resets WDG's counter.
- void wdg_lld_init (void)

    Low level WDG driver initialization.
- void wdg_lld_start (WDGDriver ∗wdgp)

    Configures and activates the WDG peripheral.
- void wdg_lld_stop (WDGDriver ∗wdgp)

Deactivates the WDG peripheral.
- void wdg_lld_reset (WDGDriver *wdgp)

  Reloads WDG's counter.

Enumerations

7.41.2 Macro Definition Documentation

7.41.2.1 #define wdgResetI(*wdgp*) wdg_lld_reset(wdgp)

Resets WDG's counter.

Parameters

in	*wdgp*	pointer to the `WDGDriver` object

Function Class:

This is an **I-Class** API, this function can be invoked from within a system lock zone by both threads and interrupt handlers.

7.41.2.2 #define PLATFORM_WDG_USE_WDG1 FALSE

WDG1 driver enable switch.

Note

The default is `FALSE`.

7.41.3 Typedef Documentation

7.41.3.1 typedef struct WDGDriver WDGDriver

Type of a structure representing an WDG driver.

7.41.4 Enumeration Type Documentation

7.41.4.1 enum wdgstate_t

Driver state machine possible states.

Enumerator

WDG_UNINIT	Not initialized.
WDG_STOP	Stopped.
WDG_READY	Ready.

7.41.5 Function Documentation

7.41.5.1 void wdgInit (void)

WDG Driver initialization.

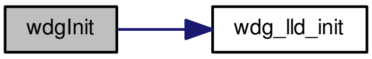
Note

This function is implicitly invoked by `halInit()`, there is no need to explicitly initialize the driver.

Function Class:

Initializer, this function just initializes an object and can be invoked before the kernel is initialized.

Here is the call graph for this function:



7.41.5.2 void wdgStart (**WDGDriver** ∗ *wdgp*, const **WDGConfig** ∗ *config*)

Configures and activates the WDG peripheral.

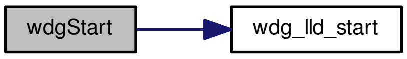
Parameters

`in`	*wdgp*	pointer to the `WDGDriver` object
`in`	*config*	pointer to the `WDGConfig` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.41.5.3 void wdgStop (**WDGDriver** ∗ *wdgp*)

Deactivates the WDG peripheral.

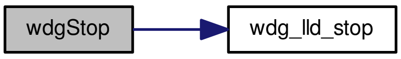
Parameters

`in`	*wdgp*	pointer to the `WDGDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

Here is the call graph for this function:



7.41.5.4 void wdgReset (WDGDriver ∗ *wdgp*)

Resets WDG's counter.

Parameters

in	*wdgp*	pointer to the `WDGDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.41.5.5 void wdg_lld_init (void)

Low level WDG driver initialization.

Function Class:

Not an API, this function is for internal use only.

7.41.5.6 void wdg_lld_start (WDGDriver ∗ *wdgp*)

Configures and activates the WDG peripheral.

Parameters

in	*wdgp*	pointer to the `WDGDriver` object

Function Class:

Not an API, this function is for internal use only.

7.41.5.7 void wdg_lld_stop (WDGDriver ∗ *wdgp*)

Deactivates the WDG peripheral.

Parameters

in	*wdgp*	pointer to the `WDGDriver` object

Function Class:

Normal API, this function can be invoked by regular system threads but not from within a lock zone.

7.41.5.8 void wdg_lld_reset (**WDGDriver** ∗ *wdgp*)

Reloads WDG's counter.

Parameters

in	*wdgp*	pointer to the `WDGDriver` object

Function Class:

Not an API, this function is for internal use only.

Chapter 8

Data Structure Documentation

8.1 ADCConfig Struct Reference

Driver configuration structure.

`#include <hal_adc_lld.h>`

Collaboration diagram for ADCConfig:

8.1.1 Detailed Description

Driver configuration structure.

Note

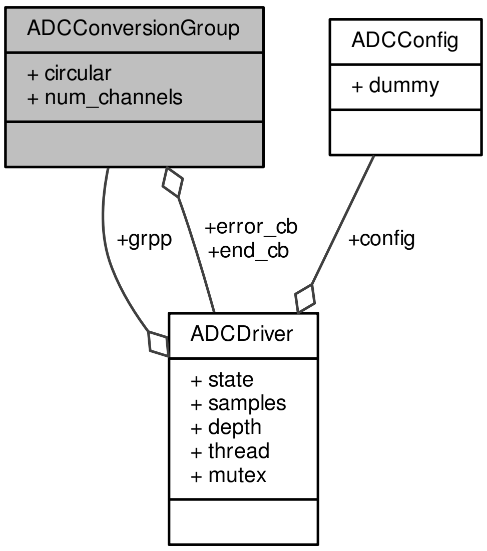
It could be empty on some architectures.

8.2 ADCConversionGroup Struct Reference

Conversion group configuration structure.

`#include <hal_adc_lld.h>`

Collaboration diagram for ADCConversionGroup:



Data Fields

- bool circular

  Enables the circular buffer mode for the group.
- adc_channels_num_t num_channels

  Number of the analog channels belonging to the conversion group.
- adccallback_t end_cb

  Callback function associated to the group or `NULL`.
- adcerrorcallback_t error_cb

  Error callback or `NULL`.

8.2.1 Detailed Description

Conversion group configuration structure.

This implementation-dependent structure describes a conversion operation.

Note

The use of this configuration structure requires knowledge of PLATFORM ADC cell registers interface, please refer to the PLATFORM reference manual for details.

8.2.2 Field Documentation

8.2.2.1 bool ADCConversionGroup::circular

Enables the circular buffer mode for the group.

8.2.2.2 adc_channels_num_t ADCConversionGroup::num_channels

Number of the analog channels belonging to the conversion group.

8.2.2.3 adccallback_t ADCConversionGroup::end_cb

Callback function associated to the group or `NULL`.

8.2.2.4 adcerrorcallback_t ADCConversionGroup::error_cb

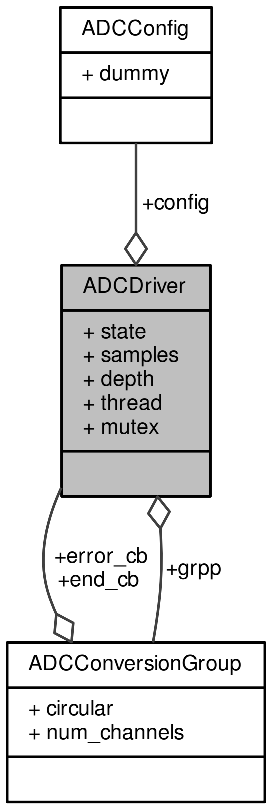
Error callback or `NULL`.

8.3 ADCDriver Struct Reference

Structure representing an ADC driver.

`#include <hal_adc_lld.h>`

Collaboration diagram for ADCDriver:



Data Fields

- adcstate_t state

  Driver state.
- const ADCConfig ∗ config

  Current configuration data.
- adcsample_t ∗ samples

  Current samples buffer pointer or `NULL`.
- size_t depth

  Current samples buffer depth or `0`.
- const ADCConversionGroup ∗ grpp

  Current conversion group pointer or `NULL`.
- thread_reference_t thread

  Waiting thread.
- mutex_t mutex

  Mutex protecting the peripheral.

8.3.1 Detailed Description

Structure representing an ADC driver.

8.3.2 Field Documentation

8.3.2.1 adcstate_t ADCDriver::state

Driver state.

8.3.2.2 const ADCConfig* ADCDriver::config

Current configuration data.

8.3.2.3 adcsample_t* ADCDriver::samples

Current samples buffer pointer or `NULL`.

8.3.2.4 size_t ADCDriver::depth

Current samples buffer depth or `0`.

8.3.2.5 const ADCConversionGroup* ADCDriver::grpp

Current conversion group pointer or `NULL`.

8.3.2.6 thread_reference_t ADCDriver::thread

Waiting thread.

8.3.2.7 mutex_t ADCDriver::mutex

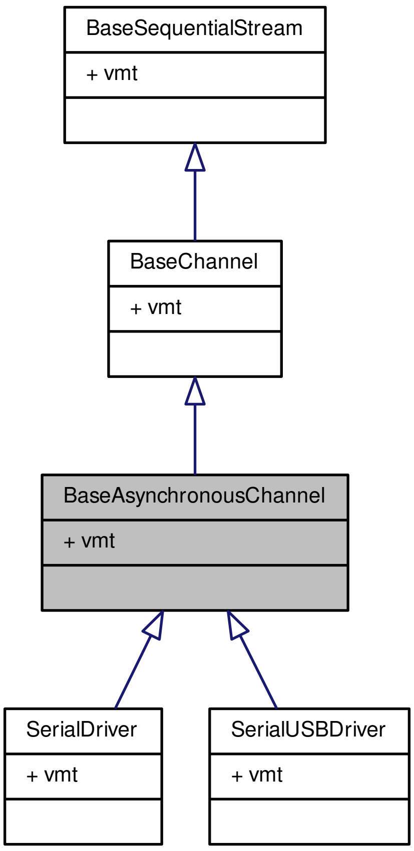
Mutex protecting the peripheral.

8.4 BaseAsynchronousChannel Struct Reference

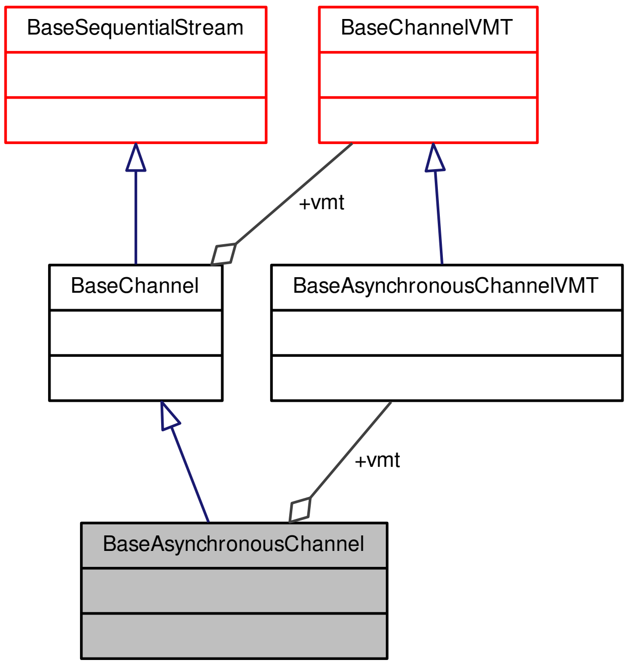
Base asynchronous channel class.

```
#include <hal_channels.h>
```

Inheritance diagram for BaseAsynchronousChannel:



Collaboration diagram for BaseAsynchronousChannel:



Data Fields

- const struct BaseAsynchronousChannelVMT ∗ vmt

    Virtual Methods Table.

8.4.1 Detailed Description

Base asynchronous channel class.

This class extends `BaseChannel` by adding event sources fields for asynchronous I/O for use in an event-driven environment.

8.4.2 Field Documentation

8.4.2.1 const struct BaseAsynchronousChannelVMT∗ BaseAsynchronousChannel::vmt

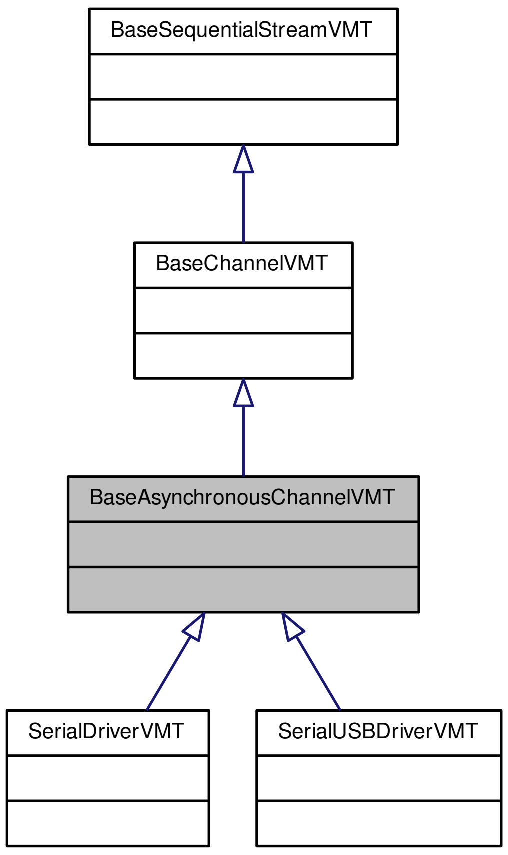
Virtual Methods Table.

8.5 BaseAsynchronousChannelVMT Struct Reference

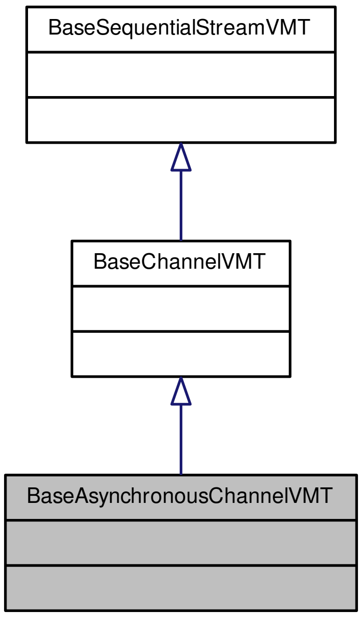
`BaseAsynchronousChannel` virtual methods table.

`#include <hal_channels.h>`

Inheritance diagram for BaseAsynchronousChannelVMT:



Collaboration diagram for BaseAsynchronousChannelVMT:



8.5.1 Detailed Description

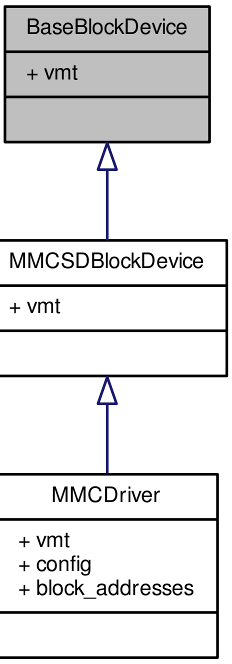
`BaseAsynchronousChannel` virtual methods table.

8.6 BaseBlockDevice Struct Reference

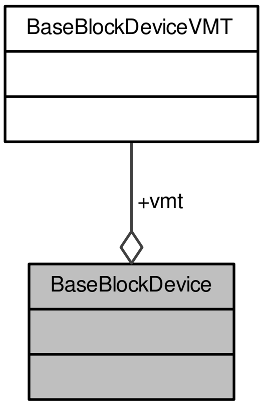
Base block device class.

`#include <hal_ioblock.h>`

Inheritance diagram for BaseBlockDevice:



Collaboration diagram for BaseBlockDevice:



Data Fields

- const struct BaseBlockDeviceVMT $*$ vmt

Virtual Methods Table.

8.6.1 Detailed Description

Base block device class.

This class represents a generic, block-accessible, device.

8.6.2 Field Documentation

8.6.2.1 const struct BaseBlockDeviceVMT∗ BaseBlockDevice::vmt

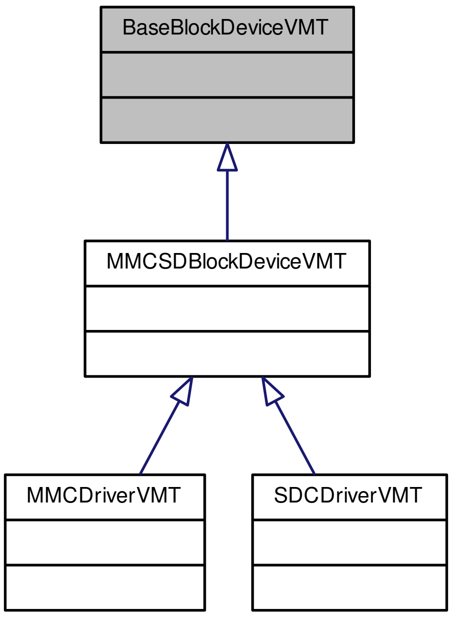
Virtual Methods Table.

8.7 BaseBlockDeviceVMT Struct Reference

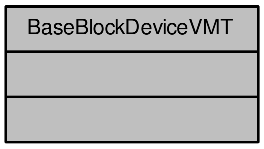
`BaseBlockDevice` virtual methods table.

`#include <hal_ioblock.h>`

Inheritance diagram for BaseBlockDeviceVMT:



Collaboration diagram for BaseBlockDeviceVMT:



8.7.1 Detailed Description

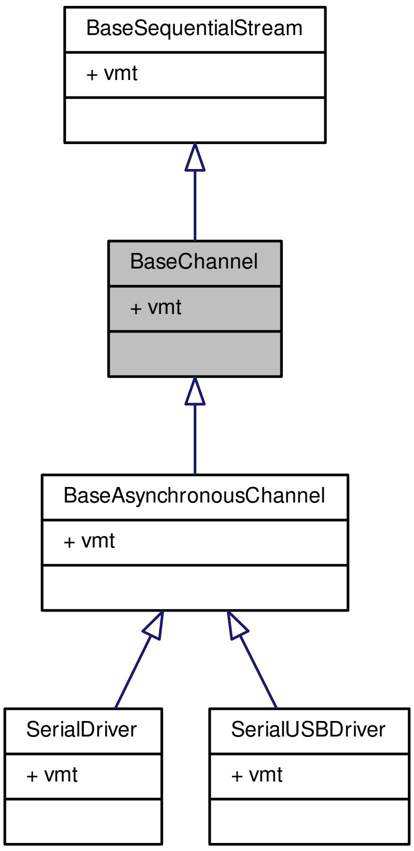
`BaseBlockDevice` virtual methods table.

8.8 BaseChannel Struct Reference

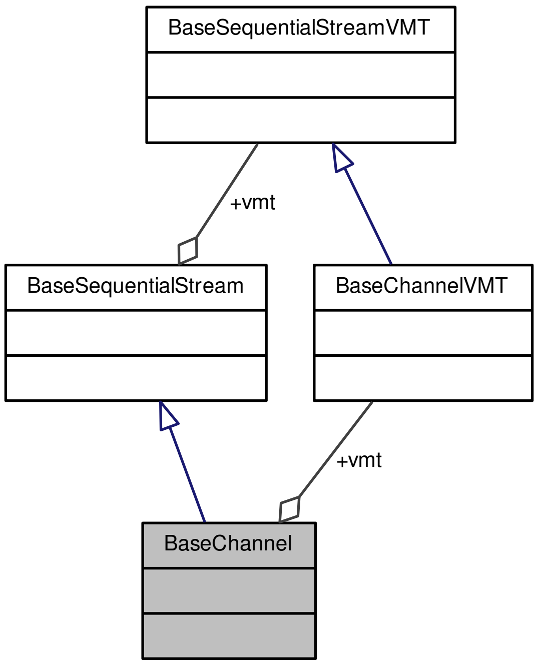
Base channel class.

`#include <hal_channels.h>`

Inheritance diagram for BaseChannel:



Collaboration diagram for BaseChannel:



Data Fields

- const struct BaseChannelVMT ∗ vmt

  Virtual Methods Table.

8.8.1 Detailed Description

Base channel class.

This class represents a generic, byte-wide, I/O channel. This class introduces generic I/O primitives with timeout specification.

8.8.2 Field Documentation

8.8.2.1 const struct BaseChannelVMT∗ BaseChannel::vmt

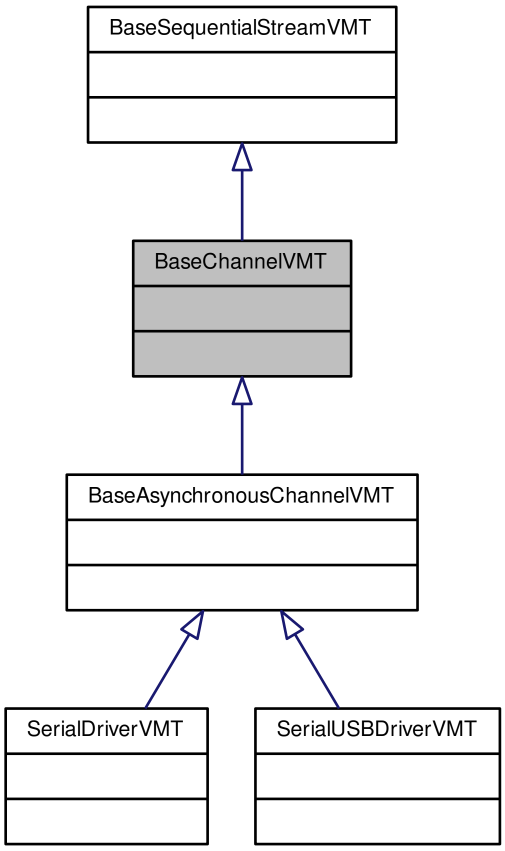
Virtual Methods Table.

8.9 BaseChannelVMT Struct Reference

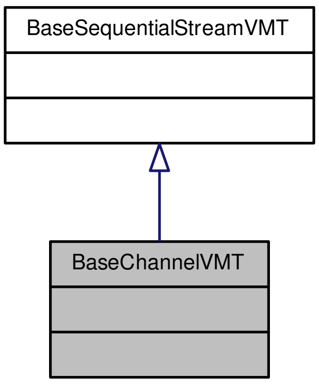
`BaseChannel` virtual methods table.

`#include <hal_channels.h>`

Inheritance diagram for BaseChannelVMT:



Collaboration diagram for BaseChannelVMT:



8.9.1 Detailed Description

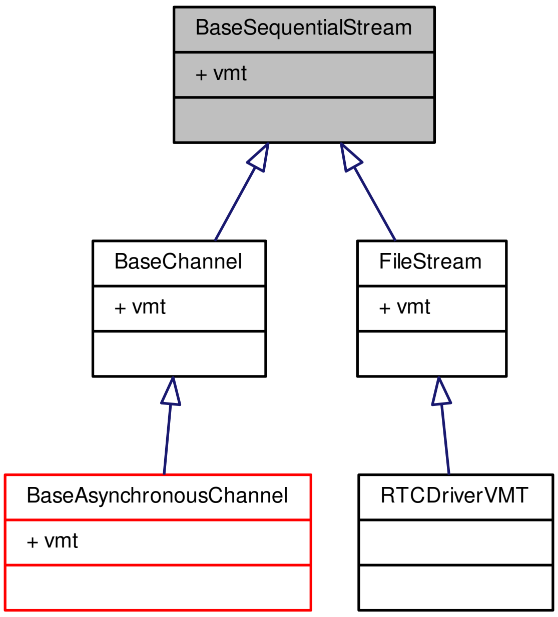
`BaseChannel` virtual methods table.

8.10 BaseSequentialStream Struct Reference

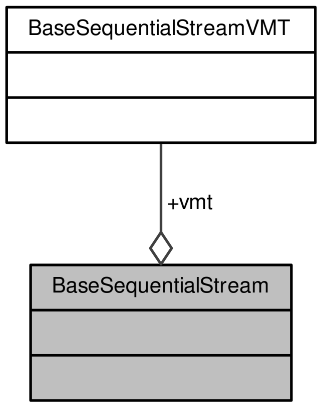
Base stream class.

`#include <hal_streams.h>`

Inheritance diagram for BaseSequentialStream:



Collaboration diagram for BaseSequentialStream:



Data Fields

- const struct BaseSequentialStreamVMT ∗ vmt

  Virtual Methods Table.

8.10.1 Detailed Description

Base stream class.

This class represents a generic blocking unbuffered sequential data stream.

8.10.2 Field Documentation

8.10.2.1 const struct **BaseSequentialStreamVMT**∗ BaseSequentialStream::vmt

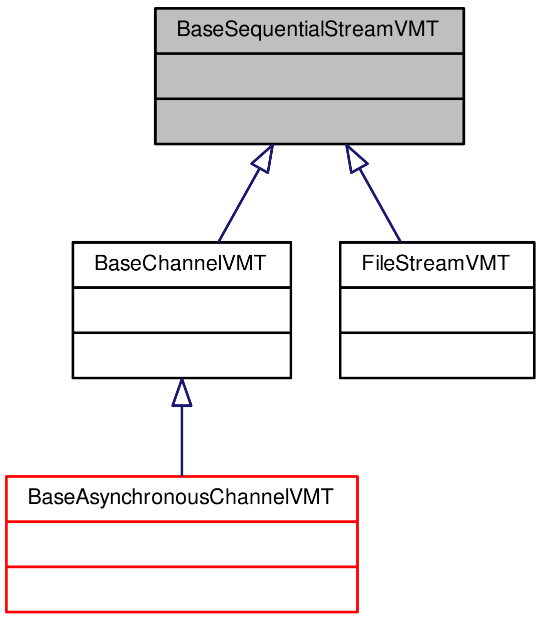
Virtual Methods Table.

8.11 BaseSequentialStreamVMT Struct Reference

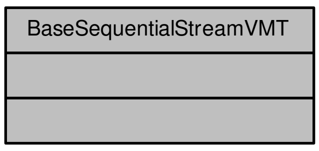
`BaseSequentialStream` virtual methods table.

`#include <hal_streams.h>`

Inheritance diagram for BaseSequentialStreamVMT:



Collaboration diagram for BaseSequentialStreamVMT:



8.11.1 Detailed Description

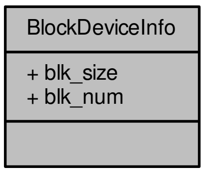
`BaseSequentialStream` virtual methods table.

8.12 BlockDeviceInfo Struct Reference

Block device info.

`#include <hal_ioblock.h>`

Collaboration diagram for BlockDeviceInfo:



Data Fields

- uint32_t blk_size

  Block size in bytes.
- uint32_t blk_num

  Total number of blocks.

8.12.1 Detailed Description

Block device info.

8.12.2 Field Documentation

8.12.2.1 uint32_t BlockDeviceInfo::blk_size

Block size in bytes.

8.12.2.2 uint32_t BlockDeviceInfo::blk_num

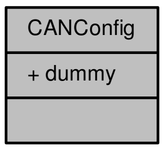
Total number of blocks.

8.13 CANConfig Struct Reference

Driver configuration structure.

```
#include <hal_can_lld.h>
```

Collaboration diagram for CANConfig:



8.13.1 Detailed Description

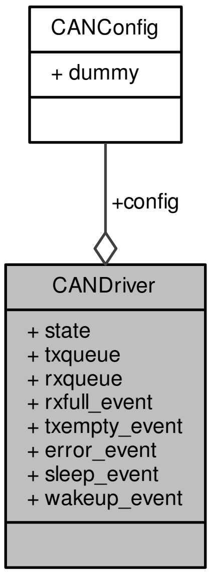
Driver configuration structure.

8.14 CANDriver Struct Reference

Structure representing an CAN driver.

```
#include <hal_can_lld.h>
```

Collaboration diagram for CANDriver:



Data Fields

- canstate_t state

Driver state.

- const CANConfig ∗ config

  Current configuration data.
- threads_queue_t txqueue

  Transmission threads queue.
- threads_queue_t rxqueue

  Receive threads queue.
- event_source_t rxfull_event

  One or more frames become available.
- event_source_t txempty_event

  One or more transmission mailbox become available.
- event_source_t error_event

  A CAN bus error happened.
- event_source_t sleep_event

  Entering sleep state event.
- event_source_t wakeup_event

  Exiting sleep state event.

8.14.1 Detailed Description

Structure representing an CAN driver.

8.14.2 Field Documentation

8.14.2.1 canstate_t CANDriver::state

Driver state.

8.14.2.2 const CANConfig∗ CANDriver::config

Current configuration data.

8.14.2.3 threads_queue_t CANDriver::txqueue

Transmission threads queue.

8.14.2.4 threads_queue_t CANDriver::rxqueue

Receive threads queue.

8.14.2.5 event_source_t CANDriver::rxfull_event

One or more frames become available.

Note

After broadcasting this event it will not be broadcasted again until the received frames queue has been completely emptied. It is **not** broadcasted for each received frame. It is responsibility of the application to empty the queue by repeatedly invoking `chReceive()` when listening to this event. This behavior minimizes the interrupt served by the system because CAN traffic.
The flags associated to the listeners will indicate which receive mailboxes become non-empty.

8.14.2.6 event_source_t CANDriver::txempty_event

One or more transmission mailbox become available.

Note

The flags associated to the listeners will indicate which transmit mailboxes become empty.

8.14.2.7 event_source_t CANDriver::error_event

A CAN bus error happened.

Note

The flags associated to the listeners will indicate the error(s) that have occurred.

8.14.2.8 event_source_t CANDriver::sleep_event

Entering sleep state event.

8.14.2.9 event_source_t CANDriver::wakeup_event

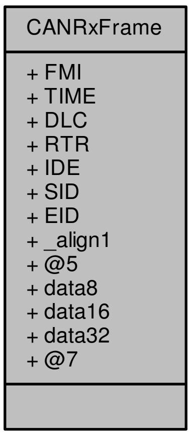
Exiting sleep state event.

8.15 CANRxFrame Struct Reference

CAN received frame.

`#include <hal_can_lld.h>`

Collaboration diagram for CANRxFrame:



Data Fields

- uint8_t FMI

  Filter id.
- uint16_t TIME

  Time stamp.
- uint8_t DLC:4

  Data length.
- uint8_t RTR:1

  Frame type.
- uint8_t IDE:1

  Identifier type.
- uint32_t SID:11

  Standard identifier.
- uint32_t EID:29

  Extended identifier.
- uint8_t data8 [8]

  Frame data.
- uint16_t data16 [4]

  Frame data.
- uint32_t data32 [2]

  Frame data.

8.15.1 Detailed Description

CAN received frame.

Note

Accessing the frame data as word16 or word32 is not portable because machine data endianness, it can be still useful for a quick filling.

8.15.2 Field Documentation

8.15.2.1 uint8_t CANRxFrame::FMI

Filter id.

8.15.2.2 uint16_t CANRxFrame::TIME

Time stamp.

8.15.2.3 uint8_t CANRxFrame::DLC

Data length.

8.15.2.4 uint8_t CANRxFrame::RTR

Frame type.

8.15.2.5 uint8_t CANRxFrame::IDE

Identifier type.

8.15.2.6 uint32_t CANRxFrame::SID

Standard identifier.

8.15.2.7 uint32_t CANRxFrame::EID

Extended identifier.

8.15.2.8 uint8_t CANRxFrame::data8[8]

Frame data.

8.15.2.9 uint16_t CANRxFrame::data16[4]

Frame data.

8.15.2.10 uint32_t CANRxFrame::data32[2]

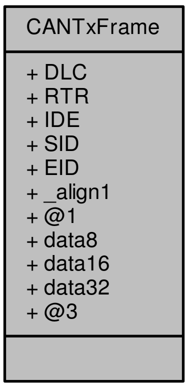
Frame data.

8.16 CANTxFrame Struct Reference

CAN transmission frame.

`#include <hal_can_lld.h>`

Collaboration diagram for CANTxFrame:



Data Fields

- uint8_t DLC:4

    Data length.
- uint8_t RTR:1

    Frame type.
- uint8_t IDE:1

    Identifier type.
- uint32_t SID:11

    Standard identifier.
- uint32_t EID:29

    Extended identifier.
- uint8_t data8 [8]

    Frame data.
- uint16_t data16 [4]

    Frame data.
- uint32_t data32 [2]

    Frame data.

8.16.1 Detailed Description

CAN transmission frame.

Note

Accessing the frame data as word16 or word32 is not portable because machine data endianness, it can be still useful for a quick filling.

8.16.2 Field Documentation

8.16.2.1 uint8_t CANTxFrame::DLC

Data length.

8.16.2.2 uint8_t CANTxFrame::RTR

Frame type.

8.16.2.3 uint8_t CANTxFrame::IDE

Identifier type.

8.16.2.4 uint32_t CANTxFrame::SID

Standard identifier.

8.16.2.5 uint32_t CANTxFrame::EID

Extended identifier.

8.16.2.6 uint8_t CANTxFrame::data8[8]

Frame data.

8.16.2.7 uint16_t CANTxFrame::data16[4]

Frame data.

8.16.2.8 uint32_t CANTxFrame::data32[2]

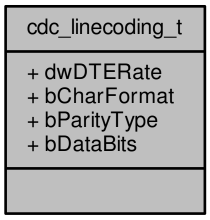
Frame data.

8.17 cdc_linecoding_t Struct Reference

Type of Line Coding structure.

`#include <hal_usb_cdc.h>`

Collaboration diagram for cdc_linecoding_t:



8.17.1 Detailed Description

Type of Line Coding structure.

8.18 DACConfig Struct Reference

Driver configuration structure.

`#include <hal_dac_lld.h>`

Collaboration diagram for DACConfig:

DACConfig
+ dummy

8.18.1 Detailed Description

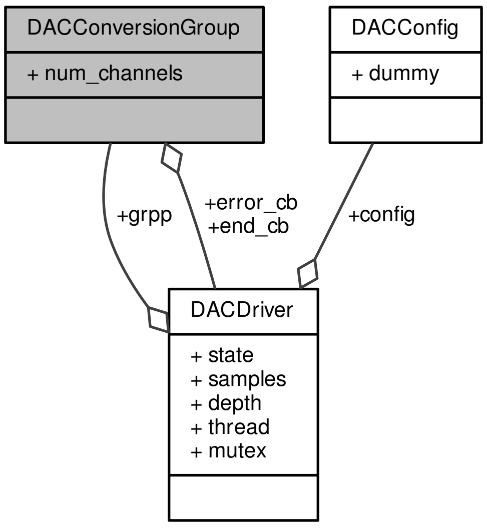
Driver configuration structure.

8.19 DACConversionGroup Struct Reference

DAC Conversion group structure.

```
#include <hal_dac_lld.h>
```

Collaboration diagram for DACConversionGroup:



Data Fields

- uint32_t num_channels

  Number of DAC channels.

- daccallback_t end_cb

    Operation complete callback or `NULL`*.*

- dacerrorcallback_t error_cb

    Error handling callback or `NULL`*.*

8.19.1 Detailed Description

DAC Conversion group structure.

8.19.2 Field Documentation

8.19.2.1 uint32_t DACConversionGroup::num_channels

Number of DAC channels.

8.19.2.2 daccallback_t DACConversionGroup::end_cb

Operation complete callback or `NULL`.

8.19.2.3 dacerrorcallback_t DACConversionGroup::error_cb

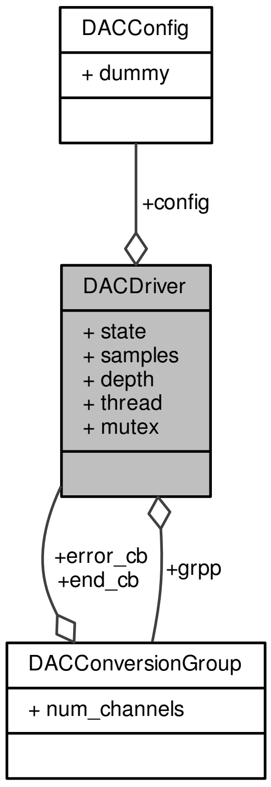
Error handling callback or `NULL`.

8.20 DACDriver Struct Reference

Structure representing a DAC driver.

```
#include <hal_dac_lld.h>
```

Collaboration diagram for DACDriver:



Data Fields

- dacstate_t state

  Driver state.
- const DACConversionGroup ∗ grpp

  Conversion group.
- dacsample_t ∗ samples

  Samples buffer pointer.
- uint16_t depth

  Samples buffer size.
- const DACConfig ∗ config

  Current configuration data.
- thread_reference_t thread

  Waiting thread.
- mutex_t mutex

  Mutex protecting the bus.

8.20.1 Detailed Description

Structure representing a DAC driver.

8.20.2 Field Documentation

8.20.2.1 dacstate_t DACDriver::state

Driver state.

8.20.2.2 const DACConversionGroup* DACDriver::grpp

Conversion group.

8.20.2.3 dacsample_t* DACDriver::samples

Samples buffer pointer.

8.20.2.4 uint16_t DACDriver::depth

Samples buffer size.

8.20.2.5 const DACConfig* DACDriver::config

Current configuration data.

8.20.2.6 thread_reference_t DACDriver::thread

Waiting thread.

8.20.2.7 mutex_t DACDriver::mutex

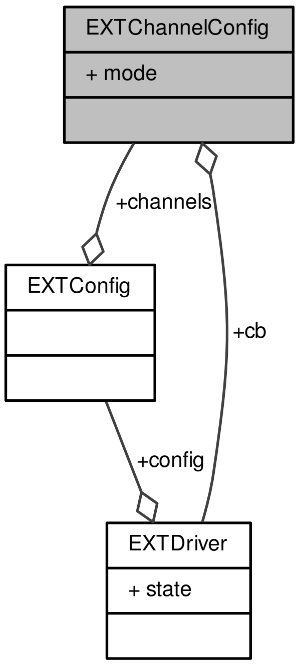
Mutex protecting the bus.

8.21 EXTChannelConfig Struct Reference

Channel configuration structure.

```
#include <hal_ext_lld.h>
```

Collaboration diagram for EXTChannelConfig:



Data Fields

- uint32_t mode

  Channel mode.
- extcallback_t cb

  Channel callback.

8.21.1 Detailed Description

Channel configuration structure.

8.21.2 Field Documentation

8.21.2.1 uint32_t EXTChannelConfig::mode

Channel mode.

8.21.2.2 extcallback_t EXTChannelConfig::cb

Channel callback.

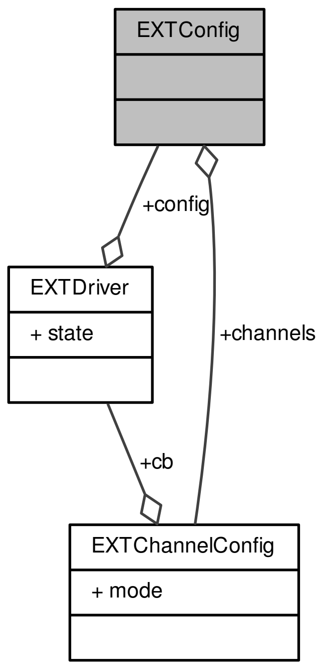
In the STM32 implementation a `NULL` callback pointer is valid and configures the channel as an event sources instead of an interrupt source.

8.22 EXTConfig Struct Reference

Driver configuration structure.

`#include <hal_ext_lld.h>`

Collaboration diagram for EXTConfig:



Data Fields

- EXTChannelConfig channels [EXT_MAX_CHANNELS]

  Channel configurations.

8.22.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.22.2 Field Documentation

8.22.2.1 EXTChannelConfig EXTConfig::channels[EXT_MAX_CHANNELS]

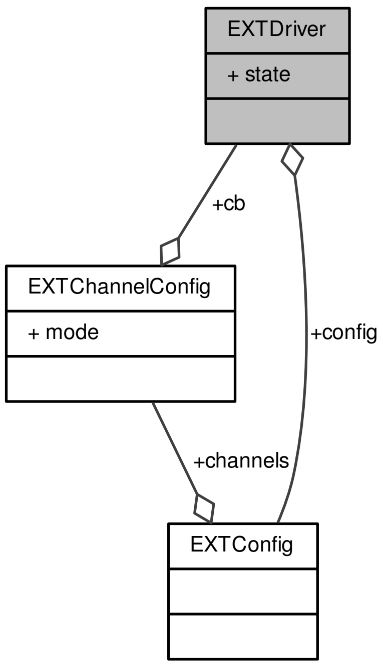
Channel configurations.

8.23 EXTDriver Struct Reference

Structure representing an EXT driver.

`#include <hal_ext_lld.h>`

Collaboration diagram for EXTDriver:



Data Fields

- extstate_t state

  Driver state.

- const EXTConfig ∗ config

  Current configuration data.

8.23.1 Detailed Description

Structure representing an EXT driver.

8.23.2 Field Documentation

8.23.2.1 extstate_t EXTDriver::state

Driver state.

8.23.2.2 const EXTConfig∗ EXTDriver::config

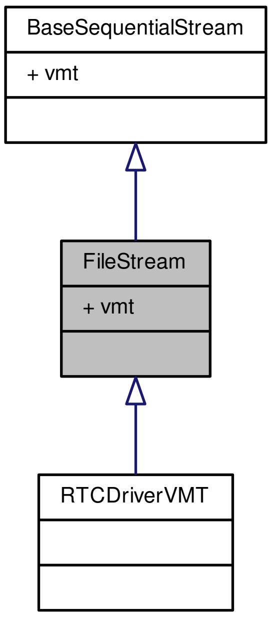
Current configuration data.

8.24 FileStream Struct Reference

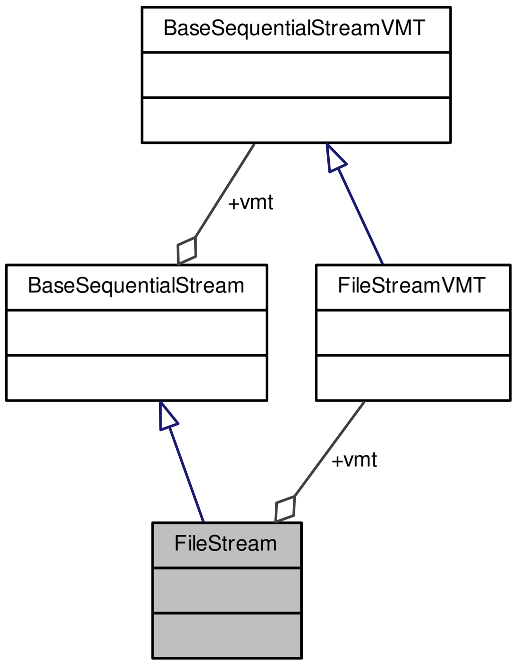
Base file stream class.

`#include <hal_files.h>`

Inheritance diagram for FileStream:



Collaboration diagram for FileStream:



Data Fields

- const struct FileStreamVMT ∗ vmt

  Virtual Methods Table.

8.24.1 Detailed Description

Base file stream class.

This class represents a generic file data stream.

8.24.2 Field Documentation

8.24.2.1 const struct FileStreamVMT∗ FileStream::vmt

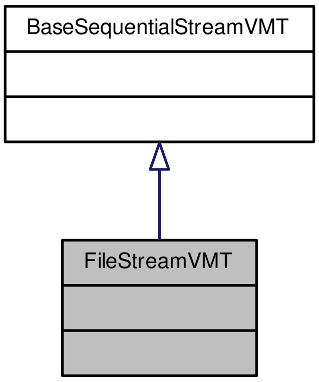
Virtual Methods Table.

8.25 FileStreamVMT Struct Reference

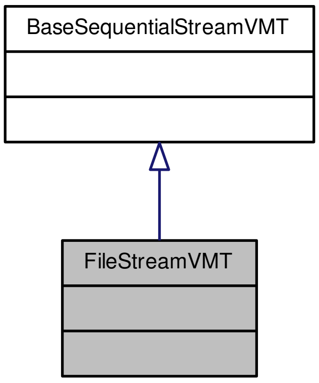
`FileStream` virtual methods table.

`#include <hal_files.h>`

Inheritance diagram for FileStreamVMT:



Collaboration diagram for FileStreamVMT:



8.25.1 Detailed Description

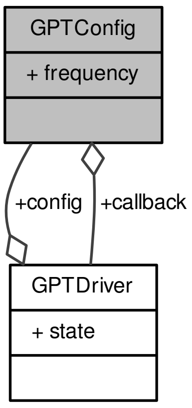
`FileStream` virtual methods table.

8.26 GPTConfig Struct Reference

Driver configuration structure.

```
#include <hal_gpt_lld.h>
```

Collaboration diagram for GPTConfig:



Data Fields

- gptfreq_t frequency

  Timer clock in Hz.
- gptcallback_t callback

  Timer callback pointer.

8.26.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.26.2 Field Documentation

8.26.2.1 gptfreq_t GPTConfig::frequency

Timer clock in Hz.

Note

The low level can use assertions in order to catch invalid frequency specifications.

8.26.2.2 gptcallback_t GPTConfig::callback

Timer callback pointer.

Note

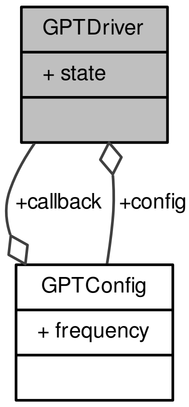
This callback is invoked on GPT counter events.

8.27 GPTDriver Struct Reference

Structure representing a GPT driver.

`#include <hal_gpt_lld.h>`

Collaboration diagram for GPTDriver:



Data Fields

- gptstate_t state

  Driver state.
- const GPTConfig ∗ config

  Current configuration data.

8.27.1 Detailed Description

Structure representing a GPT driver.

8.27.2 Field Documentation

8.27.2.1 gptstate_t GPTDriver::state

Driver state.

8.27.2.2 const GPTConfig∗ GPTDriver::config

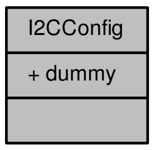
Current configuration data.

8.28 I2CConfig Struct Reference

Type of I2C driver configuration structure.

`#include <hal_i2c_lld.h>`

Collaboration diagram for I2CConfig:



8.28.1 Detailed Description

Type of I2C driver configuration structure.

Note

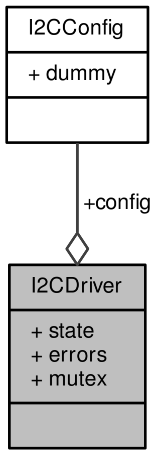
Implementations may extend this structure to contain more, architecture dependent, fields.

8.29 I2CDriver Struct Reference

Structure representing an I2C driver.

`#include <hal_i2c_lld.h>`

Collaboration diagram for I2CDriver:



Data Fields

- i2cstate_t state

  Driver state.

- const I2CConfig ∗ config

  Current configuration data.

- i2cflags_t errors

  Error flags.

8.29.1 Detailed Description

Structure representing an I2C driver.

8.29.2 Field Documentation

8.29.2.1 i2cstate_t I2CDriver::state

Driver state.

8.29.2.2 const I2CConfig∗ I2CDriver::config

Current configuration data.

8.29.2.3 i2cflags_t I2CDriver::errors

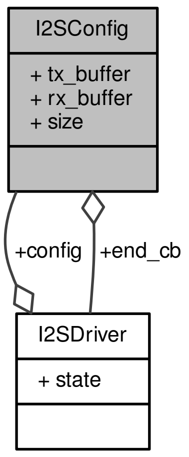
Error flags.

8.30 I2SConfig Struct Reference

Driver configuration structure.

```
#include <hal_i2s_lld.h>
```

Collaboration diagram for I2SConfig:



Data Fields

- const void ∗ tx_buffer

  Transmission buffer pointer.
- void ∗ rx_buffer

  Receive buffer pointer.
- size_t size

  TX and RX buffers size as number of samples.
- i2scallback_t end_cb

  Callback function called during streaming.

8.30.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.30.2 Field Documentation

8.30.2.1 const void∗ I2SConfig::tx_buffer

Transmission buffer pointer.

Note

Can be `NULL` if TX is not required.

8.30.2.2 void∗ I2SConfig::rx_buffer

Receive buffer pointer.

Note

Can be `NULL` if RX is not required.

8.30.2.3 size_t I2SConfig::size

TX and RX buffers size as number of samples.

8.30.2.4 i2scallback_t I2SConfig::end_cb

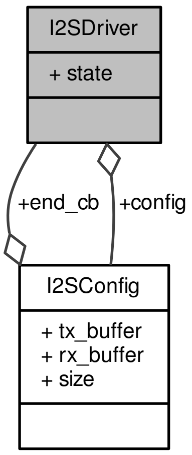
Callback function called during streaming.

8.31 I2SDriver Struct Reference

Structure representing an I2S driver.

`#include <hal_i2s_lld.h>`

Collaboration diagram for I2SDriver:



Data Fields

- i2sstate_t state

    Driver state.
- const I2SConfig ∗ config

    Current configuration data.

8.31.1 Detailed Description

Structure representing an I2S driver.

8.31.2 Field Documentation

8.31.2.1 i2sstate_t I2SDriver::state

Driver state.

8.31.2.2 const I2SConfig* I2SDriver::config

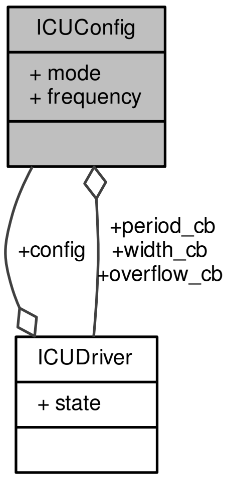
Current configuration data.

8.32 ICUConfig Struct Reference

Driver configuration structure.

```
#include <hal_icu_lld.h>
```

Collaboration diagram for ICUConfig:



Data Fields

- icumode_t mode

    Driver mode.
- icufreq_t frequency

    Timer clock in Hz.
- icucallback_t width_cb

    Callback for pulse width measurement.
- icucallback_t period_cb

Callback for cycle period measurement.

- icucallback_t overflow_cb

  Callback for timer overflow.

8.32.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.32.2 Field Documentation

8.32.2.1 icumode_t ICUConfig::mode

Driver mode.

8.32.2.2 icufreq_t ICUConfig::frequency

Timer clock in Hz.

Note

The low level can use assertions in order to catch invalid frequency specifications.

8.32.2.3 icucallback_t ICUConfig::width_cb

Callback for pulse width measurement.

8.32.2.4 icucallback_t ICUConfig::period_cb

Callback for cycle period measurement.

8.32.2.5 icucallback_t ICUConfig::overflow_cb

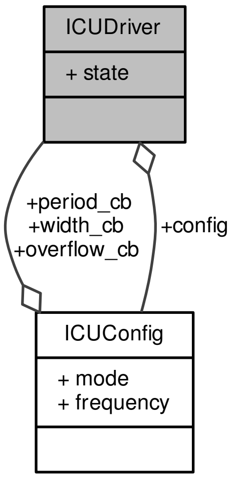
Callback for timer overflow.

8.33 ICUDriver Struct Reference

Structure representing an ICU driver.

```
#include <hal_icu_lld.h>
```

Collaboration diagram for ICUDriver:



Data Fields

- icustate_t state

    Driver state.

- const ICUConfig ∗ config

    Current configuration data.

8.33.1 Detailed Description

Structure representing an ICU driver.

8.33.2 Field Documentation

8.33.2.1 icustate_t ICUDriver::state

Driver state.

8.33.2.2 const ICUConfig∗ ICUDriver::config

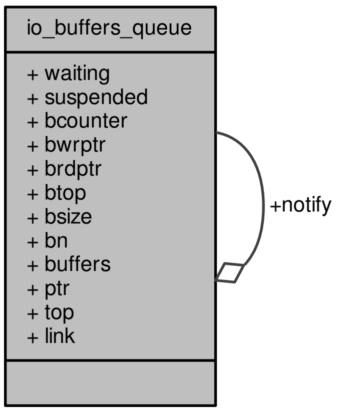
Current configuration data.

8.34 io_buffers_queue Struct Reference

Structure of a generic buffers queue.

```
#include <hal_buffers.h>
```

Collaboration diagram for io_buffers_queue:



Data Fields

- threads_queue_t waiting

  Queue of waiting threads.
- bool suspended

  Queue suspended state flag.
- volatile size_t bcounter

  Active buffers counter.
- uint8_t ∗ bwrptr

  Buffer write pointer.
- uint8_t ∗ brdptr

  Buffer read pointer.
- uint8_t ∗ btop

  Pointer to the buffers boundary.
- size_t bsize

  Size of buffers.
- size_t bn

  Number of buffers.
- uint8_t ∗ buffers

  Queue of buffer objects.
- uint8_t ∗ ptr

  Pointer for R/W sequential access.
- uint8_t ∗ top

  Boundary for R/W sequential access.
- bqnotify_t notify

  Data notification callback.
- void ∗ link

  Application defined field.

8.34.1 Detailed Description

Structure of a generic buffers queue.

8.34.2 Field Documentation

8.34.2.1 threads_queue_t io_buffers_queue::waiting

Queue of waiting threads.

8.34.2.2 bool io_buffers_queue::suspended

Queue suspended state flag.

8.34.2.3 volatile size_t io_buffers_queue::bcounter

Active buffers counter.

8.34.2.4 uint8_t* io_buffers_queue::bwrptr

Buffer write pointer.

8.34.2.5 uint8_t* io_buffers_queue::brdptr

Buffer read pointer.

8.34.2.6 uint8_t* io_buffers_queue::btop

Pointer to the buffers boundary.

8.34.2.7 size_t io_buffers_queue::bsize

Size of buffers.

Note

The buffer size must be not lower than `sizeof(size_t) + 2` because the first bytes are used to store the used size of the buffer.

8.34.2.8 size_t io_buffers_queue::bn

Number of buffers.

8.34.2.9 uint8_t* io_buffers_queue::buffers

Queue of buffer objects.

8.34.2.10 uint8_t∗ io_buffers_queue::ptr

Pointer for R/W sequential access.

Note

It is `NULL` if a new buffer must be fetched from the queue.

8.34.2.11 uint8_t∗ io_buffers_queue::top

Boundary for R/W sequential access.

8.34.2.12 bqnotify_t io_buffers_queue::notify

Data notification callback.

8.34.2.13 void∗ io_buffers_queue::link

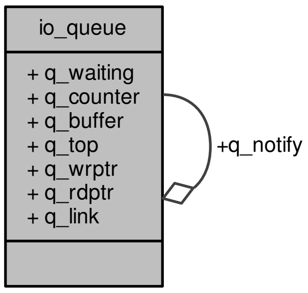
Application defined field.

8.35 io_queue Struct Reference

Generic I/O queue structure.

`#include <hal_queues.h>`

Collaboration diagram for io_queue:



Data Fields

- threads_queue_t q_waiting

  Queue of waiting threads.
- volatile size_t q_counter

  Resources counter.
- uint8_t ∗ q_buffer

  Pointer to the queue buffer.

- uint8_t ∗ q_top

    Pointer to the first location after the buffer.

- uint8_t ∗ q_wrptr

    Write pointer.

- uint8_t ∗ q_rdptr

    Read pointer.

- qnotify_t q_notify

    Data notification callback.

- void ∗ q_link

    Application defined field.

8.35.1 Detailed Description

Generic I/O queue structure.

This structure represents a generic Input or Output asymmetrical queue. The queue is asymmetrical because one end is meant to be accessed from a thread context, and thus can be blocking, the other end is accessible from interrupt handlers or from within a kernel lock zone and is non-blocking.

8.35.2 Field Documentation

8.35.2.1 threads_queue_t io_queue::q_waiting

Queue of waiting threads.

8.35.2.2 volatile size_t io_queue::q_counter

Resources counter.

8.35.2.3 uint8_t∗ io_queue::q_buffer

Pointer to the queue buffer.

8.35.2.4 uint8_t∗ io_queue::q_top

Pointer to the first location after the buffer.

8.35.2.5 uint8_t∗ io_queue::q_wrptr

Write pointer.

8.35.2.6 uint8_t∗ io_queue::q_rdptr

Read pointer.

8.35.2.7 qnotify_t io_queue::q_notify

Data notification callback.

8.35.2.8 void∗ io_queue::q_link

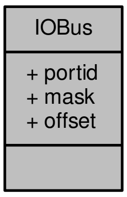
Application defined field.

8.36 IOBus Struct Reference

I/O bus descriptor.

`#include <hal_pal.h>`

Collaboration diagram for IOBus:



Data Fields

- ioportid_t portid
  Port identifier.
- ioportmask_t mask
  Bus mask aligned to port bit 0.
- uint_fast8_t offset
  Offset, within the port, of the least significant bit of the bus.

8.36.1 Detailed Description

I/O bus descriptor.

This structure describes a group of contiguous digital I/O lines that have to be handled as bus.

Note

I/O operations on a bus do not affect I/O lines on the same port but not belonging to the bus.

8.36.2 Field Documentation

8.36.2.1 ioportid_t IOBus::portid

Port identifier.

8.36.2.2 ioportmask_t IOBus::mask

Bus mask aligned to port bit 0.

Note

The bus mask implicitly define the bus width. A logic AND is performed on the bus data.

8.36.2.3 uint_fast8_t IOBus::offset

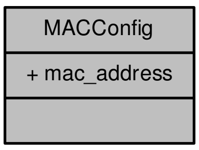
Offset, within the port, of the least significant bit of the bus.

8.37 MACConfig Struct Reference

Driver configuration structure.

```
#include <hal_mac_lld.h>
```

Collaboration diagram for MACConfig:



Data Fields

- uint8_t ∗ mac_address

  MAC address.

8.37.1 Detailed Description

Driver configuration structure.

8.37.2 Field Documentation

8.37.2.1 uint8_t∗ MACConfig::mac_address

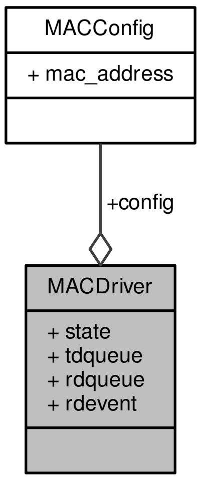
MAC address.

8.38 MACDriver Struct Reference

Structure representing a MAC driver.

```
#include <hal_mac_lld.h>
```

Collaboration diagram for MACDriver:



Data Fields

- macstate_t state

  Driver state.
- const MACConfig ∗ config

  Current configuration data.
- threads_queue_t tdqueue

  Transmit semaphore.
- threads_queue_t rdqueue

  Receive semaphore.
- event_source_t rdevent

  Receive event.

8.38.1 Detailed Description

Structure representing a MAC driver.

8.38.2 Field Documentation

8.38.2.1 macstate_t MACDriver::state

Driver state.

8.38.2.2 const MACConfig∗ MACDriver::config

Current configuration data.

8.38.2.3 threads_queue_t MACDriver::tdqueue

Transmit semaphore.

8.38.2.4 threads_queue_t MACDriver::rdqueue

Receive semaphore.

8.38.2.5 event_source_t MACDriver::rdevent

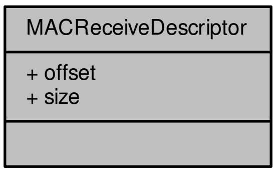
Receive event.

8.39 MACReceiveDescriptor Struct Reference

Structure representing a receive descriptor.

`#include <hal_mac_lld.h>`

Collaboration diagram for MACReceiveDescriptor:



Data Fields

- size_t offset

  Current read offset.
- size_t size

  Available data size.

8.39.1 Detailed Description

Structure representing a receive descriptor.

8.39.2 Field Documentation

8.39.2.1 size_t MACReceiveDescriptor::offset

Current read offset.

8.39.2.2 size_t MACReceiveDescriptor::size

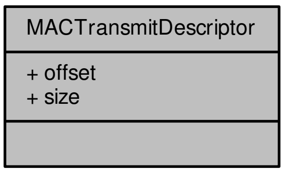
Available data size.

8.40 MACTransmitDescriptor Struct Reference

Structure representing a transmit descriptor.

`#include <hal_mac_lld.h>`

Collaboration diagram for MACTransmitDescriptor:



Data Fields

- size_t offset

  Current write offset.
- size_t size

  Available space size.

8.40.1 Detailed Description

Structure representing a transmit descriptor.

8.40.2 Field Documentation

8.40.2.1 size_t MACTransmitDescriptor::offset

Current write offset.

8.40.2.2 size_t MACTransmitDescriptor::size

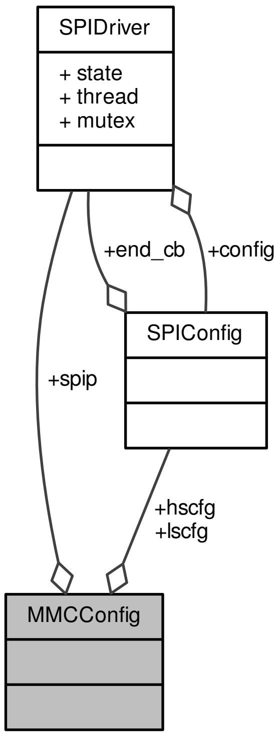
Available space size.

8.41 MMCConfig Struct Reference

MMC/SD over SPI driver configuration structure.

`#include <hal_mmc_spi.h>`

Collaboration diagram for MMCConfig:



Data Fields

- SPIDriver ∗ spip

  SPI driver associated to this MMC driver.
- const SPIConfig ∗ lscfg

  SPI low speed configuration used during initialization.
- const SPIConfig ∗ hscfg

  SPI high speed configuration used during transfers.

8.41.1 Detailed Description

MMC/SD over SPI driver configuration structure.

8.41.2 Field Documentation

8.41.2.1 SPIDriver∗ MMCConfig::spip

SPI driver associated to this MMC driver.

8.41.2.2 const SPIConfig∗ MMCConfig::lscfg

SPI low speed configuration used during initialization.

8.41.2.3 const SPIConfig∗ MMCConfig::hscfg

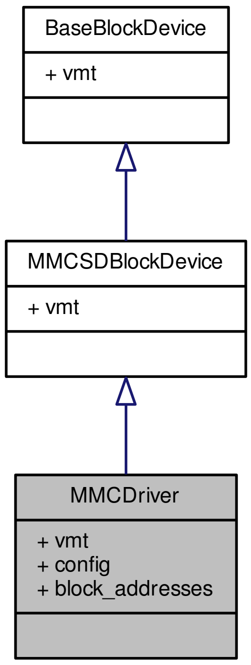
SPI high speed configuration used during transfers.

8.42 MMCDriver Struct Reference

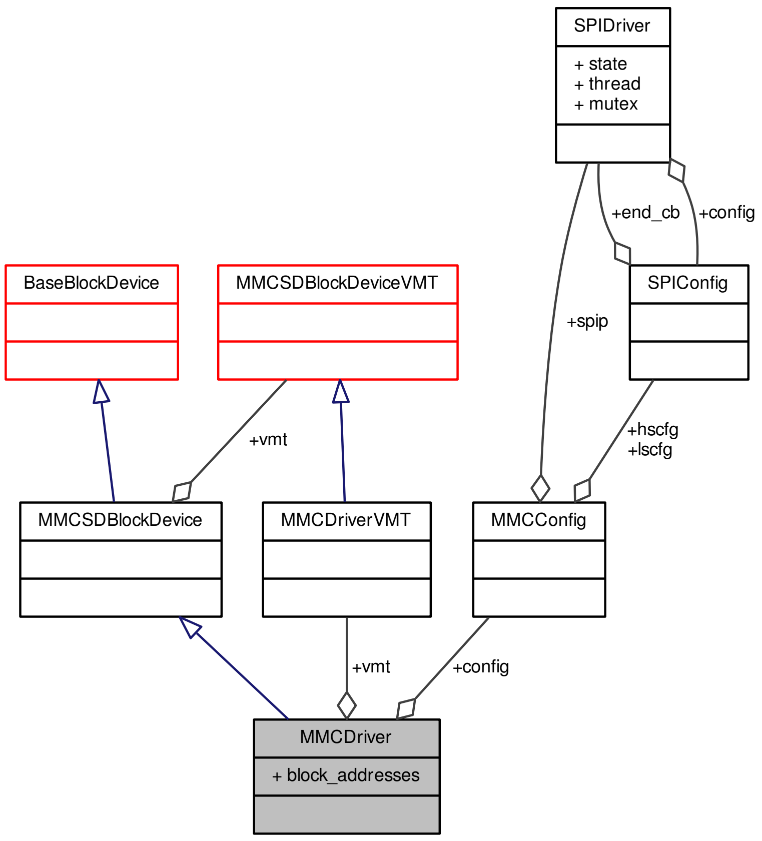
Structure representing a MMC/SD over SPI driver.

`#include <hal_mmc_spi.h>`

Inheritance diagram for MMCDriver:



Collaboration diagram for MMCDriver:



Data Fields

- const struct MMCDriverVMT ∗ vmt

  Virtual Methods Table.
- _mmcsd_block_device_data const MMCConfig ∗ config

  Current configuration data.

8.42.1 Detailed Description

Structure representing a MMC/SD over SPI driver.

8.42.2 Field Documentation

8.42.2.1 const struct MMCDriverVMT∗ MMCDriver::vmt

Virtual Methods Table.

8.42.2.2 _mmcsd_block_device_data const MMCConfig∗ MMCDriver::config

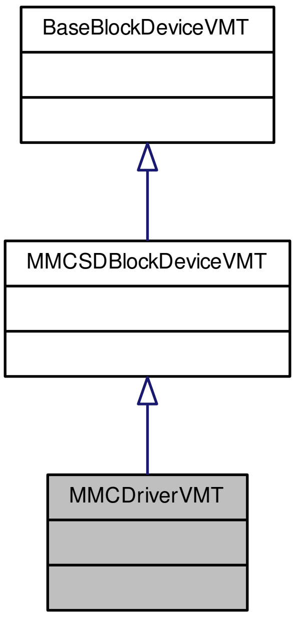
Current configuration data.

8.43 MMCDriverVMT Struct Reference

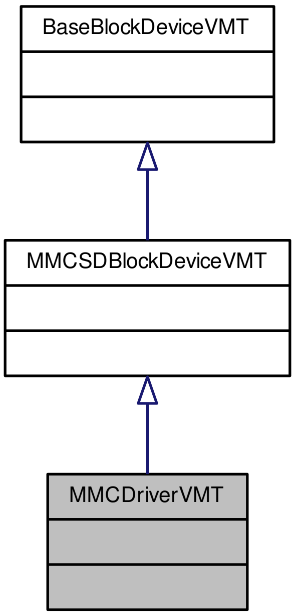
`MMCDriver` virtual methods table.

`#include <hal_mmc_spi.h>`

Inheritance diagram for MMCDriverVMT:



Collaboration diagram for MMCDriverVMT:



8.43.1 Detailed Description

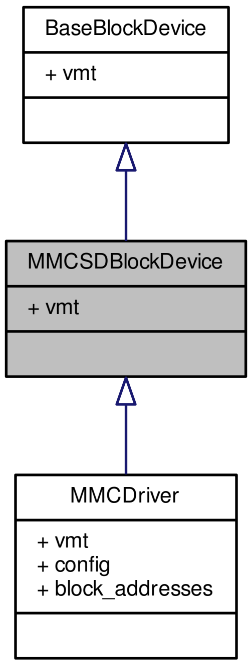
MMCDriver virtual methods table.

8.44 MMCSDBlockDevice Struct Reference

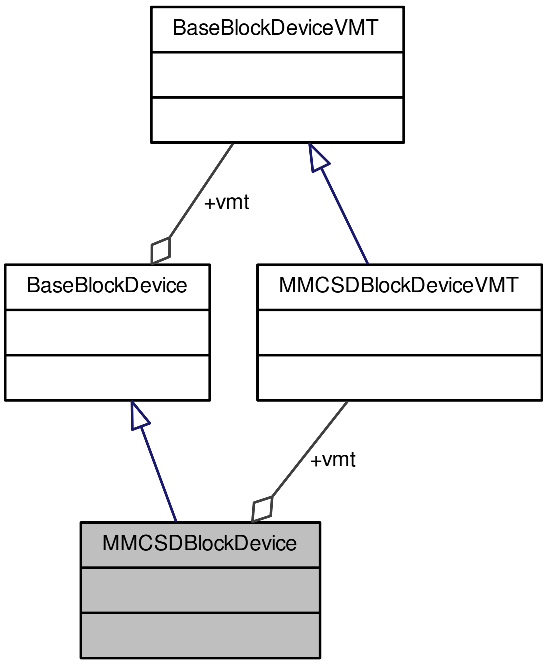
MCC/SD block device class.

`#include <hal_mmcsd.h>`

Inheritance diagram for MMCSDBlockDevice:



Collaboration diagram for MMCSDBlockDevice:



Data Fields

- const struct MMCSDBlockDeviceVMT ∗ vmt

  Virtual Methods Table.

8.44.1 Detailed Description

MCC/SD block device class.

This class represents a, block-accessible, MMC/SD device.

8.44.2 Field Documentation

8.44.2.1 const struct MMCSDBlockDeviceVMT∗ MMCSDBlockDevice::vmt

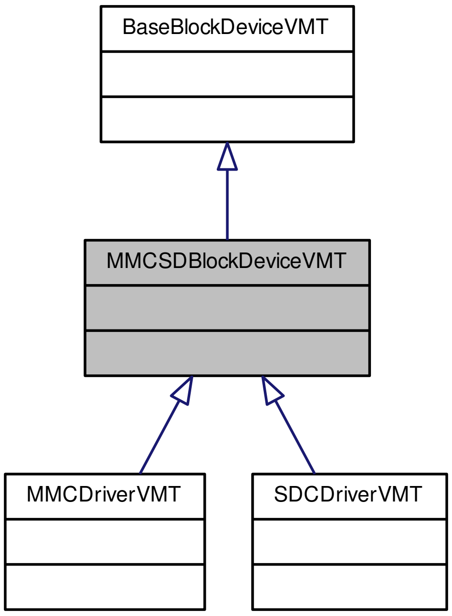
Virtual Methods Table.

8.45 MMCSDBlockDeviceVMT Struct Reference

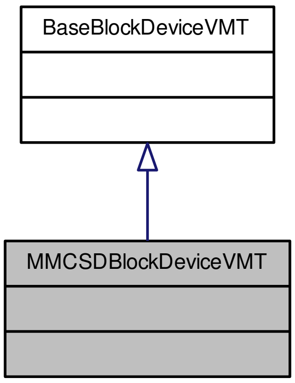
`MMCSDBlockDevice` virtual methods table.

`#include <hal_mmcsd.h>`

Inheritance diagram for MMCSDBlockDeviceVMT:



Collaboration diagram for MMCSDBlockDeviceVMT:



8.45.1 Detailed Description

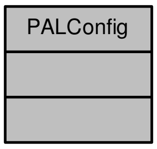
`MMCSDBlockDevice` virtual methods table.

8.46 PALConfig Struct Reference

Generic I/O ports static initializer.

`#include <hal_pal_lld.h>`

Collaboration diagram for PALConfig:



8.46.1 Detailed Description

Generic I/O ports static initializer.

An instance of this structure must be passed to `palInit()` at system startup time in order to initialized the digital I/O subsystem. This represents only the initial setup, specific pads or whole ports can be reprogrammed at later time.

Note

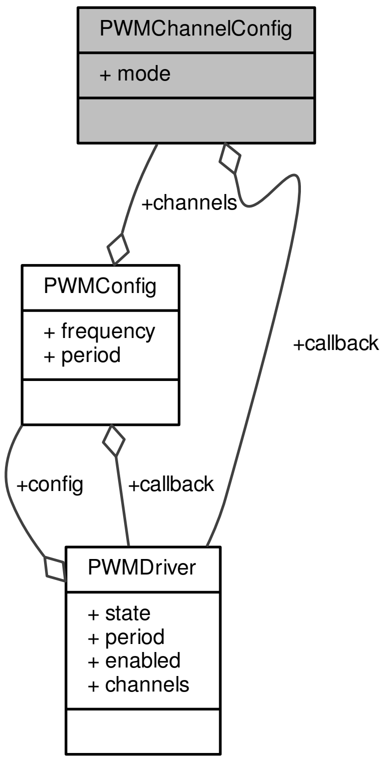
Implementations may extend this structure to contain more, architecture dependent, fields.

8.47 PWMChannelConfig Struct Reference

Type of a PWM driver channel configuration structure.

`#include <hal_pwm_lld.h>`

Collaboration diagram for PWMChannelConfig:



Data Fields

- pwmmode_t mode

  Channel active logic level.
- pwmcallback_t callback

  Channel callback pointer.

8.47.1 Detailed Description

Type of a PWM driver channel configuration structure.

8.47.2 Field Documentation

8.47.2.1 pwmmode_t PWMChannelConfig::mode

Channel active logic level.

8.47.2.2 pwmcallback_t PWMChannelConfig::callback

Channel callback pointer.

Note

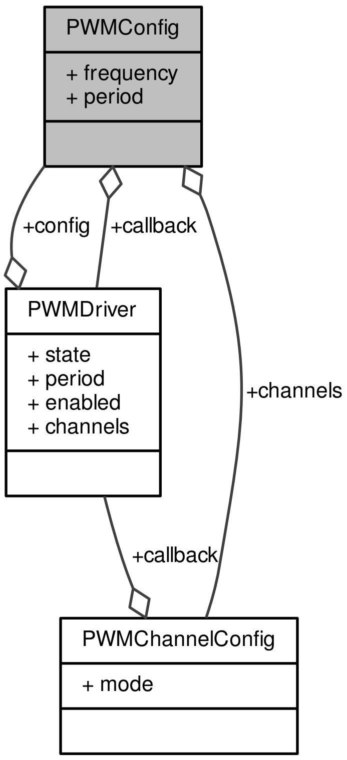
This callback is invoked on the channel compare event. If set to `NULL` then the callback is disabled.

8.48 PWMConfig Struct Reference

Type of a PWM driver configuration structure.

`#include <hal_pwm_lld.h>`

Collaboration diagram for PWMConfig:



Data Fields

- uint32_t frequency

  Timer clock in Hz.
- pwmcnt_t period

  PWM period in ticks.
- pwmcallback_t callback

  Periodic callback pointer.
- PWMChannelConfig channels [PWM_CHANNELS]

  Channels configurations.

8.48.1 Detailed Description

Type of a PWM driver configuration structure.

8.48.2 Field Documentation

8.48.2.1 uint32_t PWMConfig::frequency

Timer clock in Hz.

Note

The low level can use assertions in order to catch invalid frequency specifications.

8.48.2.2 pwmcnt_t PWMConfig::period

PWM period in ticks.

Note

The low level can use assertions in order to catch invalid period specifications.

8.48.2.3 pwmcallback_t PWMConfig::callback

Periodic callback pointer.

Note

This callback is invoked on PWM counter reset. If set to `NULL` then the callback is disabled.

8.48.2.4 PWMChannelConfig PWMConfig::channels[PWM_CHANNELS]

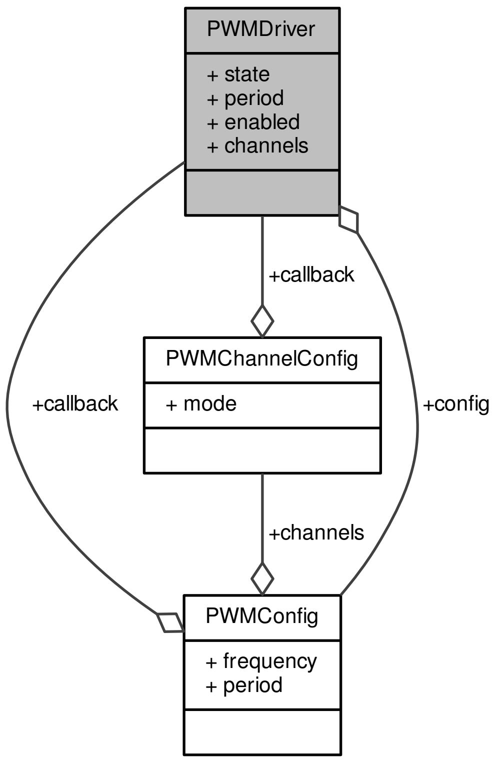
Channels configurations.

8.49 PWMDriver Struct Reference

Structure representing a PWM driver.

`#include <hal_pwm_lld.h>`

Collaboration diagram for PWMDriver:



Data Fields

- pwmstate_t state

  Driver state.
- const PWMConfig ∗ config

  Current driver configuration data.
- pwmcnt_t period

  Current PWM period in ticks.
- pwmchnmsk_t enabled

  Mask of the enabled channels.
- pwmchannel_t channels

  Number of channels in this instance.

8.49.1 Detailed Description

Structure representing a PWM driver.

8.49.2 Field Documentation

8.49.2.1 pwmstate_t PWMDriver::state

Driver state.

8.49.2.2 const PWMConfig∗ PWMDriver::config

Current driver configuration data.

8.49.2.3 pwmcnt_t PWMDriver::period

Current PWM period in ticks.

8.49.2.4 pwmchnmsk_t PWMDriver::enabled

Mask of the enabled channels.

8.49.2.5 pwmchannel_t PWMDriver::channels

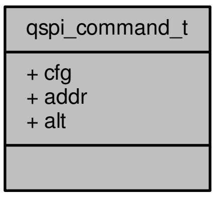
Number of channels in this instance.

8.50 qspi_command_t Struct Reference

Type of a QSPI command descriptor.

`#include <hal_qspi.h>`

Collaboration diagram for qspi_command_t:



8.50.1 Detailed Description

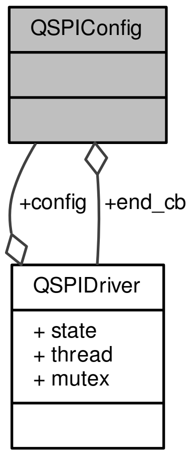
Type of a QSPI command descriptor.

8.51 QSPIConfig Struct Reference

Driver configuration structure.

`#include <hal_qspi_lld.h>`

Collaboration diagram for QSPIConfig:



Data Fields

- qspicallback_t end_cb

  Operation complete callback or `NULL`.

8.51.1 Detailed Description

Driver configuration structure.

8.51.2 Field Documentation

8.51.2.1 qspicallback_t QSPIConfig::end_cb

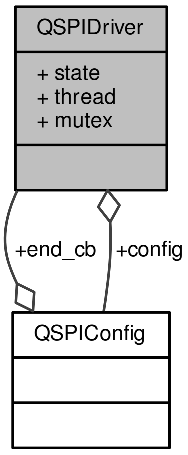
Operation complete callback or `NULL`.

8.52 QSPIDriver Struct Reference

Structure representing an QSPI driver.

```
#include <hal_qspi_lld.h>
```

Collaboration diagram for QSPIDriver:



Data Fields

- qspistate_t state

  Driver state.
- const QSPIConfig ∗ config

  Current configuration data.
- thread_reference_t thread

  Waiting thread.
- mutex_t mutex

  Mutex protecting the peripheral.

8.52.1 Detailed Description

Structure representing an QSPI driver.

8.52.2 Field Documentation

8.52.2.1 qspistate_t QSPIDriver::state

Driver state.

8.52.2.2 const QSPIConfig∗ QSPIDriver::config

Current configuration data.

8.52.2.3 thread_reference_t QSPIDriver::thread

Waiting thread.

8.52.2.4 mutex_t QSPIDriver::mutex

Mutex protecting the peripheral.

8.53 RTCAlarm Struct Reference

Type of a structure representing an RTC alarm time stamp.

`#include <hal_rtc_lld.h>`

Collaboration diagram for RTCAlarm:

RTCAlarm
+ dummy

8.53.1 Detailed Description

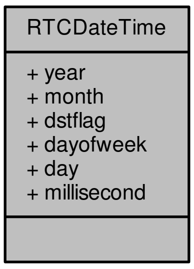
Type of a structure representing an RTC alarm time stamp.

8.54 RTCDateTime Struct Reference

Type of a structure representing an RTC date/time stamp.

`#include <hal_rtc.h>`

Collaboration diagram for RTCDateTime:



Data Fields

- uint32_t year: 8

  Years since 1980.
- uint32_t month: 4

  Months 1..12.
- uint32_t dstflag: 1

  DST correction flag.
- uint32_t dayofweek: 3

  Day of week 1..7.
- uint32_t day: 5

  Day of the month 1..31.
- uint32_t millisecond: 27

  Milliseconds since midnight.

8.54.1 Detailed Description

Type of a structure representing an RTC date/time stamp.

8.54.2 Field Documentation

8.54.2.1 uint32_t RTCDateTime::year

Years since 1980.

8.54.2.2 uint32_t RTCDateTime::month

Months 1..12.

8.54.2.3 uint32_t RTCDateTime::dstflag

DST correction flag.

8.54.2.4 uint32_t RTCDateTime::dayofweek

Day of week 1..7.

8.54.2.5 uint32_t RTCDateTime::day

Day of the month 1..31.

8.54.2.6 uint32_t RTCDateTime::millisecond

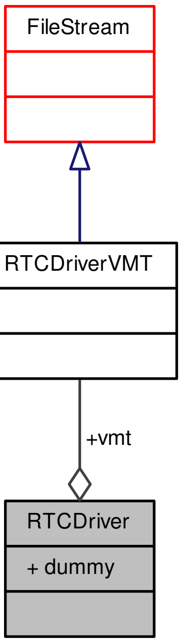
Milliseconds since midnight.

8.55 RTCDriver Struct Reference

Structure representing an RTC driver.

`#include <hal_rtc_lld.h>`

Collaboration diagram for RTCDriver:



Data Fields

- const struct RTCDriverVMT ∗ vmt

  Virtual Methods Table.

8.55.1 Detailed Description

Structure representing an RTC driver.

8.55.2 Field Documentation

8.55.2.1 const struct RTCDriverVMT∗ RTCDriver::vmt

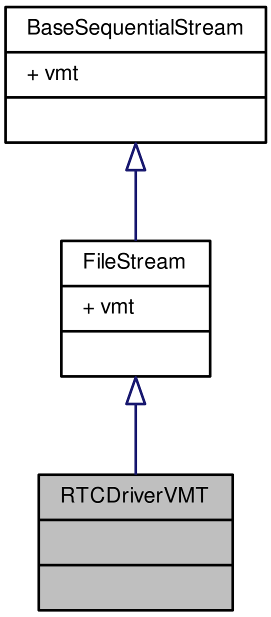
Virtual Methods Table.

8.56 RTCDriverVMT Struct Reference

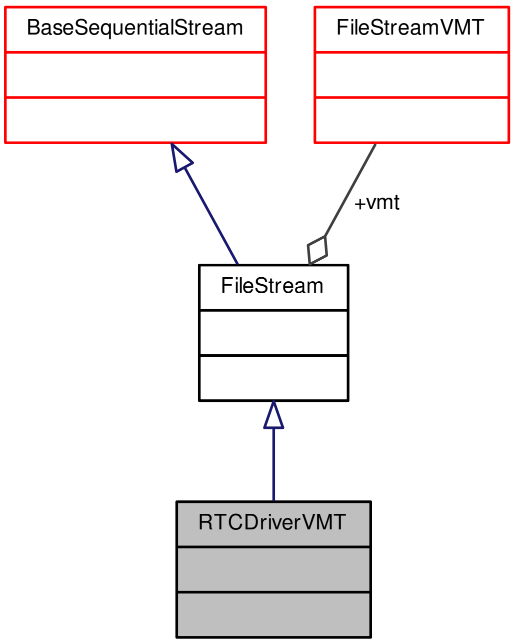
`RTCDriver` virtual methods table.

```
#include <hal_rtc_lld.h>
```

Inheritance diagram for RTCDriverVMT:



Collaboration diagram for RTCDriverVMT:



Additional Inherited Members

8.56.1 Detailed Description

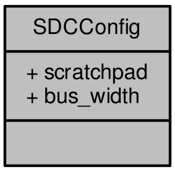
`RTCDriver` virtual methods table.

8.57 SDCConfig Struct Reference

Driver configuration structure.

`#include <hal_sdc_lld.h>`

Collaboration diagram for SDCConfig:



Data Fields

- uint8_t ∗ scratchpad

    Working area for memory consuming operations.
- sdcbusmode_t bus_width

    Bus width.

8.57.1 Detailed Description

Driver configuration structure.

Note

It could be empty on some architectures.

8.57.2 Field Documentation

8.57.2.1 uint8_t∗ SDCConfig::scratchpad

Working area for memory consuming operations.

Note

It is mandatory for detecting MMC cards bigger than 2GB else it can be `NULL`.
Memory pointed by this buffer is only used by `sdcConnect()`, afterward it can be reused for other purposes.

8.57.2.2 sdcbusmode_t SDCConfig::bus_width

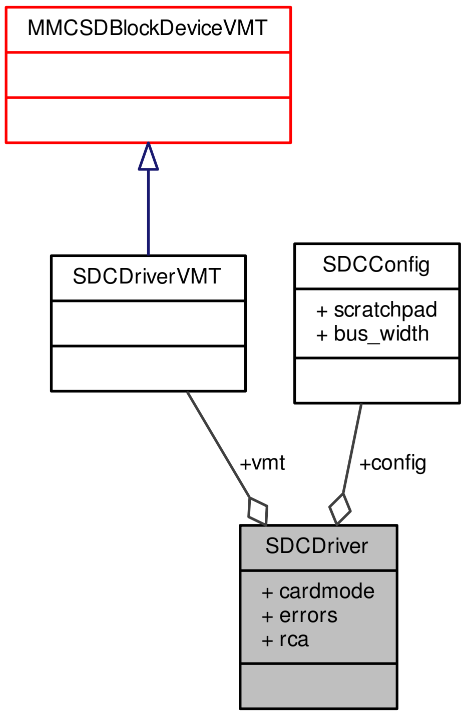
Bus width.

8.58 SDCDriver Struct Reference

Structure representing an SDC driver.

`#include <hal_sdc_lld.h>`

Collaboration diagram for SDCDriver:



Data Fields

- const struct SDCDriverVMT ∗ vmt

  Virtual Methods Table.
- _mmcsd_block_device_data const SDCConfig ∗ config

  Current configuration data.
- sdcmode_t cardmode

  Various flags regarding the mounted card.
- sdcflags_t errors

  Errors flags.
- uint32_t rca

  Card RCA.

8.58.1 Detailed Description

Structure representing an SDC driver.

8.58.2 Field Documentation

8.58.2.1 const struct SDCDriverVMT∗ SDCDriver::vmt

Virtual Methods Table.

8.58.2.2 _mmcsd_block_device_data const SDCConfig∗ SDCDriver::config

Current configuration data.

8.58.2.3 sdcmode_t SDCDriver::cardmode

Various flags regarding the mounted card.

8.58.2.4 sdcflags_t SDCDriver::errors

Errors flags.

8.58.2.5 uint32_t SDCDriver::rca

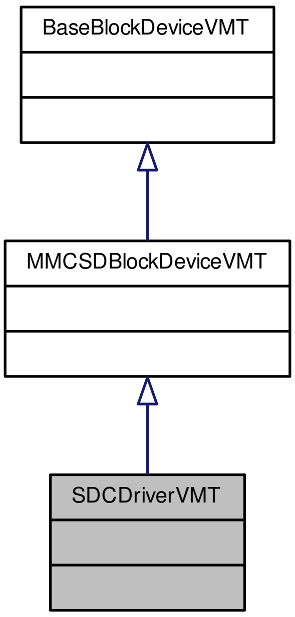
Card RCA.

8.59 SDCDriverVMT Struct Reference

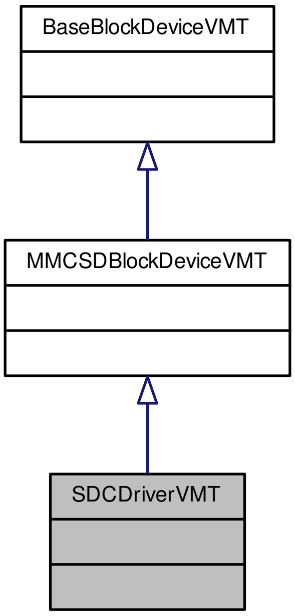
`SDCDriver` virtual methods table.

`#include <hal_sdc_lld.h>`

Inheritance diagram for SDCDriverVMT:



Collaboration diagram for SDCDriverVMT:



8.59.1 Detailed Description

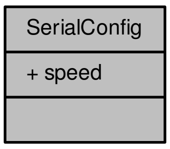
SDCDriver virtual methods table.

8.60 SerialConfig Struct Reference

PLATFORM Serial Driver configuration structure.

`#include <hal_serial_lld.h>`

Collaboration diagram for SerialConfig:



Data Fields

- uint32_t speed

  Bit rate.

8.60.1 Detailed Description

PLATFORM Serial Driver configuration structure.

An instance of this structure must be passed to `sdStart()` in order to configure and start a serial driver operations.

Note

This structure content is architecture dependent, each driver implementation defines its own version and the custom static initializers.

8.60.2 Field Documentation

8.60.2.1 uint32_t SerialConfig::speed

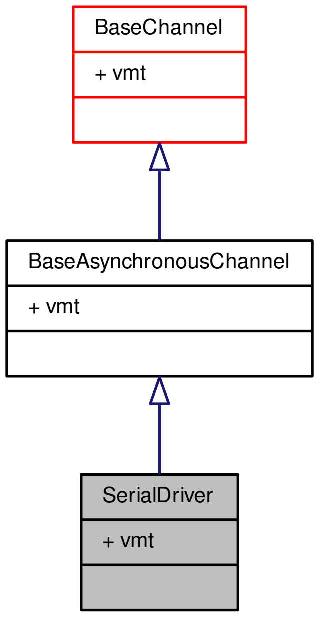
Bit rate.

8.61 SerialDriver Struct Reference

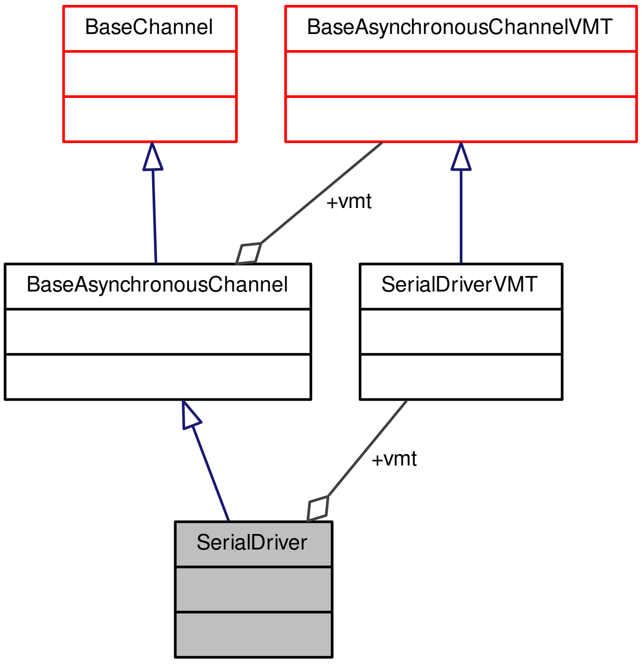
Full duplex serial driver class.

`#include <hal_serial.h>`

Inheritance diagram for SerialDriver:



Collaboration diagram for SerialDriver:



Data Fields

- const struct SerialDriverVMT ∗ vmt

  Virtual Methods Table.

8.61.1 Detailed Description

Full duplex serial driver class.

This class extends `BaseAsynchronousChannel` by adding physical I/O queues.

8.61.2 Field Documentation

8.61.2.1 const struct SerialDriverVMT∗ SerialDriver::vmt

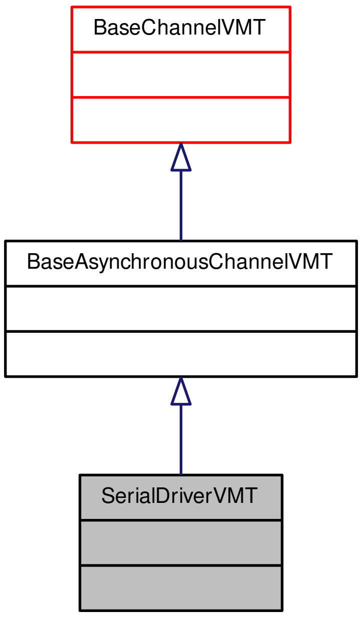
Virtual Methods Table.

8.62 SerialDriverVMT Struct Reference

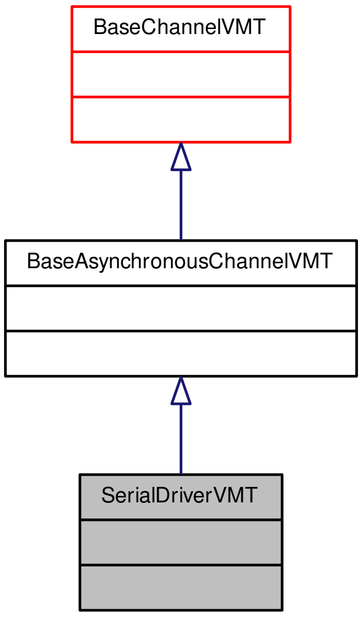
`SerialDriver` virtual methods table.

```
#include <hal_serial.h>
```

Inheritance diagram for SerialDriverVMT:



Collaboration diagram for SerialDriverVMT:



8.62.1 Detailed Description

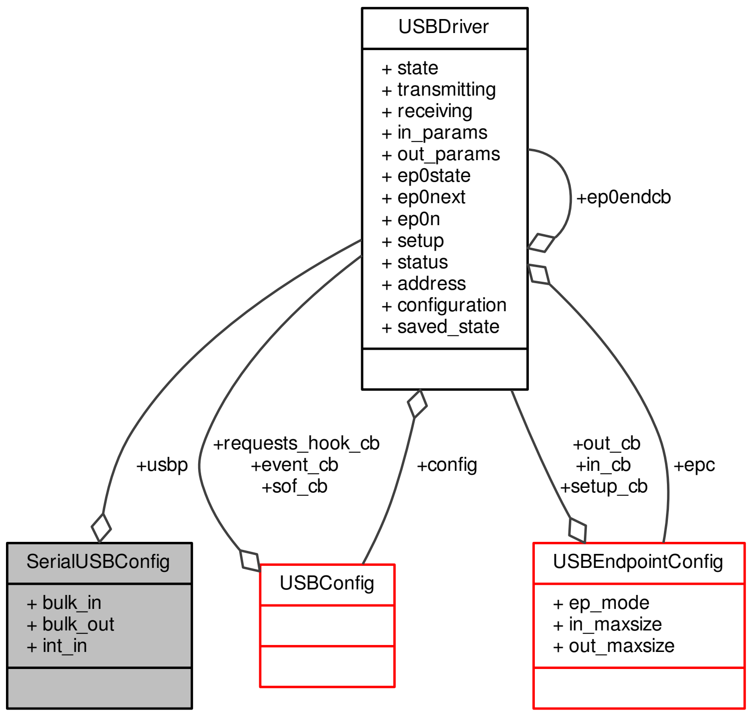
`SerialDriver` virtual methods table.

8.63 SerialUSBConfig Struct Reference

Serial over USB Driver configuration structure.

`#include <hal_serial_usb.h>`

Collaboration diagram for SerialUSBConfig:



Data Fields

- USBDriver ∗ usbp

    USB driver to use.

- usbep_t bulk_in

    Bulk IN endpoint used for outgoing data transfer.

- usbep_t bulk_out

    Bulk OUT endpoint used for incoming data transfer.

- usbep_t int_in

    Interrupt IN endpoint used for notifications.

8.63.1 Detailed Description

Serial over USB Driver configuration structure.

An instance of this structure must be passed to `sduStart()` in order to configure and start the driver operations.

8.63.2 Field Documentation

8.63.2.1 USBDriver∗ SerialUSBConfig::usbp

USB driver to use.

8.63.2.2 usbep_t SerialUSBConfig::bulk_in

Bulk IN endpoint used for outgoing data transfer.

8.63.2.3 usbep_t SerialUSBConfig::bulk_out

Bulk OUT endpoint used for incoming data transfer.

8.63.2.4 usbep_t SerialUSBConfig::int_in

Interrupt IN endpoint used for notifications.

Note

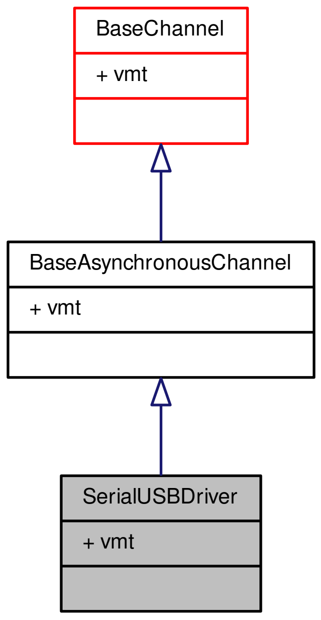
If set to zero then the INT endpoint is assumed to be not present, USB descriptors must be changed accordingly.

8.64 SerialUSBDriver Struct Reference

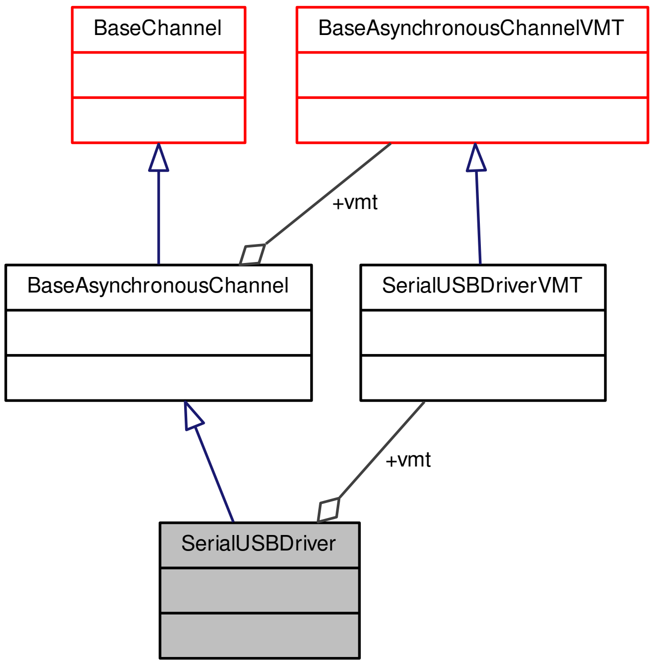
Full duplex serial driver class.

```
#include <hal_serial_usb.h>
```

Inheritance diagram for SerialUSBDriver:



Collaboration diagram for SerialUSBDriver:



Data Fields

- const struct SerialUSBDriverVMT ∗ vmt

  Virtual Methods Table.

8.64.1 Detailed Description

Full duplex serial driver class.

This class extends `BaseAsynchronousChannel` by adding physical I/O queues.

8.64.2 Field Documentation

8.64.2.1 const struct SerialUSBDriverVMT∗ SerialUSBDriver::vmt

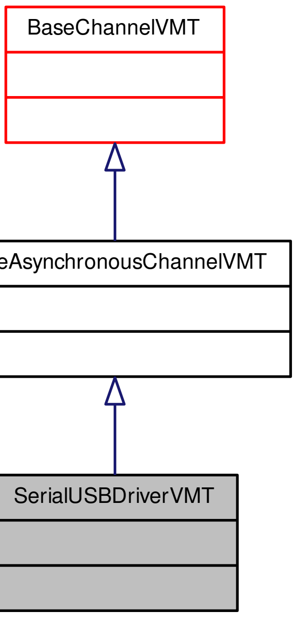
Virtual Methods Table.

8.65 SerialUSBDriverVMT Struct Reference

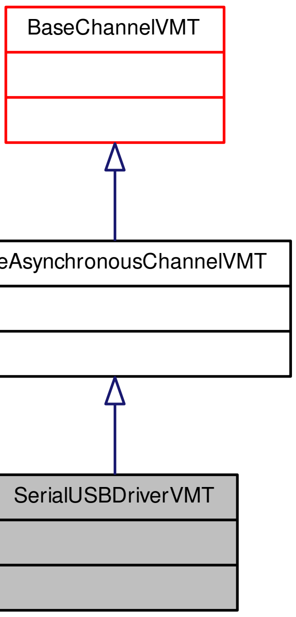
`SerialDriver` virtual methods table.

`#include <hal_serial_usb.h>`

Inheritance diagram for SerialUSBDriverVMT:



Collaboration diagram for SerialUSBDriverVMT:



8.65.1 Detailed Description

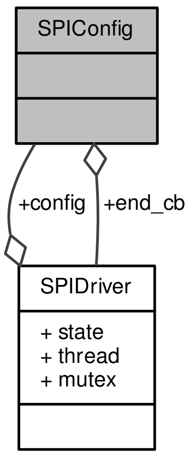
`SerialDriver` virtual methods table.

8.66 SPIConfig Struct Reference

Driver configuration structure.

`#include <hal_spi_lld.h>`

Collaboration diagram for SPIConfig:



Data Fields

- spicallback_t end_cb

  Operation complete callback or `NULL`*.*

8.66.1 Detailed Description

Driver configuration structure.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

8.66.2 Field Documentation

8.66.2.1 spicallback_t SPIConfig::end_cb

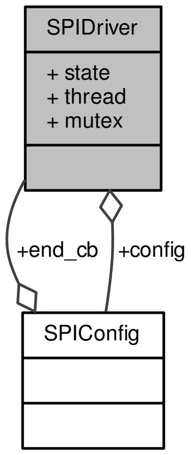
Operation complete callback or `NULL`.

8.67 SPIDriver Struct Reference

Structure representing an SPI driver.

`#include <hal_spi_lld.h>`

Collaboration diagram for SPIDriver:



Data Fields

- spistate_t state

    Driver state.
- const SPIConfig ∗ config

    Current configuration data.
- thread_reference_t thread

    Waiting thread.
- mutex_t mutex

    Mutex protecting the peripheral.

8.67.1 Detailed Description

Structure representing an SPI driver.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

8.67.2 Field Documentation

8.67.2.1 spistate_t SPIDriver::state

Driver state.

8.67.2.2 const SPIConfig∗ SPIDriver::config

Current configuration data.

8.67.2.3 thread_reference_t SPIDriver::thread

Waiting thread.

8.67.2.4 mutex_t SPIDriver::mutex

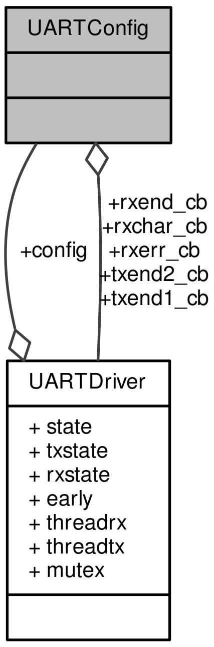
Mutex protecting the peripheral.

8.68 UARTConfig Struct Reference

Driver configuration structure.

`#include <hal_uart_lld.h>`

Collaboration diagram for UARTConfig:



Data Fields

- uartcb_t txend1_cb

    End of transmission buffer callback.
- uartcb_t txend2_cb

    Physical end of transmission callback.

- uartcb_t rxend_cb

  Receive buffer filled callback.
- uartccb_t rxchar_cb

  Character received while out if the `UART_RECEIVE` state.
- uartecb_t rxerr_cb

  Receive error callback.

8.68.1 Detailed Description

Driver configuration structure.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

8.68.2 Field Documentation

8.68.2.1 uartcb_t UARTConfig::txend1_cb

End of transmission buffer callback.

8.68.2.2 uartcb_t UARTConfig::txend2_cb

Physical end of transmission callback.

8.68.2.3 uartcb_t UARTConfig::rxend_cb

Receive buffer filled callback.

8.68.2.4 uartccb_t UARTConfig::rxchar_cb

Character received while out if the `UART_RECEIVE` state.

8.68.2.5 uartecb_t UARTConfig::rxerr_cb

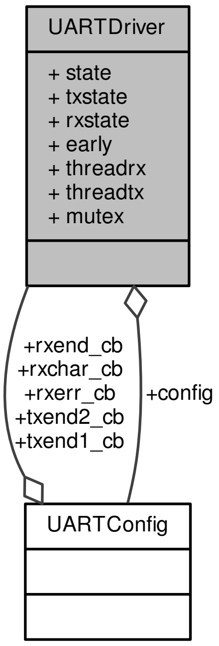
Receive error callback.

8.69 UARTDriver Struct Reference

Structure representing an UART driver.

`#include <hal_uart_lld.h>`

Collaboration diagram for UARTDriver:



Data Fields

- uartstate_t state

    Driver state.
- uarttxstate_t txstate

    Transmitter state.
- uartrxstate_t rxstate

    Receiver state.
- const UARTConfig ∗ config

    Current configuration data.
- bool early

    Synchronization flag for transmit operations.
- thread_reference_t threadrx

    Waiting thread on RX.
- thread_reference_t threadtx

    Waiting thread on TX.
- mutex_t mutex

    Mutex protecting the peripheral.

8.69.1 Detailed Description

Structure representing an UART driver.

Note

Implementations may extend this structure to contain more, architecture dependent, fields.

8.69.2 Field Documentation

8.69.2.1 uartstate_t UARTDriver::state

Driver state.

8.69.2.2 uarttxstate_t UARTDriver::txstate

Transmitter state.

8.69.2.3 uartrxstate_t UARTDriver::rxstate

Receiver state.

8.69.2.4 const UARTConfig* UARTDriver::config

Current configuration data.

8.69.2.5 bool UARTDriver::early

Synchronization flag for transmit operations.

8.69.2.6 thread_reference_t UARTDriver::threadrx

Waiting thread on RX.

8.69.2.7 thread_reference_t UARTDriver::threadtx

Waiting thread on TX.

8.69.2.8 mutex_t UARTDriver::mutex

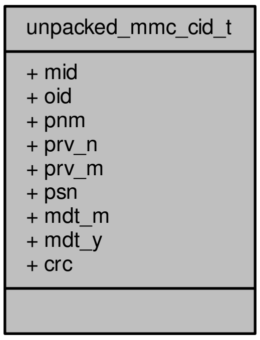
Mutex protecting the peripheral.

8.70 unpacked_mmc_cid_t Struct Reference

Unpacked CID register from MMC.

```
#include <hal_mmcsd.h>
```

Collaboration diagram for unpacked_mmc_cid_t:



8.70.1 Detailed Description

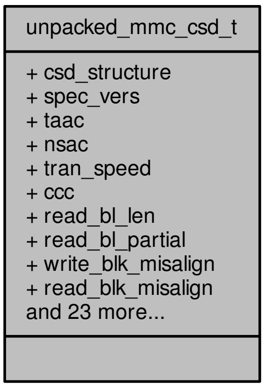
Unpacked CID register from MMC.

8.71 unpacked_mmc_csd_t Struct Reference

Unpacked CSD register from MMC.

`#include <hal_mmcsd.h>`

Collaboration diagram for unpacked_mmc_csd_t:



8.71.1 Detailed Description

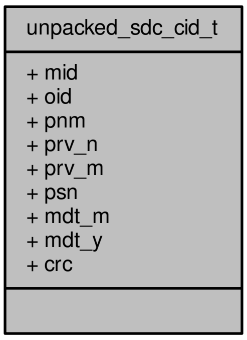
Unpacked CSD register from MMC.

8.72 unpacked_sdc_cid_t Struct Reference

Unpacked CID register from SDC.

`#include <hal_mmcsd.h>`

Collaboration diagram for unpacked_sdc_cid_t:



8.72.1 Detailed Description

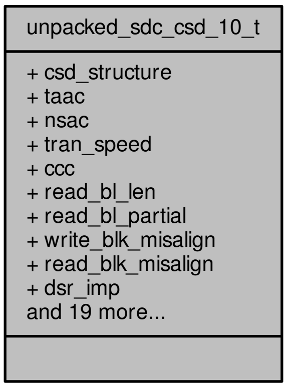
Unpacked CID register from SDC.

8.73 unpacked_sdc_csd_10_t Struct Reference

Unpacked CSD v1.0 register from SDC.

`#include <hal_mmcsd.h>`

Collaboration diagram for unpacked_sdc_csd_10_t:



8.73.1 Detailed Description

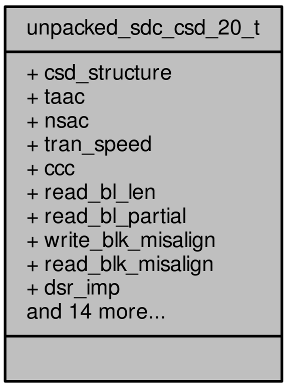
Unpacked CSD v1.0 register from SDC.

8.74 unpacked_sdc_csd_20_t Struct Reference

Unpacked CSD v2.0 register from SDC.

`#include <hal_mmcsd.h>`

Collaboration diagram for unpacked_sdc_csd_20_t:



8.74.1 Detailed Description

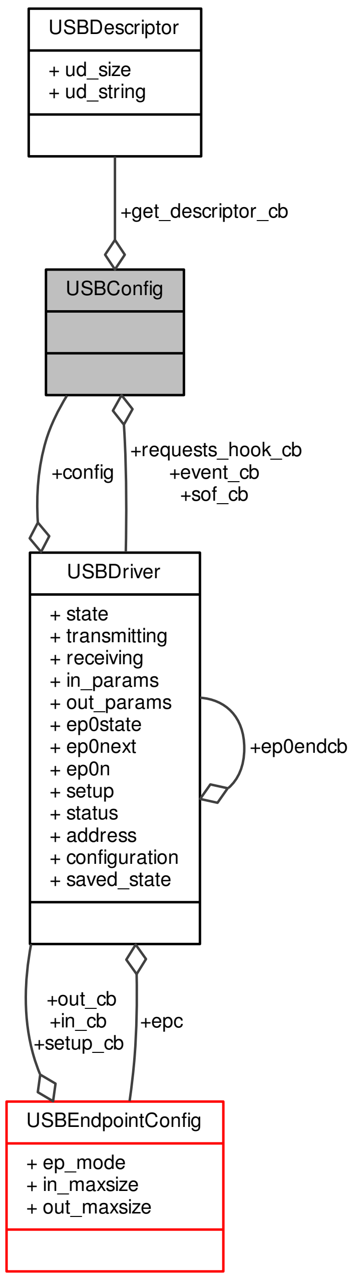
Unpacked CSD v2.0 register from SDC.

8.75 USBConfig Struct Reference

Type of an USB driver configuration structure.

```
#include <hal_usb_lld.h>
```

Collaboration diagram for USBConfig:



Data Fields

- usbeventcb_t event_cb

  USB events callback.
- usbgetdescriptor_t get_descriptor_cb

  Device GET_DESCRIPTOR request callback.
- usbreqhandler_t requests_hook_cb

Requests hook callback.

- usbcallback_t sof_cb

  Start Of Frame callback.

8.75.1 Detailed Description

Type of an USB driver configuration structure.

8.75.2 Field Documentation

8.75.2.1 usbeventcb_t USBConfig::event_cb

USB events callback.

This callback is invoked when an USB driver event is registered.

8.75.2.2 usbgetdescriptor_t USBConfig::get_descriptor_cb

Device GET_DESCRIPTOR request callback.

Note

This callback is mandatory and cannot be set to `NULL`.

8.75.2.3 usbreqhandler_t USBConfig::requests_hook_cb

Requests hook callback.

This hook allows to be notified of standard requests or to handle non standard requests.

8.75.2.4 usbcallback_t USBConfig::sof_cb

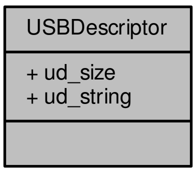
Start Of Frame callback.

8.76 USBDescriptor Struct Reference

Type of an USB descriptor.

`#include <hal_usb.h>`

Collaboration diagram for USBDescriptor:



Data Fields

- size_t ud_size

  Descriptor size in unicode characters.
- const uint8_t ∗ ud_string

  Pointer to the descriptor.

8.76.1 Detailed Description

Type of an USB descriptor.

8.76.2 Field Documentation

8.76.2.1 size_t USBDescriptor::ud_size

Descriptor size in unicode characters.

8.76.2.2 const uint8_t∗ USBDescriptor::ud_string

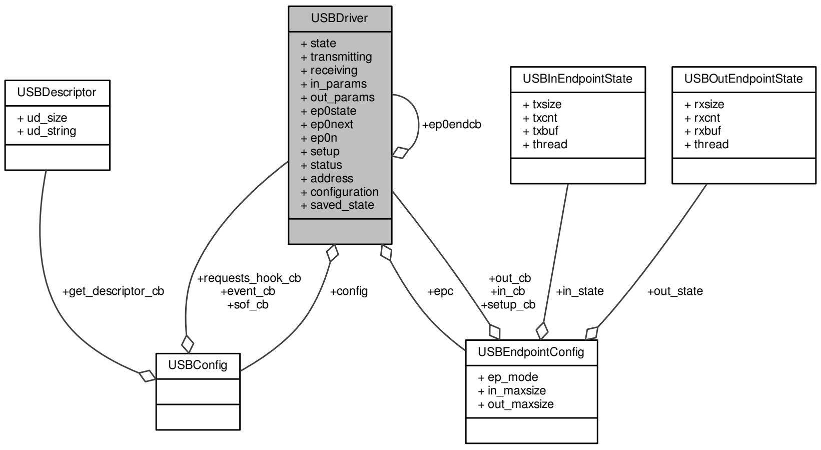
Pointer to the descriptor.

8.77 USBDriver Struct Reference

Structure representing an USB driver.

```
#include <hal_usb_lld.h>
```

Collaboration diagram for USBDriver:



Data Fields

- usbstate_t state

  Driver state.
- const USBConfig ∗ config

Current configuration data.
- uint16_t transmitting

  Bit map of the transmitting IN endpoints.
- uint16_t receiving

  Bit map of the receiving OUT endpoints.
- const USBEndpointConfig ∗ epc [USB_MAX_ENDPOINTS+1]

  Active endpoints configurations.
- void ∗ in_params [USB_MAX_ENDPOINTS]

  Fields available to user, it can be used to associate an application-defined handler to an IN endpoint.
- void ∗ out_params [USB_MAX_ENDPOINTS]

  Fields available to user, it can be used to associate an application-defined handler to an OUT endpoint.
- usbep0state_t ep0state

  Endpoint 0 state.
- uint8_t ∗ ep0next

  Next position in the buffer to be transferred through endpoint 0.
- size_t ep0n

  Number of bytes yet to be transferred through endpoint 0.
- usbcallback_t ep0endcb

  Endpoint 0 end transaction callback.
- uint8_t setup [8]

  Setup packet buffer.
- uint16_t status

  Current USB device status.
- uint8_t address

  Assigned USB address.
- uint8_t configuration

  Current USB device configuration.
- usbstate_t saved_state

  State of the driver when a suspend happened.

8.77.1 Detailed Description

Structure representing an USB driver.

8.77.2 Field Documentation

8.77.2.1 usbstate_t USBDriver::state

Driver state.

8.77.2.2 const USBConfig∗ USBDriver::config

Current configuration data.

8.77.2.3 uint16_t USBDriver::transmitting

Bit map of the transmitting IN endpoints.

8.77.2.4 uint16_t USBDriver::receiving

Bit map of the receiving OUT endpoints.

8.77.2.5 const USBEndpointConfig∗ USBDriver::epc[USB_MAX_ENDPOINTS+1]

Active endpoints configurations.

8.77.2.6 void∗ USBDriver::in_params[USB_MAX_ENDPOINTS]

Fields available to user, it can be used to associate an application-defined handler to an IN endpoint.

Note

The base index is one, the endpoint zero does not have a reserved element in this array.

8.77.2.7 void∗ USBDriver::out_params[USB_MAX_ENDPOINTS]

Fields available to user, it can be used to associate an application-defined handler to an OUT endpoint.

Note

The base index is one, the endpoint zero does not have a reserved element in this array.

8.77.2.8 usbep0state_t USBDriver::ep0state

Endpoint 0 state.

8.77.2.9 uint8_t∗ USBDriver::ep0next

Next position in the buffer to be transferred through endpoint 0.

8.77.2.10 size_t USBDriver::ep0n

Number of bytes yet to be transferred through endpoint 0.

8.77.2.11 usbcallback_t USBDriver::ep0endcb

Endpoint 0 end transaction callback.

8.77.2.12 uint8_t USBDriver::setup[8]

Setup packet buffer.

8.77.2.13 uint16_t USBDriver::status

Current USB device status.

8.77.2.14 uint8_t USBDriver::address

Assigned USB address.

8.77.2.15 uint8_t USBDriver::configuration

Current USB device configuration.

8.77.2.16 usbstate_t USBDriver::saved_state

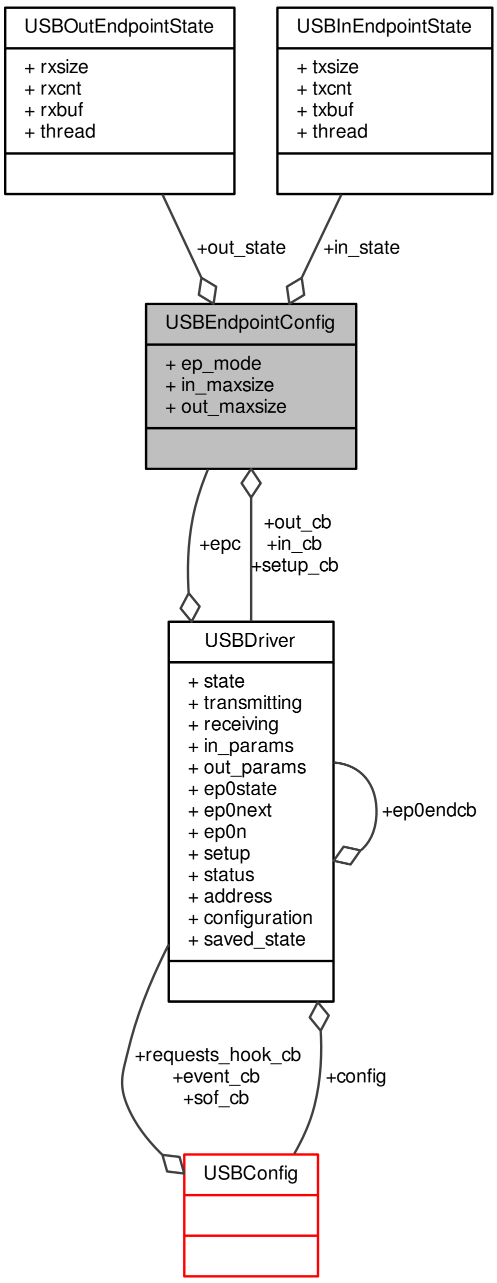
State of the driver when a suspend happened.

8.78 USBEndpointConfig Struct Reference

Type of an USB endpoint configuration structure.

```
#include <hal_usb_lld.h>
```

Collaboration diagram for USBEndpointConfig:



Data Fields

- uint32_t ep_mode

  Type and mode of the endpoint.
- usbepcallback_t setup_cb

  Setup packet notification callback.
- usbepcallback_t in_cb

IN endpoint notification callback.

- usbepcallback_t out_cb

  OUT endpoint notification callback.
- uint16_t in_maxsize

  IN endpoint maximum packet size.
- uint16_t out_maxsize

  OUT endpoint maximum packet size.
- USBInEndpointState ∗ in_state

  `USBEndpointState` associated to the IN endpoint.
- USBOutEndpointState ∗ out_state

  `USBEndpointState` associated to the OUT endpoint.

8.78.1 Detailed Description

Type of an USB endpoint configuration structure.

Note

Platform specific restrictions may apply to endpoints.

8.78.2 Field Documentation

8.78.2.1 uint32_t USBEndpointConfig::ep_mode

Type and mode of the endpoint.

8.78.2.2 usbepcallback_t USBEndpointConfig::setup_cb

Setup packet notification callback.

This callback is invoked when a setup packet has been received.

Postcondition

The application must immediately call `usbReadPacket()` in order to access the received packet.

Note

This field is only valid for `USB_EP_MODE_TYPE_CTRL` endpoints, it should be set to `NULL` for other endpoint types.

8.78.2.3 usbepcallback_t USBEndpointConfig::in_cb

IN endpoint notification callback.

This field must be set to `NULL` if the IN endpoint is not used.

8.78.2.4 usbepcallback_t USBEndpointConfig::out_cb

OUT endpoint notification callback.

This field must be set to `NULL` if the OUT endpoint is not used.

8.78.2.5 uint16_t USBEndpointConfig::in_maxsize

IN endpoint maximum packet size.

This field must be set to zero if the IN endpoint is not used.

8.78.2.6 uint16_t USBEndpointConfig::out_maxsize

OUT endpoint maximum packet size.

This field must be set to zero if the OUT endpoint is not used.

8.78.2.7 USBInEndpointState∗ USBEndpointConfig::in_state

`USBEndpointState` associated to the IN endpoint.

This structure maintains the state of the IN endpoint.

8.78.2.8 USBOutEndpointState∗ USBEndpointConfig::out_state

`USBEndpointState` associated to the OUT endpoint.

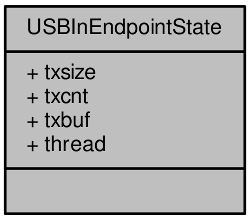
This structure maintains the state of the OUT endpoint.

8.79 USBInEndpointState Struct Reference

Type of an IN endpoint state structure.

`#include <hal_usb_lld.h>`

Collaboration diagram for USBInEndpointState:



Data Fields

- size_t txsize

  Requested transmit transfer size.
- size_t txcnt

  Transmitted bytes so far.
- const uint8_t ∗ txbuf

  Pointer to the transmission linear buffer.

- thread_reference_t thread

  Waiting thread.

8.79.1 Detailed Description

Type of an IN endpoint state structure.

8.79.2 Field Documentation

8.79.2.1 size_t USBInEndpointState::txsize

Requested transmit transfer size.

8.79.2.2 size_t USBInEndpointState::txcnt

Transmitted bytes so far.

8.79.2.3 const uint8_t∗ USBInEndpointState::txbuf

Pointer to the transmission linear buffer.

8.79.2.4 thread_reference_t USBInEndpointState::thread

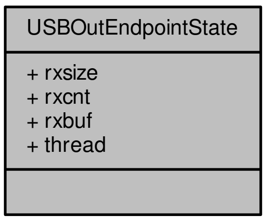
Waiting thread.

8.80 USBOutEndpointState Struct Reference

Type of an OUT endpoint state structure.

`#include <hal_usb_lld.h>`

Collaboration diagram for USBOutEndpointState:



Data Fields

- size_t rxsize

  Requested receive transfer size.

- size_t rxcnt

  Received bytes so far.
- uint8_t ∗ rxbuf

  Pointer to the receive linear buffer.
- thread_reference_t thread

  Waiting thread.

8.80.1 Detailed Description

Type of an OUT endpoint state structure.

8.80.2 Field Documentation

8.80.2.1 size_t USBOutEndpointState::rxsize

Requested receive transfer size.

8.80.2.2 size_t USBOutEndpointState::rxcnt

Received bytes so far.

8.80.2.3 uint8_t∗ USBOutEndpointState::rxbuf

Pointer to the receive linear buffer.

8.80.2.4 thread_reference_t USBOutEndpointState::thread

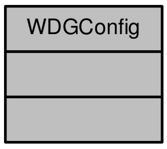
Waiting thread.

8.81 WDGConfig Struct Reference

Driver configuration structure.

`#include <hal_wdg_lld.h>`

Collaboration diagram for WDGConfig:



8.81.1 Detailed Description

Driver configuration structure.

Note

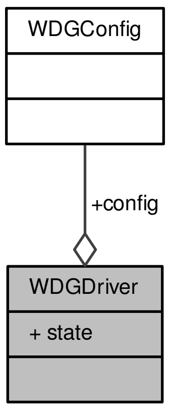
It could be empty on some architectures.

8.82 WDGDriver Struct Reference

Structure representing an WDG driver.

`#include <hal_wdg_lld.h>`

Collaboration diagram for WDGDriver:



Data Fields

- wdgstate_t state

    Driver state.
- const WDGConfig ∗ config

    Current configuration data.

8.82.1 Detailed Description

Structure representing an WDG driver.

8.82.2 Field Documentation

8.82.2.1 wdgstate_t WDGDriver::state

Driver state.

8.82.2.2 const WDGConfig* WDGDriver::config

Current configuration data.

Chapter 9

File Documentation

9.1 hal.c File Reference

HAL subsystem code.

```
#include "hal.h"
```

Functions

- void halInit (void)

  HAL initialization.

9.1.1 Detailed Description

HAL subsystem code.

9.2 hal.h File Reference

HAL subsystem header.

```
#include "osal.h"
#include "board.h"
#include "halconf.h"
#include "hal_lld.h"
#include "hal_streams.h"
#include "hal_channels.h"
#include "hal_files.h"
#include "hal_ioblock.h"
#include "hal_mmcsd.h"
#include "hal_buffers.h"
#include "hal_queues.h"
#include "hal_pal.h"
#include "hal_adc.h"
#include "hal_can.h"
#include "hal_dac.h"
#include "hal_ext.h"
#include "hal_gpt.h"
#include "hal_i2c.h"
#include "hal_i2s.h"
#include "hal_icu.h"
#include "hal_mac.h"
#include "hal_pwm.h"
#include "hal_qspi.h"
#include "hal_rtc.h"
#include "hal_serial.h"
#include "hal_sdc.h"
#include "hal_spi.h"
#include "hal_uart.h"
#include "hal_usb.h"
#include "hal_wdg.h"
#include "hal_mmc_spi.h"
#include "hal_serial_usb.h"
#include "hal_community.h"
```

Macros

- #define _CHIBIOS_HAL_

  ChibiOS/HAL identification macro.
- #define CH_HAL_STABLE 1

  Stable release flag.

ChibiOS/HAL version identification

- #define HAL_VERSION "5.0.0"

  HAL version string.
- #define CH_HAL_MAJOR 5

  HAL version major number.
- #define CH_HAL_MINOR 0

  HAL version minor number.
- #define CH_HAL_PATCH 0

  HAL version patch number.

Return codes

- #define **HAL_SUCCESS** false
- #define **HAL_FAILED** true

Functions

- void halInit (void)

  HAL initialization.

9.2.1 Detailed Description

HAL subsystem header.

9.3 hal_adc.c File Reference

ADC Driver code.

```
#include "hal.h"
```

Functions

- void adcInit (void)

  ADC Driver initialization.
- void adcObjectInit (ADCDriver ∗adcp)

  Initializes the standard part of a `ADCDriver` structure.
- void adcStart (ADCDriver ∗adcp, const ADCConfig ∗config)

  Configures and activates the ADC peripheral.
- void adcStop (ADCDriver ∗adcp)

  Deactivates the ADC peripheral.
- void adcStartConversion (ADCDriver ∗adcp, const ADCConversionGroup ∗grpp, adcsample_t ∗samples, size_t depth)

  Starts an ADC conversion.
- void adcStartConversionI (ADCDriver ∗adcp, const ADCConversionGroup ∗grpp, adcsample_t ∗samples, size_t depth)

  Starts an ADC conversion.
- void adcStopConversion (ADCDriver ∗adcp)

  Stops an ongoing conversion.
- void adcStopConversionI (ADCDriver ∗adcp)

  Stops an ongoing conversion.
- msg_t adcConvert (ADCDriver ∗adcp, const ADCConversionGroup ∗grpp, adcsample_t ∗samples, size_t depth)

  Performs an ADC conversion.
- void adcAcquireBus (ADCDriver ∗adcp)

  Gains exclusive access to the ADC peripheral.
- void adcReleaseBus (ADCDriver ∗adcp)

  Releases exclusive access to the ADC peripheral.

9.3.1 Detailed Description

ADC Driver code.

9.4 hal_adc.h File Reference

ADC Driver macros and structures.

```
#include "hal_adc_lld.h"
```

Macros

ADC configuration options

- #define ADC_USE_WAIT TRUE

  Enables synchronous APIs.
- #define ADC_USE_MUTUAL_EXCLUSION TRUE

  Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.

Low level driver helper macros

- #define _adc_reset_i(adcp) osalThreadResumeI(&(adcp)->thread, MSG_RESET)

  Resumes a thread waiting for a conversion completion.
- #define _adc_reset_s(adcp) osalThreadResumeS(&(adcp)->thread, MSG_RESET)

  Resumes a thread waiting for a conversion completion.
- #define _adc_wakeup_isr(adcp)

  Wakes up the waiting thread.
- #define _adc_timeout_isr(adcp)

  Wakes up the waiting thread with a timeout message.
- #define _adc_isr_half_code(adcp)

  Common ISR code, half buffer event.
- #define _adc_isr_full_code(adcp)

  Common ISR code, full buffer event.
- #define _adc_isr_error_code(adcp, err)

  Common ISR code, error event.

Enumerations

Functions

- void adcInit (void)

  ADC Driver initialization.
- void adcObjectInit (ADCDriver ∗adcp)

  Initializes the standard part of a `ADCDriver` structure.
- void adcStart (ADCDriver ∗adcp, const ADCConfig ∗config)

  Configures and activates the ADC peripheral.
- void adcStop (ADCDriver ∗adcp)

  Deactivates the ADC peripheral.
- void adcStartConversion (ADCDriver ∗adcp, const ADCConversionGroup ∗grpp, adcsample_t ∗samples, size_t depth)

  Starts an ADC conversion.
- void adcStartConversionI (ADCDriver ∗adcp, const ADCConversionGroup ∗grpp, adcsample_t ∗samples, size_t depth)

  Starts an ADC conversion.
- void adcStopConversion (ADCDriver ∗adcp)

  Stops an ongoing conversion.
- void adcStopConversionI (ADCDriver ∗adcp)

Stops an ongoing conversion.

- msg_t adcConvert (ADCDriver ∗adcp, const ADCConversionGroup ∗grpp, adcsample_t ∗samples, size_t depth)

    Performs an ADC conversion.
- void adcAcquireBus (ADCDriver ∗adcp)

    Gains exclusive access to the ADC peripheral.
- void adcReleaseBus (ADCDriver ∗adcp)

    Releases exclusive access to the ADC peripheral.

9.4.1 Detailed Description

ADC Driver macros and structures.

9.5 hal_adc_lld.c File Reference

PLATFORM ADC subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void adc_lld_init (void)

    Low level ADC driver initialization.
- void adc_lld_start (ADCDriver ∗adcp)

    Configures and activates the ADC peripheral.
- void adc_lld_stop (ADCDriver ∗adcp)

    Deactivates the ADC peripheral.
- void adc_lld_start_conversion (ADCDriver ∗adcp)

    Starts an ADC conversion.
- void adc_lld_stop_conversion (ADCDriver ∗adcp)

    Stops an ongoing conversion.

Variables

- ADCDriver ADCD1

    ADC1 driver identifier.

9.5.1 Detailed Description

PLATFORM ADC subsystem low level driver source.

9.6 hal_adc_lld.h File Reference

PLATFORM ADC subsystem low level driver header.

Data Structures

- struct ADCConversionGroup

  Conversion group configuration structure.
- struct ADCConfig

  Driver configuration structure.
- struct ADCDriver

  Structure representing an ADC driver.

Macros

PLATFORM configuration options

- #define PLATFORM_ADC_USE_ADC1 FALSE

  ADC1 driver enable switch.

Typedefs

- typedef uint16_t adcsample_t

  ADC sample data type.
- typedef uint16_t adc_channels_num_t

  Channels number in a conversion group.
- typedef struct ADCDriver ADCDriver

  Type of a structure representing an ADC driver.
- typedef void(∗ adccallback_t) (ADCDriver ∗adcp, adcsample_t ∗buffer, size_t n)

  ADC notification callback type.
- typedef void(∗ adcerrorcallback_t) (ADCDriver ∗adcp, adcerror_t err)

  ADC error callback type.

Enumerations

Functions

- void adc_lld_init (void)

  Low level ADC driver initialization.
- void adc_lld_start (ADCDriver ∗adcp)

  Configures and activates the ADC peripheral.
- void adc_lld_stop (ADCDriver ∗adcp)

  Deactivates the ADC peripheral.
- void adc_lld_start_conversion (ADCDriver ∗adcp)

  Starts an ADC conversion.
- void adc_lld_stop_conversion (ADCDriver ∗adcp)

  Stops an ongoing conversion.

9.6.1 Detailed Description

PLATFORM ADC subsystem low level driver header.

9.7 hal_buffers.c File Reference

I/O Buffers code.

```
#include <string.h>
#include "hal.h"
```

Functions

- void ibqObjectInit (input_buffers_queue_t ∗ibqp, bool suspended, uint8_t ∗bp, size_t size, size_t n, bqnotify↩_t infy, void ∗link)

    Initializes an input buffers queue object.
- void ibqResetI (input_buffers_queue_t ∗ibqp)

    Resets an input buffers queue.
- uint8_t ∗ ibqGetEmptyBufferI (input_buffers_queue_t ∗ibqp)

    Gets the next empty buffer from the queue.
- void ibqPostFullBufferI (input_buffers_queue_t ∗ibqp, size_t size)

    Posts a new filled buffer to the queue.
- msg_t ibqGetFullBufferTimeout (input_buffers_queue_t ∗ibqp, systime_t timeout)

    Gets the next filled buffer from the queue.
- msg_t ibqGetFullBufferTimeoutS (input_buffers_queue_t ∗ibqp, systime_t timeout)

    Gets the next filled buffer from the queue.
- void ibqReleaseEmptyBuffer (input_buffers_queue_t ∗ibqp)

    Releases the buffer back in the queue.
- void ibqReleaseEmptyBufferS (input_buffers_queue_t ∗ibqp)

    Releases the buffer back in the queue.
- msg_t ibqGetTimeout (input_buffers_queue_t ∗ibqp, systime_t timeout)

    Input queue read with timeout.
- size_t ibqReadTimeout (input_buffers_queue_t ∗ibqp, uint8_t ∗bp, size_t n, systime_t timeout)

    Input queue read with timeout.
- void obqObjectInit (output_buffers_queue_t ∗obqp, bool suspended, uint8_t ∗bp, size_t size, size_t n, bqnotify_t onfy, void ∗link)

    Initializes an output buffers queue object.
- void obqResetI (output_buffers_queue_t ∗obqp)

    Resets an output buffers queue.
- uint8_t ∗ obqGetFullBufferI (output_buffers_queue_t ∗obqp, size_t ∗sizep)

    Gets the next filled buffer from the queue.
- void obqReleaseEmptyBufferI (output_buffers_queue_t ∗obqp)

    Releases the next filled buffer back in the queue.
- msg_t obqGetEmptyBufferTimeout (output_buffers_queue_t ∗obqp, systime_t timeout)

    Gets the next empty buffer from the queue.
- msg_t obqGetEmptyBufferTimeoutS (output_buffers_queue_t ∗obqp, systime_t timeout)

    Gets the next empty buffer from the queue.
- void obqPostFullBuffer (output_buffers_queue_t ∗obqp, size_t size)

    Posts a new filled buffer to the queue.
- void obqPostFullBufferS (output_buffers_queue_t ∗obqp, size_t size)

    Posts a new filled buffer to the queue.
- msg_t obqPutTimeout (output_buffers_queue_t ∗obqp, uint8_t b, systime_t timeout)

    Output queue write with timeout.
- size_t obqWriteTimeout (output_buffers_queue_t ∗obqp, const uint8_t ∗bp, size_t n, systime_t timeout)

    Output queue write with timeout.

- bool obqTryFlushI (output_buffers_queue_t ∗obqp)

  Flushes the current, partially filled, buffer to the queue.
- void obqFlush (output_buffers_queue_t ∗obqp)

  Flushes the current, partially filled, buffer to the queue.

9.7.1 Detailed Description

I/O Buffers code.

9.8 hal_buffers.h File Reference

I/O Buffers macros and structures.

Data Structures

- struct io_buffers_queue

  Structure of a generic buffers queue.

Macros

- #define BQ_BUFFER_SIZE(n, size) (((size_t)(size) + sizeof (size_t)) ∗ (size_t)(n))

  Computes the size of a buffers queue buffer size.

Macro Functions

- #define bqSizeX(bqp) ((bqp)->bn)

  Returns the queue's number of buffers.
- #define bqSpaceI(bqp) ((bqp)->bcounter)

  Return the ready buffers number.
- #define bqGetLinkX(bqp) ((bqp)->link)

  Returns the queue application-defined link.
- #define bqIsSuspendedX(bqp) ((bqp)->suspended)

  Return the suspended state of the queue.
- #define bqSuspendI(bqp)

  Puts the queue in suspended state.
- #define bqResumeX(bqp)

  Resumes normal queue operations.
- #define ibqIsEmptyI(ibqp) ((bool)(bqSpaceI(ibqp) == 0U))

  Evaluates to `TRUE` if the specified input buffers queue is empty.
- #define ibqIsFullI(ibqp)

  Evaluates to `TRUE` if the specified input buffers queue is full.
- #define obqIsEmptyI(obqp)

  Evaluates to `true` if the specified output buffers queue is empty.
- #define obqIsFullI(obqp) ((bool)(bqSpaceI(obqp) == 0U))

  Evaluates to `true` if the specified output buffers queue is full.

Typedefs

- typedef struct io_buffers_queue io_buffers_queue_t

  Type of a generic queue of buffers.
- typedef void(∗ bqnotify_t) (io_buffers_queue_t ∗bqp)

  Double buffer notification callback type.

- typedef io_buffers_queue_t input_buffers_queue_t

    Type of an input buffers queue.
- typedef io_buffers_queue_t output_buffers_queue_t

    Type of an output buffers queue.

Functions

- void ibqObjectInit (input_buffers_queue_t ∗ibqp, bool suspended, uint8_t ∗bp, size_t size, size_t n, bqnotify↩ _t infy, void ∗link)

    Initializes an input buffers queue object.
- void ibqResetI (input_buffers_queue_t ∗ibqp)

    Resets an input buffers queue.
- uint8_t ∗ ibqGetEmptyBufferI (input_buffers_queue_t ∗ibqp)

    Gets the next empty buffer from the queue.
- void ibqPostFullBufferI (input_buffers_queue_t ∗ibqp, size_t size)

    Posts a new filled buffer to the queue.
- msg_t ibqGetFullBufferTimeout (input_buffers_queue_t ∗ibqp, systime_t timeout)

    Gets the next filled buffer from the queue.
- msg_t ibqGetFullBufferTimeoutS (input_buffers_queue_t ∗ibqp, systime_t timeout)

    Gets the next filled buffer from the queue.
- void ibqReleaseEmptyBuffer (input_buffers_queue_t ∗ibqp)

    Releases the buffer back in the queue.
- void ibqReleaseEmptyBufferS (input_buffers_queue_t ∗ibqp)

    Releases the buffer back in the queue.
- msg_t ibqGetTimeout (input_buffers_queue_t ∗ibqp, systime_t timeout)

    Input queue read with timeout.
- size_t ibqReadTimeout (input_buffers_queue_t ∗ibqp, uint8_t ∗bp, size_t n, systime_t timeout)

    Input queue read with timeout.
- void obqObjectInit (output_buffers_queue_t ∗obqp, bool suspended, uint8_t ∗bp, size_t size, size_t n, bqnotify_t onfy, void ∗link)

    Initializes an output buffers queue object.
- void obqResetI (output_buffers_queue_t ∗obqp)

    Resets an output buffers queue.
- uint8_t ∗ obqGetFullBufferI (output_buffers_queue_t ∗obqp, size_t ∗sizep)

    Gets the next filled buffer from the queue.
- void obqReleaseEmptyBufferI (output_buffers_queue_t ∗obqp)

    Releases the next filled buffer back in the queue.
- msg_t obqGetEmptyBufferTimeout (output_buffers_queue_t ∗obqp, systime_t timeout)

    Gets the next empty buffer from the queue.
- msg_t obqGetEmptyBufferTimeoutS (output_buffers_queue_t ∗obqp, systime_t timeout)

    Gets the next empty buffer from the queue.
- void obqPostFullBuffer (output_buffers_queue_t ∗obqp, size_t size)

    Posts a new filled buffer to the queue.
- void obqPostFullBufferS (output_buffers_queue_t ∗obqp, size_t size)

    Posts a new filled buffer to the queue.
- msg_t obqPutTimeout (output_buffers_queue_t ∗obqp, uint8_t b, systime_t timeout)

    Output queue write with timeout.
- size_t obqWriteTimeout (output_buffers_queue_t ∗obqp, const uint8_t ∗bp, size_t n, systime_t timeout)

    Output queue write with timeout.
- bool obqTryFlushI (output_buffers_queue_t ∗obqp)

    Flushes the current, partially filled, buffer to the queue.
- void obqFlush (output_buffers_queue_t ∗obqp)

    Flushes the current, partially filled, buffer to the queue.

9.8.1 Detailed Description

I/O Buffers macros and structures.

9.9 hal_can.c File Reference

CAN Driver code.

```
#include "hal.h"
```

Functions

- void canInit (void)

  CAN Driver initialization.
- void canObjectInit (CANDriver ∗canp)

  Initializes the standard part of a `CANDriver` structure.
- void canStart (CANDriver ∗canp, const CANConfig ∗config)

  Configures and activates the CAN peripheral.
- void canStop (CANDriver ∗canp)

  Deactivates the CAN peripheral.
- bool canTryTransmitI (CANDriver ∗canp, canmbx_t mailbox, const CANTxFrame ∗ctfp)

  Can frame transmission attempt.
- bool canTryReceiveI (CANDriver ∗canp, canmbx_t mailbox, CANRxFrame ∗crfp)

  Can frame receive attempt.
- msg_t canTransmitTimeout (CANDriver ∗canp, canmbx_t mailbox, const CANTxFrame ∗ctfp, systime_t timeout)

  Can frame transmission.
- msg_t canReceiveTimeout (CANDriver ∗canp, canmbx_t mailbox, CANRxFrame ∗crfp, systime_t timeout)

  Can frame receive.
- void canSleep (CANDriver ∗canp)

  Enters the sleep mode.
- void canWakeup (CANDriver ∗canp)

  Enforces leaving the sleep mode.

9.9.1 Detailed Description

CAN Driver code.

9.10 hal_can.h File Reference

CAN Driver macros and structures.

```
#include "hal_can_lld.h"
```

Macros

- #define CAN_ANY_MAILBOX 0

    Special mailbox identifier.

CAN status flags

- #define CAN_LIMIT_WARNING 1U

    Errors rate warning.
- #define CAN_LIMIT_ERROR 2U

    Errors rate error.
- #define CAN_BUS_OFF_ERROR 4U

    Bus off condition reached.
- #define CAN_FRAMING_ERROR 8U

    Framing error of some kind on the CAN bus.
- #define CAN_OVERFLOW_ERROR 16U

    Overflow in receive queue.

CAN configuration options

- #define CAN_USE_SLEEP_MODE TRUE

    Sleep mode related APIs inclusion switch.

Macro Functions

- #define CAN_MAILBOX_TO_MASK(mbx) (1U << ((mbx) - 1U))

    Converts a mailbox index to a bit mask.
- #define canTransmit(canp, mailbox, ctfp, timeout) canTransmitTimeout(canp, mailbox, ctfp, timeout)

    Legacy name for `canTransmitTimeout()`.
- #define canReceive(canp, mailbox, crfp, timeout) canReceiveTimeout(canp, mailbox, crfp, timeout)

    Legacy name for `canReceiveTimeout()`.

Enumerations

Functions

- void canInit (void)

    CAN Driver initialization.
- void canObjectInit (CANDriver ∗canp)

    Initializes the standard part of a `CANDriver` structure.
- void canStart (CANDriver ∗canp, const CANConfig ∗config)

    Configures and activates the CAN peripheral.
- void canStop (CANDriver ∗canp)

    Deactivates the CAN peripheral.
- bool canTryTransmitI (CANDriver ∗canp, canmbx_t mailbox, const CANTxFrame ∗ctfp)

    Can frame transmission attempt.
- bool canTryReceiveI (CANDriver ∗canp, canmbx_t mailbox, CANRxFrame ∗crfp)

    Can frame receive attempt.
- msg_t canTransmitTimeout (CANDriver ∗canp, canmbx_t mailbox, const CANTxFrame ∗ctfp, systime_t timeout)

    Can frame transmission.
- msg_t canReceiveTimeout (CANDriver ∗canp, canmbx_t mailbox, CANRxFrame ∗crfp, systime_t timeout)

    Can frame receive.

9.10.1 Detailed Description

CAN Driver macros and structures.

9.11 hal_can_lld.c File Reference

PLATFORM CAN subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void can_lld_init (void)

  Low level CAN driver initialization.
- void can_lld_start (CANDriver ∗canp)

  Configures and activates the CAN peripheral.
- void can_lld_stop (CANDriver ∗canp)

  Deactivates the CAN peripheral.
- bool can_lld_is_tx_empty (CANDriver ∗canp, canmbx_t mailbox)

  Determines whether a frame can be transmitted.
- void can_lld_transmit (CANDriver ∗canp, canmbx_t mailbox, const CANTxFrame ∗ctfp)

  Inserts a frame into the transmit queue.
- bool can_lld_is_rx_nonempty (CANDriver ∗canp, canmbx_t mailbox)

  Determines whether a frame has been received.
- void can_lld_receive (CANDriver ∗canp, canmbx_t mailbox, CANRxFrame ∗crfp)

  Receives a frame from the input queue.
- void can_lld_sleep (CANDriver ∗canp)

  Enters the sleep mode.
- void can_lld_wakeup (CANDriver ∗canp)

  Enforces leaving the sleep mode.

Variables

- CANDriver CAND1

  CAN1 driver identifier.

9.11.1 Detailed Description

PLATFORM CAN subsystem low level driver source.

9.12 hal_can_lld.h File Reference

PLATFORM CAN subsystem low level driver header.

Data Structures

- struct CANTxFrame

  CAN transmission frame.
- struct CANRxFrame

  CAN received frame.
- struct CANConfig

  Driver configuration structure.
- struct CANDriver

  Structure representing an CAN driver.

Macros

- #define CAN_TX_MAILBOXES 1

  Number of transmit mailboxes.
- #define CAN_RX_MAILBOXES 1

  Number of receive mailboxes.

PLATFORM configuration options

- #define PLATFORM_CAN_USE_CAN1 FALSE

  CAN1 driver enable switch.

Typedefs

- typedef uint32_t canmbx_t

  Type of a transmission mailbox index.

Functions

- void can_lld_init (void)

  Low level CAN driver initialization.
- void can_lld_start (CANDriver ∗canp)

  Configures and activates the CAN peripheral.
- void can_lld_stop (CANDriver ∗canp)

  Deactivates the CAN peripheral.
- bool can_lld_is_tx_empty (CANDriver ∗canp, canmbx_t mailbox)

  Determines whether a frame can be transmitted.
- void can_lld_transmit (CANDriver ∗canp, canmbx_t mailbox, const CANTxFrame ∗ctfp)

  Inserts a frame into the transmit queue.
- bool can_lld_is_rx_nonempty (CANDriver ∗canp, canmbx_t mailbox)

  Determines whether a frame has been received.
- void can_lld_receive (CANDriver ∗canp, canmbx_t mailbox, CANRxFrame ∗crfp)

  Receives a frame from the input queue.
- void can_lld_sleep (CANDriver ∗canp)

  Enters the sleep mode.
- void can_lld_wakeup (CANDriver ∗canp)

  Enforces leaving the sleep mode.

9.12.1 Detailed Description

PLATFORM CAN subsystem low level driver header.

9.13 hal_channels.h File Reference

I/O channels access.

Data Structures

- struct BaseChannelVMT

    BaseChannel virtual methods table.
- struct BaseChannel

    Base channel class.
- struct BaseAsynchronousChannelVMT

    BaseAsynchronousChannel virtual methods table.
- struct BaseAsynchronousChannel

    Base asynchronous channel class.

Macros

- #define _base_channel_methods

    BaseChannel specific methods.
- #define _base_channel_data _base_sequential_stream_data

    BaseChannel specific data.
- #define _base_asynchronous_channel_methods _base_channel_methods \

    BaseAsynchronousChannel specific methods.
- #define _base_asynchronous_channel_data

    BaseAsynchronousChannel specific data.

Macro Functions (BaseChannel)

- #define chnPutTimeout(ip, b, time) ((ip)->vmt->putt(ip, b, time))

    Channel blocking byte write with timeout.
- #define chnGetTimeout(ip, time) ((ip)->vmt->gett(ip, time))

    Channel blocking byte read with timeout.
- #define chnWrite(ip, bp, n) streamWrite(ip, bp, n)

    Channel blocking write.
- #define chnWriteTimeout(ip, bp, n, time) ((ip)->vmt->writet(ip, bp, n, time))

    Channel blocking write with timeout.
- #define chnRead(ip, bp, n) streamRead(ip, bp, n)

    Channel blocking read.
- #define chnReadTimeout(ip, bp, n, time) ((ip)->vmt->readt(ip, bp, n, time))

    Channel blocking read with timeout.

I/O status flags added to the event listener

- #define CHN_NO_ERROR (eventflags_t)0

    No pending conditions.
- #define CHN_CONNECTED (eventflags_t)1

    Connection happened.
- #define CHN_DISCONNECTED (eventflags_t)2

    Disconnection happened.

- #define CHN_INPUT_AVAILABLE (eventflags_t)4

  Data available in the input queue.
- #define CHN_OUTPUT_EMPTY (eventflags_t)8

  Output queue empty.
- #define CHN_TRANSMISSION_END (eventflags_t)16

  Transmission end.

Macro Functions (BaseAsynchronousChannel)

- #define chnGetEventSource(ip) (&((ip)->event))

  Returns the I/O condition event source.
- #define chnAddFlagsI(ip, flags)

  Adds status flags to the listeners's flags mask.

9.13.1 Detailed Description

I/O channels access.

This header defines an abstract interface useful to access generic I/O serial devices in a standardized way.

9.14 hal_dac.c File Reference

DAC Driver code.

```
#include "hal.h"
```

Functions

- void dacInit (void)

  DAC Driver initialization.
- void dacObjectInit (DACDriver ∗dacp)

  Initializes the standard part of a `DACDriver` structure.
- void dacStart (DACDriver ∗dacp, const DACConfig ∗config)

  Configures and activates the DAC peripheral.
- void dacStop (DACDriver ∗dacp)

  Deactivates the DAC peripheral.
- void dacPutChannelX (DACDriver ∗dacp, dacchannel_t channel, dacsample_t sample)

  Outputs a value directly on a DAC channel.
- void dacStartConversion (DACDriver ∗dacp, const DACConversionGroup ∗grpp, dacsample_t ∗samples, size_t depth)

  Starts a DAC conversion.
- void dacStartConversionI (DACDriver ∗dacp, const DACConversionGroup ∗grpp, dacsample_t ∗samples, size_t depth)

  Starts a DAC conversion.
- void dacStopConversion (DACDriver ∗dacp)

  Stops an ongoing conversion.
- void dacStopConversionI (DACDriver ∗dacp)

  Stops an ongoing conversion.
- msg_t dacConvert (DACDriver ∗dacp, const DACConversionGroup ∗grpp, dacsample_t ∗samples, size_t depth)

  Performs a DAC conversion.
- void dacAcquireBus (DACDriver ∗dacp)

Gains exclusive access to the DAC bus.
- void dacReleaseBus (DACDriver ∗dacp)

  Releases exclusive access to the DAC bus.

9.14.1 Detailed Description

DAC Driver code.

9.15 hal_dac.h File Reference

DAC Driver macros and structures.

```
#include "hal_dac_lld.h"
```

Macros

DAC configuration options

- #define DAC_USE_WAIT TRUE

  Enables synchronous APIs.
- #define DAC_USE_MUTUAL_EXCLUSION TRUE

  Enables the `dacAcquireBus()` and `dacReleaseBus()` APIs.

Low level driver helper macros

- #define _dac_wait_s(dacp) osalThreadSuspendS(&(dacp)->thread)

  Waits for operation completion.
- #define _dac_reset_i(dacp) osalThreadResumeI(&(dacp)->thread, MSG_RESET)

  Resumes a thread waiting for a conversion completion.
- #define _dac_reset_s(dacp) osalThreadResumeS(&(dacp)->thread, MSG_RESET)

  Resumes a thread waiting for a conversion completion.
- #define _dac_wakeup_isr(dacp)

  Wakes up the waiting thread.
- #define _dac_timeout_isr(dacp)

  Wakes up the waiting thread with a timeout message.
- #define _dac_isr_half_code(dacp)

  Common ISR code, half buffer event.
- #define _dac_isr_full_code(dacp)

  Common ISR code, full buffer event.
- #define _dac_isr_error_code(dacp, err)

  Common ISR code, error event.

Enumerations

Functions

- void dacInit (void)

  DAC Driver initialization.
- void dacObjectInit (DACDriver ∗dacp)

  Initializes the standard part of a `DACDriver` structure.
- void dacStart (DACDriver ∗dacp, const DACConfig ∗config)

  Configures and activates the DAC peripheral.

- void dacStop (DACDriver ∗dacp)

  Deactivates the DAC peripheral.
- void dacPutChannelX (DACDriver ∗dacp, dacchannel_t channel, dacsample_t sample)

  Outputs a value directly on a DAC channel.
- void dacStartConversion (DACDriver ∗dacp, const DACConversionGroup ∗grpp, dacsample_t ∗samples, size_t depth)

  Starts a DAC conversion.
- void dacStartConversionI (DACDriver ∗dacp, const DACConversionGroup ∗grpp, dacsample_t ∗samples, size_t depth)

  Starts a DAC conversion.
- void dacStopConversion (DACDriver ∗dacp)

  Stops an ongoing conversion.
- void dacStopConversionI (DACDriver ∗dacp)

  Stops an ongoing conversion.

9.15.1 Detailed Description

DAC Driver macros and structures.

9.16 hal_dac_lld.c File Reference

PLATFORM DAC subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void dac_lld_init (void)

  Low level DAC driver initialization.
- void dac_lld_start (DACDriver ∗dacp)

  Configures and activates the DAC peripheral.
- void dac_lld_stop (DACDriver ∗dacp)

  Deactivates the DAC peripheral.
- void dac_lld_put_channel (DACDriver ∗dacp, dacchannel_t channel, dacsample_t sample)

  Outputs a value directly on a DAC channel.
- void dac_lld_start_conversion (DACDriver ∗dacp)

  Starts a DAC conversion.
- void dac_lld_stop_conversion (DACDriver ∗dacp)

  Stops an ongoing conversion.

Variables

- DACDriver DACD1

  DAC1 driver identifier.

9.16.1 Detailed Description

PLATFORM DAC subsystem low level driver source.

9.17 hal_dac_lld.h File Reference

PLATFORM DAC subsystem low level driver header.

Data Structures

- struct DACConversionGroup

  DAC Conversion group structure.
- struct DACConfig

  Driver configuration structure.
- struct DACDriver

  Structure representing a DAC driver.

Macros

- #define DAC_MAX_CHANNELS 2

  Maximum number of DAC channels per unit.

Configuration options

- #define PLATFORM_DAC_USE_DAC1 FALSE

  DAC1 CH1 driver enable switch.

Typedefs

- typedef uint32_t dacchannel_t

  Type of a DAC channel index.
- typedef struct DACDriver DACDriver

  Type of a structure representing an DAC driver.
- typedef uint16_t dacsample_t

  Type representing a DAC sample.
- typedef void(∗ daccallback_t) (DACDriver ∗dacp, dacsample_t ∗buffer, size_t n)

  DAC notification callback type.
- typedef void(∗ dacerrorcallback_t) (DACDriver ∗dacp, dacerror_t err)

  ADC error callback type.

Enumerations

Functions

- void dac_lld_init (void)

  Low level DAC driver initialization.
- void dac_lld_start (DACDriver ∗dacp)

  Configures and activates the DAC peripheral.
- void dac_lld_stop (DACDriver ∗dacp)

  Deactivates the DAC peripheral.
- void dac_lld_put_channel (DACDriver ∗dacp, dacchannel_t channel, dacsample_t sample)

  Outputs a value directly on a DAC channel.
- void dac_lld_start_conversion (DACDriver ∗dacp)

  Starts a DAC conversion.
- void dac_lld_stop_conversion (DACDriver ∗dacp)

  Stops an ongoing conversion.

9.17.1 Detailed Description

PLATFORM DAC subsystem low level driver header.

9.18 hal_ext.c File Reference

EXT Driver code.

```
#include "hal.h"
```

Functions

- void extInit (void)

  EXT Driver initialization.
- void extObjectInit (EXTDriver ∗extp)

  Initializes the standard part of a `EXTDriver` structure.
- void extStart (EXTDriver ∗extp, const EXTConfig ∗config)

  Configures and activates the EXT peripheral.
- void extStop (EXTDriver ∗extp)

  Deactivates the EXT peripheral.
- void extChannelEnable (EXTDriver ∗extp, expchannel_t channel)

  Enables an EXT channel.
- void extChannelDisable (EXTDriver ∗extp, expchannel_t channel)

  Disables an EXT channel.
- void extSetChannelModel (EXTDriver ∗extp, expchannel_t channel, const EXTChannelConfig ∗extcp)

  Changes the operation mode of a channel.

9.18.1 Detailed Description

EXT Driver code.

9.19 hal_ext.h File Reference

EXT Driver macros and structures.

```
#include "hal_ext_lld.h"
```

Macros

EXT channel modes

- #define EXT_CH_MODE_EDGES_MASK 3U

  Mask of edges field.
- #define EXT_CH_MODE_DISABLED 0U

  Channel disabled.
- #define EXT_CH_MODE_RISING_EDGE 1U

  Rising edge callback.
- #define EXT_CH_MODE_FALLING_EDGE 2U

  Falling edge callback.

- #define EXT_CH_MODE_BOTH_EDGES 3U
  Both edges callback.
- #define EXT_CH_MODE_LOW_LEVEL 5U
  low level callback.
- #define EXT_CH_MODE_AUTOSTART 4U
  Channel started automatically on driver start.

Macro Functions

- #define extChannelEnableI(extp, channel) ext_lld_channel_enable(extp, channel)
  Enables an EXT channel.
- #define extChannelDisableI(extp, channel) ext_lld_channel_disable(extp, channel)
  Disables an EXT channel.
- #define extSetChannelMode(extp, channel, extcp)
  Changes the operation mode of a channel.

Typedefs

- typedef struct EXTDriver EXTDriver
  Type of a structure representing a EXT driver.

Enumerations

Functions

- void extInit (void)
  EXT Driver initialization.
- void extObjectInit (EXTDriver ∗extp)
  Initializes the standard part of a `EXTDriver` structure.
- void extStart (EXTDriver ∗extp, const EXTConfig ∗config)
  Configures and activates the EXT peripheral.
- void extStop (EXTDriver ∗extp)
  Deactivates the EXT peripheral.
- void extChannelEnable (EXTDriver ∗extp, expchannel_t channel)
  Enables an EXT channel.
- void extChannelDisable (EXTDriver ∗extp, expchannel_t channel)
  Disables an EXT channel.
- void extSetChannelModeI (EXTDriver ∗extp, expchannel_t channel, const EXTChannelConfig ∗extcp)
  Changes the operation mode of a channel.

9.19.1 Detailed Description

EXT Driver macros and structures.

9.20 hal_ext_lld.c File Reference

PLATFORM EXT subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void ext_lld_init (void)
  Low level EXT driver initialization.
- void ext_lld_start (EXTDriver ∗extp)
  Configures and activates the EXT peripheral.
- void ext_lld_stop (EXTDriver ∗extp)
  Deactivates the EXT peripheral.
- void ext_lld_channel_enable (EXTDriver ∗extp, expchannel_t channel)
  Enables an EXT channel.
- void ext_lld_channel_disable (EXTDriver ∗extp, expchannel_t channel)
  Disables an EXT channel.

Variables

- EXTDriver EXTD1
  EXT1 driver identifier.

9.20.1 Detailed Description

PLATFORM EXT subsystem low level driver source.

9.21 hal_ext_lld.h File Reference

PLATFORM EXT subsystem low level driver header.

Data Structures

- struct EXTChannelConfig
  Channel configuration structure.
- struct EXTConfig
  Driver configuration structure.
- struct EXTDriver
  Structure representing an EXT driver.

Macros

- #define EXT_MAX_CHANNELS 20
  Available number of EXT channels.

PLATFORM configuration options

- #define PLATFORM_EXT_USE_EXT1 FALSE
  EXT driver enable switch.

Typedefs

- typedef uint32_t expchannel_t
  EXT channel identifier.
- typedef void(∗ extcallback_t) (EXTDriver ∗extp, expchannel_t channel)
  Type of an EXT generic notification callback.

Functions

- void ext_lld_init (void)

  Low level EXT driver initialization.
- void ext_lld_start (EXTDriver ∗extp)

  Configures and activates the EXT peripheral.
- void ext_lld_stop (EXTDriver ∗extp)

  Deactivates the EXT peripheral.
- void ext_lld_channel_enable (EXTDriver ∗extp, expchannel_t channel)

  Enables an EXT channel.
- void ext_lld_channel_disable (EXTDriver ∗extp, expchannel_t channel)

  Disables an EXT channel.

9.21.1 Detailed Description

PLATFORM EXT subsystem low level driver header.

9.22 hal_files.h File Reference

Data files.

Data Structures

- struct FileStreamVMT

  `FileStream` virtual methods table.
- struct FileStream

  Base file stream class.

Macros

- #define _file_stream_methods

  FileStream specific methods.
- #define _file_stream_data _base_sequential_stream_data

  `FileStream` specific data.

Files return codes

- #define FILE_OK STM_OK

  No error return code.
- #define FILE_ERROR STM_TIMEOUT

  Error code from the file stream methods.
- #define FILE_EOF STM_RESET

  End-of-file condition for file get/put methods.

Macro Functions (FileStream)

- #define fileStreamWrite(ip, bp, n) streamWrite(ip, bp, n)

  File stream write.
- #define fileStreamRead(ip, bp, n) streamRead(ip, bp, n)

  File stream read.
- #define fileStreamPut(ip, b) streamPut(ip, b)

  File stream blocking byte write.
- #define fileStreamGet(ip) streamGet(ip)
  File stream blocking byte read.
- #define fileStreamClose(ip) ((ip)->vmt->close(ip))
  File Stream close.
- #define fileStreamGetError(ip) ((ip)->vmt->geterror(ip))
  Returns an implementation dependent error code.
- #define fileStreamGetSize(ip) ((ip)->vmt->getsize(ip))
  Returns the current file size.
- #define fileStreamGetPosition(ip) ((ip)->vmt->getposition(ip))
  Returns the current file pointer position.
- #define fileStreamSeek(ip, offset) ((ip)->vmt->lseek(ip, offset))
  Moves the file current pointer to an absolute position.

Typedefs

- typedef uint32_t fileoffset_t
  File offset type.

9.22.1 Detailed Description

Data files.

This header defines abstract interfaces useful to access generic data files in a standardized way.

9.23 hal_gpt.c File Reference

GPT Driver code.

`#include "hal.h"`

Functions

- void gptInit (void)
  GPT Driver initialization.
- void gptObjectInit (GPTDriver ∗gptp)
  Initializes the standard part of a `GPTDriver` structure.
- void gptStart (GPTDriver ∗gptp, const GPTConfig ∗config)
  Configures and activates the GPT peripheral.
- void gptStop (GPTDriver ∗gptp)
  Deactivates the GPT peripheral.
- void gptChangeInterval (GPTDriver ∗gptp, gptcnt_t interval)
  Changes the interval of GPT peripheral.
- void gptStartContinuous (GPTDriver ∗gptp, gptcnt_t interval)
  Starts the timer in continuous mode.
- void gptStartContinuousI (GPTDriver ∗gptp, gptcnt_t interval)
  Starts the timer in continuous mode.
- void gptStartOneShot (GPTDriver ∗gptp, gptcnt_t interval)
  Starts the timer in one shot mode.
- void gptStartOneShotI (GPTDriver ∗gptp, gptcnt_t interval)
  Starts the timer in one shot mode.

- void gptStopTimer (GPTDriver ∗gptp)

  Stops the timer.
- void gptStopTimerI (GPTDriver ∗gptp)

  Stops the timer.
- void gptPolledDelay (GPTDriver ∗gptp, gptcnt_t interval)

  Starts the timer in one shot mode and waits for completion.

9.23.1 Detailed Description

GPT Driver code.

9.24 hal_gpt.h File Reference

GPT Driver macros and structures.

```
#include "hal_gpt_lld.h"
```

Macros

- #define gptChangeIntervalI(gptp, interval)

  Changes the interval of GPT peripheral.
- #define gptGetIntervalX(gptp) gpt_lld_get_interval(gptp)

  Returns the interval of GPT peripheral.
- #define gptGetCounterX(gptp) gpt_lld_get_counter(gptp)

  Returns the counter value of GPT peripheral.

Typedefs

- typedef struct GPTDriver GPTDriver

  Type of a structure representing a GPT driver.
- typedef void(∗ gptcallback_t) (GPTDriver ∗gptp)

  GPT notification callback type.

Enumerations

Functions

- void gptInit (void)

  GPT Driver initialization.
- void gptObjectInit (GPTDriver ∗gptp)

  Initializes the standard part of a `GPTDriver` structure.
- void gptStart (GPTDriver ∗gptp, const GPTConfig ∗config)

  Configures and activates the GPT peripheral.
- void gptStop (GPTDriver ∗gptp)

  Deactivates the GPT peripheral.
- void gptStartContinuous (GPTDriver ∗gptp, gptcnt_t interval)

  Starts the timer in continuous mode.
- void gptStartContinuousI (GPTDriver ∗gptp, gptcnt_t interval)

Starts the timer in continuous mode.
- void gptChangeInterval (GPTDriver ∗gptp, gptcnt_t interval)

    Changes the interval of GPT peripheral.
- void gptStartOneShot (GPTDriver ∗gptp, gptcnt_t interval)

    Starts the timer in one shot mode.
- void gptStartOneShotI (GPTDriver ∗gptp, gptcnt_t interval)

    Starts the timer in one shot mode.
- void gptStopTimer (GPTDriver ∗gptp)

    Stops the timer.
- void gptStopTimerI (GPTDriver ∗gptp)

    Stops the timer.
- void gptPolledDelay (GPTDriver ∗gptp, gptcnt_t interval)

    Starts the timer in one shot mode and waits for completion.

9.24.1 Detailed Description

GPT Driver macros and structures.

9.25 hal_gpt_lld.c File Reference

PLATFORM GPT subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void gpt_lld_init (void)

    Low level GPT driver initialization.
- void gpt_lld_start (GPTDriver ∗gptp)

    Configures and activates the GPT peripheral.
- void gpt_lld_stop (GPTDriver ∗gptp)

    Deactivates the GPT peripheral.
- void gpt_lld_start_timer (GPTDriver ∗gptp, gptcnt_t interval)

    Starts the timer in continuous mode.
- void gpt_lld_stop_timer (GPTDriver ∗gptp)

    Stops the timer.
- void gpt_lld_polled_delay (GPTDriver ∗gptp, gptcnt_t interval)

    Starts the timer in one shot mode and waits for completion.

Variables

- GPTDriver GPTD1

    GPTD1 driver identifier.

9.25.1 Detailed Description

PLATFORM GPT subsystem low level driver source.

9.26 hal_gpt_lld.h File Reference

PLATFORM GPT subsystem low level driver header.

Data Structures

- struct GPTConfig

  Driver configuration structure.
- struct GPTDriver

  Structure representing a GPT driver.

Macros

- #define gpt_lld_change_interval(gptp, interval)

  Changes the interval of GPT peripheral.

PLATFORM configuration options

- #define PLATFORM_GPT_USE_GPT1 FALSE

  GPTD1 driver enable switch.

Typedefs

- typedef uint32_t gptfreq_t

  GPT frequency type.
- typedef uint16_t gptcnt_t

  GPT counter type.

Functions

- void gpt_lld_init (void)

  Low level GPT driver initialization.
- void gpt_lld_start (GPTDriver ∗gptp)

  Configures and activates the GPT peripheral.
- void gpt_lld_stop (GPTDriver ∗gptp)

  Deactivates the GPT peripheral.
- void gpt_lld_start_timer (GPTDriver ∗gptp, gptcnt_t interval)

  Starts the timer in continuous mode.
- void gpt_lld_stop_timer (GPTDriver ∗gptp)

  Stops the timer.
- void gpt_lld_polled_delay (GPTDriver ∗gptp, gptcnt_t interval)

  Starts the timer in one shot mode and waits for completion.

9.26.1 Detailed Description

PLATFORM GPT subsystem low level driver header.

9.27 hal_i2c.c File Reference

I2C Driver code.

```
#include "hal.h"
```

Functions

- void i2cInit (void)

  I2C Driver initialization.
- void i2cObjectInit (I2CDriver ∗i2cp)

  Initializes the standard part of a `I2CDriver` structure.
- void i2cStart (I2CDriver ∗i2cp, const I2CConfig ∗config)

  Configures and activates the I2C peripheral.
- void i2cStop (I2CDriver ∗i2cp)

  Deactivates the I2C peripheral.
- i2cflags_t i2cGetErrors (I2CDriver ∗i2cp)

  Returns the errors mask associated to the previous operation.
- msg_t i2cMasterTransmitTimeout (I2CDriver ∗i2cp, i2caddr_t addr, const uint8_t ∗txbuf, size_t txbytes, uint8_t ∗rxbuf, size_t rxbytes, systime_t timeout)

  Sends data via the I2C bus.
- msg_t i2cMasterReceiveTimeout (I2CDriver ∗i2cp, i2caddr_t addr, uint8_t ∗rxbuf, size_t rxbytes, systime_t timeout)

  Receives data from the I2C bus.
- void i2cAcquireBus (I2CDriver ∗i2cp)

  Gains exclusive access to the I2C bus.
- void i2cReleaseBus (I2CDriver ∗i2cp)

  Releases exclusive access to the I2C bus.

9.27.1 Detailed Description

I2C Driver code.

9.28 hal_i2c.h File Reference

I2C Driver macros and structures.

```
#include "hal_i2c_lld.h"
```

Macros

- #define I2C_USE_MUTUAL_EXCLUSION TRUE

  Enables the mutual exclusion APIs on the I2C bus.
- #define _i2c_wakeup_isr(i2cp)

  Wakes up the waiting thread notifying no errors.
- #define _i2c_wakeup_error_isr(i2cp)

  Wakes up the waiting thread notifying errors.
- #define i2cMasterTransmit(i2cp, addr, txbuf, txbytes, rxbuf, rxbytes)

  Wrap i2cMasterTransmitTimeout function with TIME_INFINITE timeout.

- #define i2cMasterReceive(i2cp, addr, rxbuf, rxbytes) (i2cMasterReceiveTimeout(i2cp, addr, rxbuf, rxbytes, TIME_INFINITE))

  Wrap i2cMasterReceiveTimeout function with TIME_INFINITE timeout.

I2C bus error conditions

- #define I2C_NO_ERROR 0x00

  No error.
- #define I2C_BUS_ERROR 0x01

  Bus Error.
- #define I2C_ARBITRATION_LOST 0x02

  Arbitration Lost.
- #define I2C_ACK_FAILURE 0x04

  Acknowledge Failure.
- #define I2C_OVERRUN 0x08

  Overrun/Underrun.
- #define I2C_PEC_ERROR 0x10

  PEC Error in reception.
- #define I2C_TIMEOUT 0x20

  Hardware timeout.
- #define I2C_SMB_ALERT 0x40

  SMBus Alert.

Enumerations

Functions

- void i2cInit (void)

  I2C Driver initialization.
- void i2cObjectInit (I2CDriver ∗i2cp)

  Initializes the standard part of a `I2CDriver` structure.
- void i2cStart (I2CDriver ∗i2cp, const I2CConfig ∗config)

  Configures and activates the I2C peripheral.
- void i2cStop (I2CDriver ∗i2cp)

  Deactivates the I2C peripheral.
- i2cflags_t i2cGetErrors (I2CDriver ∗i2cp)

  Returns the errors mask associated to the previous operation.
- msg_t i2cMasterTransmitTimeout (I2CDriver ∗i2cp, i2caddr_t addr, const uint8_t ∗txbuf, size_t txbytes, uint8_t ∗rxbuf, size_t rxbytes, systime_t timeout)

  Sends data via the I2C bus.
- msg_t i2cMasterReceiveTimeout (I2CDriver ∗i2cp, i2caddr_t addr, uint8_t ∗rxbuf, size_t rxbytes, systime_t timeout)

  Receives data from the I2C bus.
- void i2cAcquireBus (I2CDriver ∗i2cp)

  Gains exclusive access to the I2C bus.
- void i2cReleaseBus (I2CDriver ∗i2cp)

  Releases exclusive access to the I2C bus.

9.28.1 Detailed Description

I2C Driver macros and structures.

9.29 hal_i2c_lld.c File Reference

PLATFORM I2C subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void i2c_lld_init (void)
  Low level I2C driver initialization.
- void i2c_lld_start (I2CDriver ∗i2cp)
  Configures and activates the I2C peripheral.
- void i2c_lld_stop (I2CDriver ∗i2cp)
  Deactivates the I2C peripheral.
- msg_t i2c_lld_master_receive_timeout (I2CDriver ∗i2cp, i2caddr_t addr, uint8_t ∗rxbuf, size_t rxbytes, systime_t timeout)
  Receives data via the I2C bus as master.
- msg_t i2c_lld_master_transmit_timeout (I2CDriver ∗i2cp, i2caddr_t addr, const uint8_t ∗txbuf, size_t txbytes, uint8_t ∗rxbuf, size_t rxbytes, systime_t timeout)
  Transmits data via the I2C bus as master.

Variables

- I2CDriver I2CD1
  I2C1 driver identifier.

9.29.1 Detailed Description

PLATFORM I2C subsystem low level driver source.

9.30 hal_i2c_lld.h File Reference

PLATFORM I2C subsystem low level driver header.

Data Structures

- struct I2CConfig
  Type of I2C driver configuration structure.
- struct I2CDriver
  Structure representing an I2C driver.

Macros

- #define i2c_lld_get_errors(i2cp) ((i2cp)->errors)
  Get errors from I2C driver.

PLATFORM configuration options

- #define PLATFORM_I2C_USE_I2C1 FALSE
  I2C1 driver enable switch.

Typedefs

- typedef uint16_t i2caddr_t

    Type representing an I2C address.
- typedef uint32_t i2cflags_t

    Type of I2C Driver condition flags.
- typedef struct I2CDriver I2CDriver

    Type of a structure representing an I2C driver.

Functions

- void i2c_lld_init (void)

    Low level I2C driver initialization.
- void i2c_lld_start (I2CDriver ∗i2cp)

    Configures and activates the I2C peripheral.
- void i2c_lld_stop (I2CDriver ∗i2cp)

    Deactivates the I2C peripheral.
- msg_t i2c_lld_master_transmit_timeout (I2CDriver ∗i2cp, i2caddr_t addr, const uint8_t ∗txbuf, size_t txbytes, uint8_t ∗rxbuf, size_t rxbytes, systime_t timeout)

    Transmits data via the I2C bus as master.
- msg_t i2c_lld_master_receive_timeout (I2CDriver ∗i2cp, i2caddr_t addr, uint8_t ∗rxbuf, size_t rxbytes, systime_t timeout)

    Receives data via the I2C bus as master.

9.30.1 Detailed Description

PLATFORM I2C subsystem low level driver header.

9.31 hal_i2s.c File Reference

I2S Driver code.

```
#include "hal.h"
```

Functions

- void i2sInit (void)

    I2S Driver initialization.
- void i2sObjectInit (I2SDriver ∗i2sp)

    Initializes the standard part of a `I2SDriver` structure.
- void i2sStart (I2SDriver ∗i2sp, const I2SConfig ∗config)

    Configures and activates the I2S peripheral.
- void i2sStop (I2SDriver ∗i2sp)

    Deactivates the I2S peripheral.
- void i2sStartExchange (I2SDriver ∗i2sp)

    Starts a I2S data exchange.
- void i2sStopExchange (I2SDriver ∗i2sp)

    Stops the ongoing data exchange.

9.31.1 Detailed Description

I2S Driver code.

9.32 hal_i2s.h File Reference

I2S Driver macros and structures.

```
#include "hal_i2s_lld.h"
```

Macros

I2S modes

- #define **I2S_MODE_SLAVE** 0
- #define **I2S_MODE_MASTER** 1

Macro Functions

- #define i2sStartExchangeI(i2sp)
  Starts a I2S data exchange.
- #define i2sStopExchangeI(i2sp)
  Stops the ongoing data exchange.
- #define _i2s_isr_half_code(i2sp)
  Common ISR code, half buffer event.
- #define _i2s_isr_full_code(i2sp)
  Common ISR code.

Enumerations

Functions

- void i2sInit (void)
  I2S Driver initialization.
- void i2sObjectInit (I2SDriver ∗i2sp)
  Initializes the standard part of a `I2SDriver` structure.
- void i2sStart (I2SDriver ∗i2sp, const I2SConfig ∗config)
  Configures and activates the I2S peripheral.
- void i2sStop (I2SDriver ∗i2sp)
  Deactivates the I2S peripheral.
- void i2sStartExchange (I2SDriver ∗i2sp)
  Starts a I2S data exchange.
- void i2sStopExchange (I2SDriver ∗i2sp)
  Stops the ongoing data exchange.

9.32.1 Detailed Description

I2S Driver macros and structures.

9.33 hal_i2s_lld.c File Reference

PLATFORM I2S subsystem low level driver source.

`#include "hal.h"`

Functions

- void i2s_lld_init (void)

  Low level I2S driver initialization.
- void i2s_lld_start (I2SDriver ∗i2sp)

  Configures and activates the I2S peripheral.
- void i2s_lld_stop (I2SDriver ∗i2sp)

  Deactivates the I2S peripheral.
- void i2s_lld_start_exchange (I2SDriver ∗i2sp)

  Starts a I2S data exchange.
- void i2s_lld_stop_exchange (I2SDriver ∗i2sp)

  Stops the ongoing data exchange.

Variables

- I2SDriver I2SD1

  I2S2 driver identifier.

9.33.1 Detailed Description

PLATFORM I2S subsystem low level driver source.

9.34 hal_i2s_lld.h File Reference

PLATFORM I2S subsystem low level driver header.

Data Structures

- struct I2SConfig

  Driver configuration structure.
- struct I2SDriver

  Structure representing an I2S driver.

Macros

PLATFORM configuration options

- #define PLATFORM_I2S_USE_I2S1 FALSE

  I2SD1 driver enable switch.

Typedefs

- typedef struct I2SDriver I2SDriver

    Type of a structure representing an I2S driver.
- typedef void(∗ i2scallback_t) (I2SDriver ∗i2sp, size_t offset, size_t n)

    I2S notification callback type.

Functions

- void i2s_lld_init (void)

    Low level I2S driver initialization.
- void i2s_lld_start (I2SDriver ∗i2sp)

    Configures and activates the I2S peripheral.
- void i2s_lld_stop (I2SDriver ∗i2sp)

    Deactivates the I2S peripheral.
- void i2s_lld_start_exchange (I2SDriver ∗i2sp)

    Starts a I2S data exchange.
- void i2s_lld_stop_exchange (I2SDriver ∗i2sp)

    Stops the ongoing data exchange.

9.34.1 Detailed Description

PLATFORM I2S subsystem low level driver header.

9.35 hal_icu.c File Reference

ICU Driver code.

```
#include "hal.h"
```

Functions

- void icuInit (void)

    ICU Driver initialization.
- void icuObjectInit (ICUDriver ∗icup)

    Initializes the standard part of a `ICUDriver` structure.
- void icuStart (ICUDriver ∗icup, const ICUConfig ∗config)

    Configures and activates the ICU peripheral.
- void icuStop (ICUDriver ∗icup)

    Deactivates the ICU peripheral.
- void icuStartCapture (ICUDriver ∗icup)

    Starts the input capture.
- bool icuWaitCapture (ICUDriver ∗icup)

    Waits for a completed capture.
- void icuStopCapture (ICUDriver ∗icup)

    Stops the input capture.
- void icuEnableNotifications (ICUDriver ∗icup)

    Enables notifications.
- void icuDisableNotifications (ICUDriver ∗icup)

    Disables notifications.

9.35.1 Detailed Description

ICU Driver code.

9.36 hal_icu.h File Reference

ICU Driver macros and structures.

`#include "hal_icu_lld.h"`

Macros

Macro Functions

- #define icuStartCaptureI(icup)
  Starts the input capture.
- #define icuStopCaptureI(icup)
  Stops the input capture.
- #define icuEnableNotificationsI(icup) icu_lld_enable_notifications(icup)
  Enables notifications.
- #define icuDisableNotificationsI(icup) icu_lld_disable_notifications(icup)
  Disables notifications.
- #define icuAreNotificationsEnabledX(icup) icu_lld_are_notifications_enabled(icup)
  Check on notifications status.
- #define icuGetWidthX(icup) icu_lld_get_width(icup)
  Returns the width of the latest pulse.
- #define icuGetPeriodX(icup) icu_lld_get_period(icup)
  Returns the width of the latest cycle.

Low level driver helper macros

- #define _icu_isr_invoke_width_cb(icup)
  Common ISR code, ICU width event.
- #define _icu_isr_invoke_period_cb(icup)
  Common ISR code, ICU period event.
- #define _icu_isr_invoke_overflow_cb(icup)
  Common ISR code, ICU timer overflow event.

Typedefs

- typedef struct ICUDriver ICUDriver
  Type of a structure representing an ICU driver.
- typedef void(∗ icucallback_t) (ICUDriver ∗icup)
  ICU notification callback type.

Enumerations

Functions

- void icuInit (void)
  ICU Driver initialization.
- void icuObjectInit (ICUDriver ∗icup)
  Initializes the standard part of a `ICUDriver` structure.

- void icuStart (ICUDriver ∗icup, const ICUConfig ∗config)

  Configures and activates the ICU peripheral.
- void icuStop (ICUDriver ∗icup)

  Deactivates the ICU peripheral.
- void icuStartCapture (ICUDriver ∗icup)

  Starts the input capture.
- bool icuWaitCapture (ICUDriver ∗icup)

  Waits for a completed capture.
- void icuStopCapture (ICUDriver ∗icup)

  Stops the input capture.
- void icuEnableNotifications (ICUDriver ∗icup)

  Enables notifications.
- void icuDisableNotifications (ICUDriver ∗icup)

  Disables notifications.

9.36.1 Detailed Description

ICU Driver macros and structures.

9.37 hal_icu_lld.c File Reference

PLATFORM ADC subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void icu_lld_init (void)

  Low level ICU driver initialization.
- void icu_lld_start (ICUDriver ∗icup)

  Configures and activates the ICU peripheral.
- void icu_lld_stop (ICUDriver ∗icup)

  Deactivates the ICU peripheral.
- void icu_lld_start_capture (ICUDriver ∗icup)

  Starts the input capture.
- bool icu_lld_wait_capture (ICUDriver ∗icup)

  Waits for a completed capture.
- void icu_lld_stop_capture (ICUDriver ∗icup)

  Stops the input capture.
- void icu_lld_enable_notifications (ICUDriver ∗icup)

  Enables notifications.
- void icu_lld_disable_notifications (ICUDriver ∗icup)

  Disables notifications.

Variables

- ICUDriver ICUD1

  ICUD1 driver identifier.

9.37.1 Detailed Description

PLATFORM ADC subsystem low level driver source.

9.38 hal_icu_lld.h File Reference

PLATFORM ICU subsystem low level driver header.

Data Structures

- struct ICUConfig

  Driver configuration structure.
- struct ICUDriver

  Structure representing an ICU driver.

Macros

- #define icu_lld_get_width(icup) 0

  Returns the width of the latest pulse.
- #define icu_lld_get_period(icup) 0

  Returns the width of the latest cycle.
- #define icu_lld_are_notifications_enabled(icup) false

  Check on notifications status.

PLATFORM configuration options

- #define PLATFORM_ICU_USE_ICU1 FALSE

  ICUD1 driver enable switch.

Typedefs

- typedef uint32_t icufreq_t

  ICU frequency type.
- typedef uint32_t icucnt_t

  ICU counter type.

Enumerations

Functions

- void icu_lld_init (void)

  Low level ICU driver initialization.
- void icu_lld_start (ICUDriver ∗icup)

  Configures and activates the ICU peripheral.
- void icu_lld_stop (ICUDriver ∗icup)

  Deactivates the ICU peripheral.
- void icu_lld_start_capture (ICUDriver ∗icup)

  Starts the input capture.
- bool icu_lld_wait_capture (ICUDriver ∗icup)

  Waits for a completed capture.

- void icu_lld_stop_capture (ICUDriver ∗icup)

  Stops the input capture.
- void icu_lld_enable_notifications (ICUDriver ∗icup)

  Enables notifications.
- void icu_lld_disable_notifications (ICUDriver ∗icup)

  Disables notifications.

9.38.1 Detailed Description

PLATFORM ICU subsystem low level driver header.

9.39 hal_ioblock.h File Reference

I/O block devices access.

Data Structures

- struct BlockDeviceInfo

  Block device info.
- struct BaseBlockDeviceVMT

  `BaseBlockDevice` *virtual methods table.*
- struct BaseBlockDevice

  Base block device class.

Macros

- #define _base_block_device_methods

  `BaseBlockDevice` *specific methods.*
- #define _base_block_device_data

  `BaseBlockDevice` *specific data.*

Macro Functions (BaseBlockDevice)

- #define blkGetDriverState(ip) ((ip)->state)

  Returns the driver state.
- #define blkIsTransferring(ip)

  Determines if the device is transferring data.
- #define blkIsInserted(ip) ((ip)->vmt->is_inserted(ip))

  Returns the media insertion status.
- #define blkIsWriteProtected(ip) ((ip)->vmt->is_protected(ip))

  Returns the media write protection status.
- #define blkConnect(ip) ((ip)->vmt->connect(ip))

  Performs the initialization procedure on the block device.
- #define blkDisconnect(ip) ((ip)->vmt->disconnect(ip))

  Terminates operations on the block device.
- #define blkRead(ip, startblk, buf, n) ((ip)->vmt->read(ip, startblk, buf, n))

  Reads one or more blocks.
- #define blkWrite(ip, startblk, buf, n) ((ip)->vmt->write(ip, startblk, buf, n))

  Writes one or more blocks.
- #define blkSync(ip) ((ip)->vmt->sync(ip))

  Ensures write synchronization.
- #define blkGetInfo(ip, bdip) ((ip)->vmt->get_info(ip, bdip))

  Returns a media information structure.

Enumerations

9.39.1 Detailed Description

I/O block devices access.

This header defines an abstract interface useful to access generic I/O block devices in a standardized way.

9.40 hal_lld.c File Reference

PLATFORM HAL subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void hal_lld_init (void)

  Low level HAL driver initialization.

9.40.1 Detailed Description

PLATFORM HAL subsystem low level driver source.

9.41 hal_lld.h File Reference

PLATFORM HAL subsystem low level driver header.

Macros

Platform identification macros

- #define **PLATFORM_NAME** "templates"

Functions

- void hal_lld_init (void)

  Low level HAL driver initialization.

9.41.1 Detailed Description

PLATFORM HAL subsystem low level driver header.

9.42 hal_mac.c File Reference

MAC Driver code.

```
#include "hal.h"
```

Functions

- void macInit (void)
  MAC Driver initialization.
- void macObjectInit (MACDriver ∗macp)
  Initialize the standard part of a `MACDriver` structure.
- void macStart (MACDriver ∗macp, const MACConfig ∗config)
  Configures and activates the MAC peripheral.
- void macStop (MACDriver ∗macp)
  Deactivates the MAC peripheral.
- msg_t macWaitTransmitDescriptor (MACDriver ∗macp, MACTransmitDescriptor ∗tdp, systime_t timeout)
  Allocates a transmission descriptor.
- void macReleaseTransmitDescriptor (MACTransmitDescriptor ∗tdp)
  Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.
- msg_t macWaitReceiveDescriptor (MACDriver ∗macp, MACReceiveDescriptor ∗rdp, systime_t timeout)
  Waits for a received frame.
- void macReleaseReceiveDescriptor (MACReceiveDescriptor ∗rdp)
  Releases a receive descriptor.
- bool macPollLinkStatus (MACDriver ∗macp)
  Updates and returns the link status.

9.42.1 Detailed Description

MAC Driver code.

9.43 hal_mac.h File Reference

MAC Driver macros and structures.

```
#include "hal_mac_lld.h"
```

Macros

MAC configuration options

- #define MAC_USE_ZERO_COPY FALSE
  Enables an event sources for incoming packets.
- #define MAC_USE_EVENTS TRUE
  Enables an event sources for incoming packets.

Macro Functions

- #define macGetReceiveEventSource(macp) (&(macp)->rdevent)
  Returns the received frames event source.
- #define macWriteTransmitDescriptor(tdp, buf, size) mac_lld_write_transmit_descriptor(tdp, buf, size)
  Writes to a transmit descriptor's stream.
- #define macReadReceiveDescriptor(rdp, buf, size) mac_lld_read_receive_descriptor(rdp, buf, size)
  Reads from a receive descriptor's stream.
- #define macGetNextTransmitBuffer(tdp, size, sizep) mac_lld_get_next_transmit_buffer(tdp, size, sizep)
  Returns a pointer to the next transmit buffer in the descriptor chain.
- #define macGetNextReceiveBuffer(rdp, sizep) mac_lld_get_next_receive_buffer(rdp, sizep)
  Returns a pointer to the next receive buffer in the descriptor chain.

Typedefs

- typedef struct MACDriver MACDriver

  Type of a structure representing a MAC driver.

Enumerations

Functions

- void macInit (void)

  MAC Driver initialization.
- void macObjectInit (MACDriver ∗macp)

  Initialize the standard part of a `MACDriver` structure.
- void macStart (MACDriver ∗macp, const MACConfig ∗config)

  Configures and activates the MAC peripheral.
- void macStop (MACDriver ∗macp)

  Deactivates the MAC peripheral.
- msg_t macWaitTransmitDescriptor (MACDriver ∗macp, MACTransmitDescriptor ∗tdp, systime_t timeout)

  Allocates a transmission descriptor.
- void macReleaseTransmitDescriptor (MACTransmitDescriptor ∗tdp)

  Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.
- msg_t macWaitReceiveDescriptor (MACDriver ∗macp, MACReceiveDescriptor ∗rdp, systime_t timeout)

  Waits for a received frame.
- void macReleaseReceiveDescriptor (MACReceiveDescriptor ∗rdp)

  Releases a receive descriptor.
- bool macPollLinkStatus (MACDriver ∗macp)

  Updates and returns the link status.

9.43.1 Detailed Description

MAC Driver macros and structures.

9.44 hal_mac_lld.c File Reference

PLATFORM MAC subsystem low level driver source.

```
#include <string.h>
#include "hal.h"
#include "hal_mii.h"
```

Functions

- void mac_lld_init (void)

  Low level MAC initialization.
- void mac_lld_start (MACDriver ∗macp)

  Configures and activates the MAC peripheral.
- void mac_lld_stop (MACDriver ∗macp)

  Deactivates the MAC peripheral.
- msg_t mac_lld_get_transmit_descriptor (MACDriver ∗macp, MACTransmitDescriptor ∗tdp)

Returns a transmission descriptor.
- void mac_lld_release_transmit_descriptor (MACTransmitDescriptor ∗tdp)

    Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.
- msg_t mac_lld_get_receive_descriptor (MACDriver ∗macp, MACReceiveDescriptor ∗rdp)

    Returns a receive descriptor.
- void mac_lld_release_receive_descriptor (MACReceiveDescriptor ∗rdp)

    Releases a receive descriptor.
- bool mac_lld_poll_link_status (MACDriver ∗macp)

    Updates and returns the link status.
- size_t mac_lld_write_transmit_descriptor (MACTransmitDescriptor ∗tdp, uint8_t ∗buf, size_t size)

    Writes to a transmit descriptor's stream.
- size_t mac_lld_read_receive_descriptor (MACReceiveDescriptor ∗rdp, uint8_t ∗buf, size_t size)

    Reads from a receive descriptor's stream.
- uint8_t ∗ mac_lld_get_next_transmit_buffer (MACTransmitDescriptor ∗tdp, size_t size, size_t ∗sizep)

    Returns a pointer to the next transmit buffer in the descriptor chain.
- const uint8_t ∗ mac_lld_get_next_receive_buffer (MACReceiveDescriptor ∗rdp, size_t ∗sizep)

    Returns a pointer to the next receive buffer in the descriptor chain.

Variables

- MACDriver ETHD1

    MAC1 driver identifier.

9.44.1 Detailed Description

PLATFORM MAC subsystem low level driver source.

9.45 hal_mac_lld.h File Reference

PLATFORM MAC subsystem low level driver header.

Data Structures

- struct MACConfig

    Driver configuration structure.
- struct MACDriver

    Structure representing a MAC driver.
- struct MACTransmitDescriptor

    Structure representing a transmit descriptor.
- struct MACReceiveDescriptor

    Structure representing a receive descriptor.

Macros

- #define MAC_SUPPORTS_ZERO_COPY TRUE

    This implementation supports the zero-copy mode API.

PLATFORM configuration options

- #define PLATFORM_MAC_USE_MAC1 FALSE

    MAC driver enable switch.

Functions

- void mac_lld_init (void)

  Low level MAC initialization.
- void mac_lld_start (MACDriver ∗macp)

  Configures and activates the MAC peripheral.
- void mac_lld_stop (MACDriver ∗macp)

  Deactivates the MAC peripheral.
- msg_t mac_lld_get_transmit_descriptor (MACDriver ∗macp, MACTransmitDescriptor ∗tdp)

  Returns a transmission descriptor.
- void mac_lld_release_transmit_descriptor (MACTransmitDescriptor ∗tdp)

  Releases a transmit descriptor and starts the transmission of the enqueued data as a single frame.
- msg_t mac_lld_get_receive_descriptor (MACDriver ∗macp, MACReceiveDescriptor ∗rdp)

  Returns a receive descriptor.
- void mac_lld_release_receive_descriptor (MACReceiveDescriptor ∗rdp)

  Releases a receive descriptor.
- bool mac_lld_poll_link_status (MACDriver ∗macp)

  Updates and returns the link status.
- size_t mac_lld_write_transmit_descriptor (MACTransmitDescriptor ∗tdp, uint8_t ∗buf, size_t size)

  Writes to a transmit descriptor's stream.
- size_t mac_lld_read_receive_descriptor (MACReceiveDescriptor ∗rdp, uint8_t ∗buf, size_t size)

  Reads from a receive descriptor's stream.
- uint8_t ∗ mac_lld_get_next_transmit_buffer (MACTransmitDescriptor ∗tdp, size_t size, size_t ∗sizep)

  Returns a pointer to the next transmit buffer in the descriptor chain.
- const uint8_t ∗ mac_lld_get_next_receive_buffer (MACReceiveDescriptor ∗rdp, size_t ∗sizep)

  Returns a pointer to the next receive buffer in the descriptor chain.

9.45.1 Detailed Description

PLATFORM MAC subsystem low level driver header.

9.46 hal_mii.h File Reference

MII macros and structures.

Macros

Generic MII registers

- #define MII_BMCR 0x00
- #define MII_BMSR 0x01
- #define MII_PHYSID1 0x02
- #define MII_PHYSID2 0x03
- #define MII_ADVERTISE 0x04
- #define MII_LPA 0x05
- #define MII_EXPANSION 0x06
- #define MII_ANNPTR 0x07
- #define MII_CTRL1000 0x09
- #define MII_STAT1000 0x0a
- #define MII_ESTATUS 0x0f
- #define MII_PHYSTS 0x10
- #define MII_MICR 0x11
- #define MII_DCOUNTER 0x12

- #define MII_FCSCOUNTER 0x13
- #define MII_NWAYTEST 0x14
- #define MII_RERRCOUNTER 0x15
- #define MII_SREVISION 0x16
- #define MII_RESV1 0x17
- #define MII_LBRERROR 0x18
- #define MII_PHYADDR 0x19
- #define MII_RESV2 0x1a
- #define MII_TPISTATUS 0x1b
- #define MII_NCONFIG 0x1c

Basic mode control register

- #define BMCR_RESV 0x007f
- #define BMCR_CTST 0x0080
- #define BMCR_FULLDPLX 0x0100
- #define BMCR_ANRESTART 0x0200
- #define BMCR_ISOLATE 0x0400
- #define BMCR_PDOWN 0x0800
- #define BMCR_ANENABLE 0x1000
- #define BMCR_SPEED100 0x2000
- #define BMCR_LOOPBACK 0x4000
- #define BMCR_RESET 0x8000

Basic mode status register

- #define BMSR_ERCAP 0x0001
- #define BMSR_JCD 0x0002
- #define BMSR_LSTATUS 0x0004
- #define BMSR_ANEGCAPABLE 0x0008
- #define BMSR_RFAULT 0x0010
- #define BMSR_ANEGCOMPLETE 0x0020
- #define BMSR_MFPRESUPPCAP 0x0040
- #define BMSR_RESV 0x0780
- #define BMSR_10HALF 0x0800
- #define BMSR_10FULL 0x1000
- #define BMSR_100HALF 0x2000
- #define BMSR_100FULL 0x4000
- #define BMSR_100BASE4 0x8000

Advertisement control register

- #define ADVERTISE_SLCT 0x001f
- #define ADVERTISE_CSMA 0x0001
- #define ADVERTISE_10HALF 0x0020
- #define ADVERTISE_10FULL 0x0040
- #define ADVERTISE_100HALF 0x0080
- #define ADVERTISE_100FULL 0x0100
- #define ADVERTISE_100BASE4 0x0200
- #define ADVERTISE_PAUSE_CAP 0x0400
- #define ADVERTISE_PAUSE_ASYM 0x0800
- #define ADVERTISE_RESV 0x1000
- #define ADVERTISE_RFAULT 0x2000
- #define ADVERTISE_LPACK 0x4000
- #define ADVERTISE_NPAGE 0x8000
- #define **ADVERTISE_FULL**
- #define **ADVERTISE_ALL**

Link partner ability register

- #define LPA_SLCT 0x001f
- #define LPA_10HALF 0x0020
- #define LPA_10FULL 0x0040

- #define LPA_100HALF 0x0080
- #define LPA_100FULL 0x0100
- #define LPA_100BASE4 0x0200
- #define LPA_PAUSE_CAP 0x0400
- #define LPA_PAUSE_ASYM 0x0800
- #define LPA_RESV 0x1000
- #define LPA_RFAULT 0x2000
- #define LPA_LPACK 0x4000
- #define LPA_NPAGE 0x8000
- #define **LPA_DUPLEX** (LPA_10FULL | LPA_100FULL)
- #define **LPA_100** (LPA_100FULL | LPA_100HALF | LPA_100BASE4)

Expansion register for auto-negotiation

- #define EXPANSION_NWAY 0x0001
- #define EXPANSION_LCWP 0x0002
- #define EXPANSION_ENABLENPAGE 0x0004
- #define EXPANSION_NPCAPABLE 0x0008
- #define EXPANSION_MFAULTS 0x0010
- #define EXPANSION_RESV 0xffe0

N-way test register

- #define NWAYTEST_RESV1 0x00ff
- #define NWAYTEST_LOOPBACK 0x0100
- #define NWAYTEST_RESV2 0xfe00

PHY identifiers

- #define **MII_DM9161_ID** 0x0181b8a0
- #define **MII_AM79C875_ID** 0x00225540
- #define **MII_KS8721_ID** 0x00221610
- #define **MII_STE101P_ID** 0x00061C50
- #define **MII_DP83848I_ID** 0x20005C90
- #define **MII_LAN8710A_ID** 0x0007C0F1
- #define **MII_LAN8720_ID** 0x0007C0F0
- #define **MII_LAN8742A_ID** 0x0007C130

9.46.1 Detailed Description

MII macros and structures.

9.47 hal_mmc_spi.c File Reference

MMC over SPI driver code.

```
#include <string.h>
#include "hal.h"
```

Functions

- static uint8_t crc7 (uint8_t crc, const uint8_t ∗buffer, size_t len)

  Calculate the MMC standard CRC-7 based on a lookup table.
- static void wait (MMCDriver ∗mmcp)

  Waits an idle condition.
- static void send_hdr (MMCDriver ∗mmcp, uint8_t cmd, uint32_t arg)

Sends a command header.

- static uint8_t recvr1 (MMCDriver ∗mmcp)

  Receives a single byte response.
- static uint8_t recvr3 (MMCDriver ∗mmcp, uint8_t ∗buffer)

  Receives a three byte response.
- static uint8_t send_command_R1 (MMCDriver ∗mmcp, uint8_t cmd, uint32_t arg)

  Sends a command an returns a single byte response.
- static uint8_t send_command_R3 (MMCDriver ∗mmcp, uint8_t cmd, uint32_t arg, uint8_t ∗response)

  Sends a command which returns a five bytes response (R3).
- static bool read_CxD (MMCDriver ∗mmcp, uint8_t cmd, uint32_t cxd[4])

  Reads the CSD.
- static void sync (MMCDriver ∗mmcp)

  Waits that the card reaches an idle state.
- void mmcInit (void)

  MMC over SPI driver initialization.
- void mmcObjectInit (MMCDriver ∗mmcp)

  Initializes an instance.
- void mmcStart (MMCDriver ∗mmcp, const MMCConfig ∗config)

  Configures and activates the MMC peripheral.
- void mmcStop (MMCDriver ∗mmcp)

  Disables the MMC peripheral.
- bool mmcConnect (MMCDriver ∗mmcp)

  Performs the initialization procedure on the inserted card.
- bool mmcDisconnect (MMCDriver ∗mmcp)

  Brings the driver in a state safe for card removal.
- bool mmcStartSequentialRead (MMCDriver ∗mmcp, uint32_t startblk)

  Starts a sequential read.
- bool mmcSequentialRead (MMCDriver ∗mmcp, uint8_t ∗buffer)

  Reads a block within a sequential read operation.
- bool mmcStopSequentialRead (MMCDriver ∗mmcp)

  Stops a sequential read gracefully.
- bool mmcStartSequentialWrite (MMCDriver ∗mmcp, uint32_t startblk)

  Starts a sequential write.
- bool mmcSequentialWrite (MMCDriver ∗mmcp, const uint8_t ∗buffer)

  Writes a block within a sequential write operation.
- bool mmcStopSequentialWrite (MMCDriver ∗mmcp)

  Stops a sequential write gracefully.
- bool mmcSync (MMCDriver ∗mmcp)

  Waits for card idle condition.
- bool mmcGetInfo (MMCDriver ∗mmcp, BlockDeviceInfo ∗bdip)

  Returns the media info.
- bool mmcErase (MMCDriver ∗mmcp, uint32_t startblk, uint32_t endblk)

  Erases blocks.

Variables

- static const struct MMCDriverVMT mmc_vmt

  Virtual methods table.
- static const uint8_t crc7_lookup_table [256]

  Lookup table for CRC-7 (based on polynomial x^ 7 + x^ 3 + 1).

9.47.1 Detailed Description

MMC over SPI driver code.

9.48 hal_mmc_spi.h File Reference

MMC over SPI driver header.

Data Structures

- struct MMCConfig

    MMC/SD over SPI driver configuration structure.
- struct MMCDriverVMT

    `MMCDriver` virtual methods table.
- struct MMCDriver

    Structure representing a MMC/SD over SPI driver.

Macros

- #define _mmc_driver_methods _mmcsd_block_device_methods

    `MMCDriver` specific methods.

MMC_SPI configuration options

- #define MMC_NICE_WAITING TRUE

    Delays insertions.

Macro Functions

- #define mmcIsCardInserted(mmcp) mmc_lld_is_card_inserted(mmcp)

    Returns the card insertion status.
- #define mmcIsWriteProtected(mmcp) mmc_lld_is_write_protected(mmcp)

    Returns the write protect status.

Functions

- void mmcInit (void)

    MMC over SPI driver initialization.
- void mmcObjectInit (MMCDriver ∗mmcp)

    Initializes an instance.
- void mmcStart (MMCDriver ∗mmcp, const MMCConfig ∗config)

    Configures and activates the MMC peripheral.
- void mmcStop (MMCDriver ∗mmcp)

    Disables the MMC peripheral.
- bool mmcConnect (MMCDriver ∗mmcp)

    Performs the initialization procedure on the inserted card.
- bool mmcDisconnect (MMCDriver ∗mmcp)

    Brings the driver in a state safe for card removal.
- bool mmcStartSequentialRead (MMCDriver ∗mmcp, uint32_t startblk)

    Starts a sequential read.
- bool mmcSequentialRead (MMCDriver ∗mmcp, uint8_t ∗buffer)

Reads a block within a sequential read operation.
- bool mmcStopSequentialRead (MMCDriver ∗mmcp)

  Stops a sequential read gracefully.
- bool mmcStartSequentialWrite (MMCDriver ∗mmcp, uint32_t startblk)

  Starts a sequential write.
- bool mmcSequentialWrite (MMCDriver ∗mmcp, const uint8_t ∗buffer)

  Writes a block within a sequential write operation.
- bool mmcStopSequentialWrite (MMCDriver ∗mmcp)

  Stops a sequential write gracefully.
- bool mmcSync (MMCDriver ∗mmcp)

  Waits for card idle condition.
- bool mmcGetInfo (MMCDriver ∗mmcp, BlockDeviceInfo ∗bdip)

  Returns the media info.
- bool mmcErase (MMCDriver ∗mmcp, uint32_t startblk, uint32_t endblk)

  Erases blocks.

9.48.1 Detailed Description

MMC over SPI driver header.

9.49 hal_mmcsd.c File Reference

MMC/SD cards common code.

```
#include "hal.h"
```

Functions

- uint32_t _mmcsd_get_slice (const uint32_t ∗data, uint32_t end, uint32_t start)

  Gets a bit field from a words array.
- uint32_t _mmcsd_get_capacity (const uint32_t ∗csd)

  Extract card capacity from a CSD.
- uint32_t _mmcsd_get_capacity_ext (const uint8_t ∗ext_csd)

  Extract MMC card capacity from EXT_CSD.
- void _mmcsd_unpack_sdc_cid (const MMCSDBlockDevice ∗sdcp, unpacked_sdc_cid_t ∗cidsdc)

  Unpacks SDC CID array in structure.
- void _mmcsd_unpack_mmc_cid (const MMCSDBlockDevice ∗sdcp, unpacked_mmc_cid_t ∗cidmmc)

  Unpacks MMC CID array in structure.
- void _mmcsd_unpack_csd_mmc (const MMCSDBlockDevice ∗sdcp, unpacked_mmc_csd_t ∗csdmmc)

  Unpacks MMC CSD array in structure.
- void _mmcsd_unpack_csd_v10 (const MMCSDBlockDevice ∗sdcp, unpacked_sdc_csd_10_t ∗csd10)

  Unpacks SDC CSD v1.0 array in structure.
- void _mmcsd_unpack_csd_v20 (const MMCSDBlockDevice ∗sdcp, unpacked_sdc_csd_20_t ∗csd20)

  Unpacks SDC CSD v2.0 array in structure.

9.49.1 Detailed Description

MMC/SD cards common code.

9.50 hal_mmcsd.h File Reference

MMC/SD cards common header.

Data Structures

- struct MMCSDBlockDeviceVMT

  MMCSDBlockDevice virtual methods table.
- struct MMCSDBlockDevice

  MCC/SD block device class.
- struct unpacked_sdc_cid_t

  Unpacked CID register from SDC.
- struct unpacked_mmc_cid_t

  Unpacked CID register from MMC.
- struct unpacked_sdc_csd_10_t

  Unpacked CSD v1.0 register from SDC.
- struct unpacked_sdc_csd_20_t

  Unpacked CSD v2.0 register from SDC.
- struct unpacked_mmc_csd_t

  Unpacked CSD register from MMC.

Macros

- #define MMCSD_BLOCK_SIZE 512U

  Fixed block size for MMC/SD block devices.
- #define MMCSD_R1_ERROR_MASK 0xFDFFE008U

  Mask of error bits in R1 responses.
- #define MMCSD_CMD8_PATTERN 0x000001AAU

  Fixed pattern for CMD8.
- #define _mmcsd_block_device_methods _base_block_device_methods

  MMCSDBlockDevice specific methods.
- #define _mmcsd_block_device_data

  MMCSDBlockDevice specific data.

SD/MMC status conditions

- #define **MMCSD_STS_IDLE** 0U
- #define **MMCSD_STS_READY** 1U
- #define **MMCSD_STS_IDENT** 2U
- #define **MMCSD_STS_STBY** 3U
- #define **MMCSD_STS_TRAN** 4U
- #define **MMCSD_STS_DATA** 5U
- #define **MMCSD_STS_RCV** 6U
- #define **MMCSD_STS_PRG** 7U
- #define **MMCSD_STS_DIS** 8U

SD/MMC commands

- #define **MMCSD_CMD_GO_IDLE_STATE** 0U
- #define **MMCSD_CMD_INIT** 1U
- #define **MMCSD_CMD_ALL_SEND_CID** 2U
- #define **MMCSD_CMD_SEND_RELATIVE_ADDR** 3U
- #define **MMCSD_CMD_SET_BUS_WIDTH** 6U
- #define **MMCSD_CMD_SWITCH** MMCSD_CMD_SET_BUS_WIDTH

- #define **MMCSD_CMD_SEL_DESEL_CARD** 7U
- #define **MMCSD_CMD_SEND_IF_COND** 8U
- #define **MMCSD_CMD_SEND_EXT_CSD** MMCSD_CMD_SEND_IF_COND
- #define **MMCSD_CMD_SEND_CSD** 9U
- #define **MMCSD_CMD_SEND_CID** 10U
- #define **MMCSD_CMD_STOP_TRANSMISSION** 12U
- #define **MMCSD_CMD_SEND_STATUS** 13U
- #define **MMCSD_CMD_SET_BLOCKLEN** 16U
- #define **MMCSD_CMD_READ_SINGLE_BLOCK** 17U
- #define **MMCSD_CMD_READ_MULTIPLE_BLOCK** 18U
- #define **MMCSD_CMD_SET_BLOCK_COUNT** 23U
- #define **MMCSD_CMD_WRITE_BLOCK** 24U
- #define **MMCSD_CMD_WRITE_MULTIPLE_BLOCK** 25U
- #define **MMCSD_CMD_ERASE_RW_BLK_START** 32U
- #define **MMCSD_CMD_ERASE_RW_BLK_END** 33U
- #define **MMCSD_CMD_ERASE** 38U
- #define **MMCSD_CMD_APP_OP_COND** 41U
- #define **MMCSD_CMD_LOCK_UNLOCK** 42U
- #define **MMCSD_CMD_APP_CMD** 55U
- #define **MMCSD_CMD_READ_OCR** 58U

CSD record offsets

- #define MMCSD_CSD_MMC_CSD_STRUCTURE_SLICE 127U,126U

  Slice position of values in CSD register.
- #define **MMCSD_CSD_MMC_SPEC_VERS_SLICE** 125U,122U
- #define **MMCSD_CSD_MMC_TAAC_SLICE** 119U,112U
- #define **MMCSD_CSD_MMC_NSAC_SLICE** 111U,104U
- #define **MMCSD_CSD_MMC_TRAN_SPEED_SLICE** 103U,96U
- #define **MMCSD_CSD_MMC_CCC_SLICE** 95U,84U
- #define **MMCSD_CSD_MMC_READ_BL_LEN_SLICE** 83U,80U
- #define **MMCSD_CSD_MMC_READ_BL_PARTIAL_SLICE** 79U,79U
- #define **MMCSD_CSD_MMC_WRITE_BLK_MISALIGN_SLICE** 78U,78U
- #define **MMCSD_CSD_MMC_READ_BLK_MISALIGN_SLICE** 77U,77U
- #define **MMCSD_CSD_MMC_DSR_IMP_SLICE** 76U,76U
- #define **MMCSD_CSD_MMC_C_SIZE_SLICE** 73U,62U
- #define **MMCSD_CSD_MMC_VDD_R_CURR_MIN_SLICE** 61U,59U
- #define **MMCSD_CSD_MMC_VDD_R_CURR_MAX_SLICE** 58U,56U
- #define **MMCSD_CSD_MMC_VDD_W_CURR_MIN_SLICE** 55U,53U
- #define **MMCSD_CSD_MMC_VDD_W_CURR_MAX_SLICE** 52U,50U
- #define **MMCSD_CSD_MMC_C_SIZE_MULT_SLICE** 49U,47U
- #define **MMCSD_CSD_MMC_ERASE_GRP_SIZE_SLICE** 46U,42U
- #define **MMCSD_CSD_MMC_ERASE_GRP_MULT_SLICE** 41U,37U
- #define **MMCSD_CSD_MMC_WP_GRP_SIZE_SLICE** 36U,32U
- #define **MMCSD_CSD_MMC_WP_GRP_ENABLE_SLICE** 31U,31U
- #define **MMCSD_CSD_MMC_DEFAULT_ECC_SLICE** 30U,29U
- #define **MMCSD_CSD_MMC_R2W_FACTOR_SLICE** 28U,26U
- #define **MMCSD_CSD_MMC_WRITE_BL_LEN_SLICE** 25U,22U
- #define **MMCSD_CSD_MMC_WRITE_BL_PARTIAL_SLICE** 21U,21U
- #define **MMCSD_CSD_MMC_CONTENT_PROT_APP_SLICE** 16U,16U
- #define **MMCSD_CSD_MMC_FILE_FORMAT_GRP_SLICE** 15U,15U
- #define **MMCSD_CSD_MMC_COPY_SLICE** 14U,14U
- #define **MMCSD_CSD_MMC_PERM_WRITE_PROTECT_SLICE** 13U,13U
- #define **MMCSD_CSD_MMC_TMP_WRITE_PROTECT_SLICE** 12U,12U
- #define **MMCSD_CSD_MMC_FILE_FORMAT_SLICE** 11U,10U
- #define **MMCSD_CSD_MMC_ECC_SLICE** 9U,8U
- #define **MMCSD_CSD_MMC_CRC_SLICE** 7U,1U
- #define **MMCSD_CSD_20_CRC_SLICE** 7U,1U
- #define **MMCSD_CSD_20_FILE_FORMAT_SLICE** 11U,10U
- #define **MMCSD_CSD_20_TMP_WRITE_PROTECT_SLICE** 12U,12U
- #define **MMCSD_CSD_20_PERM_WRITE_PROTECT_SLICE** 13U,13U
- #define **MMCSD_CSD_20_COPY_SLICE** 14U,14U
- #define **MMCSD_CSD_20_FILE_FORMAT_GRP_SLICE** 15U,15U
- #define **MMCSD_CSD_20_WRITE_BL_PARTIAL_SLICE** 21U,21U

- #define **MMCSD_CSD_20_WRITE_BL_LEN_SLICE** 25U,12U
- #define **MMCSD_CSD_20_R2W_FACTOR_SLICE** 28U,26U
- #define **MMCSD_CSD_20_WP_GRP_ENABLE_SLICE** 31U,31U
- #define **MMCSD_CSD_20_WP_GRP_SIZE_SLICE** 38U,32U
- #define **MMCSD_CSD_20_ERASE_SECTOR_SIZE_SLICE** 45U,39U
- #define **MMCSD_CSD_20_ERASE_BLK_EN_SLICE** 46U,46U
- #define **MMCSD_CSD_20_C_SIZE_SLICE** 69U,48U
- #define **MMCSD_CSD_20_DSR_IMP_SLICE** 76U,76U
- #define **MMCSD_CSD_20_READ_BLK_MISALIGN_SLICE** 77U,77U
- #define **MMCSD_CSD_20_WRITE_BLK_MISALIGN_SLICE** 78U,78U
- #define **MMCSD_CSD_20_READ_BL_PARTIAL_SLICE** 79U,79U
- #define **MMCSD_CSD_20_READ_BL_LEN_SLICE** 83U,80U
- #define **MMCSD_CSD_20_CCC_SLICE** 95U,84U
- #define **MMCSD_CSD_20_TRANS_SPEED_SLICE** 103U,96U
- #define **MMCSD_CSD_20_NSAC_SLICE** 111U,104U
- #define **MMCSD_CSD_20_TAAC_SLICE** 119U,112U
- #define **MMCSD_CSD_20_CSD_STRUCTURE_SLICE** 127U,126U
- #define **MMCSD_CSD_10_CRC_SLICE** MMCSD_CSD_20_CRC_SLICE
- #define **MMCSD_CSD_10_FILE_FORMAT_SLICE** MMCSD_CSD_20_FILE_FORMAT_SLICE
- #define **MMCSD_CSD_10_TMP_WRITE_PROTECT_SLICE** MMCSD_CSD_20_TMP_WRITE_PROT↩ ECT_SLICE
- #define **MMCSD_CSD_10_PERM_WRITE_PROTECT_SLICE** MMCSD_CSD_20_PERM_WRITE_PR↩ OTECT_SLICE
- #define **MMCSD_CSD_10_COPY_SLICE** MMCSD_CSD_20_COPY_SLICE
- #define **MMCSD_CSD_10_FILE_FORMAT_GRP_SLICE** MMCSD_CSD_20_FILE_FORMAT_GRP_S↩ LICE
- #define **MMCSD_CSD_10_WRITE_BL_PARTIAL_SLICE** MMCSD_CSD_20_WRITE_BL_PARTIAL_S↩ LICE
- #define **MMCSD_CSD_10_WRITE_BL_LEN_SLICE** MMCSD_CSD_20_WRITE_BL_LEN_SLICE
- #define **MMCSD_CSD_10_R2W_FACTOR_SLICE** MMCSD_CSD_20_R2W_FACTOR_SLICE
- #define **MMCSD_CSD_10_WP_GRP_ENABLE_SLICE** MMCSD_CSD_20_WP_GRP_ENABLE_SLICE
- #define **MMCSD_CSD_10_WP_GRP_SIZE_SLICE** MMCSD_CSD_20_WP_GRP_SIZE_SLICE
- #define **MMCSD_CSD_10_ERASE_SECTOR_SIZE_SLICE** MMCSD_CSD_20_ERASE_SECTOR_SI↩ ZE_SLICE
- #define **MMCSD_CSD_10_ERASE_BLK_EN_SLICE** MMCSD_CSD_20_ERASE_BLK_EN_SLICE
- #define **MMCSD_CSD_10_C_SIZE_MULT_SLICE** 49U,47U
- #define **MMCSD_CSD_10_VDD_W_CURR_MAX_SLICE** 52U,50U
- #define **MMCSD_CSD_10_VDD_W_CURR_MIN_SLICE** 55U,53U
- #define **MMCSD_CSD_10_VDD_R_CURR_MAX_SLICE** 58U,56U
- #define **MMCSD_CSD_10_VDD_R_CURR_MIX_SLICE** 61U,59U
- #define **MMCSD_CSD_10_C_SIZE_SLICE** 73U,62U
- #define **MMCSD_CSD_10_DSR_IMP_SLICE** MMCSD_CSD_20_DSR_IMP_SLICE
- #define **MMCSD_CSD_10_READ_BLK_MISALIGN_SLICE** MMCSD_CSD_20_READ_BLK_MISALIG↩ N_SLICE
- #define **MMCSD_CSD_10_WRITE_BLK_MISALIGN_SLICE** MMCSD_CSD_20_WRITE_BLK_MISALI↩ GN_SLICE
- #define **MMCSD_CSD_10_READ_BL_PARTIAL_SLICE** MMCSD_CSD_20_READ_BL_PARTIAL_SLI↩ CE
- #define **MMCSD_CSD_10_READ_BL_LEN_SLICE** 83U,80U
- #define **MMCSD_CSD_10_CCC_SLICE** MMCSD_CSD_20_CCC_SLICE
- #define **MMCSD_CSD_10_TRANS_SPEED_SLICE** MMCSD_CSD_20_TRANS_SPEED_SLICE
- #define **MMCSD_CSD_10_NSAC_SLICE** MMCSD_CSD_20_NSAC_SLICE
- #define **MMCSD_CSD_10_TAAC_SLICE** MMCSD_CSD_20_TAAC_SLICE
- #define **MMCSD_CSD_10_CSD_STRUCTURE_SLICE** MMCSD_CSD_20_CSD_STRUCTURE_SLICE

CID record offsets

- #define MMCSD_CID_SDC_CRC_SLICE 7U,1U

  Slice position of values in CID register.
- #define **MMCSD_CID_SDC_MDT_M_SLICE** 11U,8U
- #define **MMCSD_CID_SDC_MDT_Y_SLICE** 19U,12U
- #define **MMCSD_CID_SDC_PSN_SLICE** 55U,24U
- #define **MMCSD_CID_SDC_PRV_M_SLICE** 59U,56U

- #define **MMCSD_CID_SDC_PRV_N_SLICE** 63U,60U
- #define **MMCSD_CID_SDC_PNM0_SLICE** 71U,64U
- #define **MMCSD_CID_SDC_PNM1_SLICE** 79U,72U
- #define **MMCSD_CID_SDC_PNM2_SLICE** 87U,80U
- #define **MMCSD_CID_SDC_PNM3_SLICE** 95U,88U
- #define **MMCSD_CID_SDC_PNM4_SLICE** 103U,96U
- #define **MMCSD_CID_SDC_OID_SLICE** 119U,104U
- #define **MMCSD_CID_SDC_MID_SLICE** 127U,120U
- #define **MMCSD_CID_MMC_CRC_SLICE** 7U,1U
- #define **MMCSD_CID_MMC_MDT_Y_SLICE** 11U,8U
- #define **MMCSD_CID_MMC_MDT_M_SLICE** 15U,12U
- #define **MMCSD_CID_MMC_PSN_SLICE** 47U,16U
- #define **MMCSD_CID_MMC_PRV_M_SLICE** 51U,48U
- #define **MMCSD_CID_MMC_PRV_N_SLICE** 55U,52U
- #define **MMCSD_CID_MMC_PNM0_SLICE** 63U,56U
- #define **MMCSD_CID_MMC_PNM1_SLICE** 71U,64U
- #define **MMCSD_CID_MMC_PNM2_SLICE** 79U,72U
- #define **MMCSD_CID_MMC_PNM3_SLICE** 87U,80U
- #define **MMCSD_CID_MMC_PNM4_SLICE** 95U,88U
- #define **MMCSD_CID_MMC_PNM5_SLICE** 103U,96U
- #define **MMCSD_CID_MMC_OID_SLICE** 119U,104U
- #define **MMCSD_CID_MMC_MID_SLICE** 127U,120U

R1 response utilities

- #define MMCSD_R1_ERROR(r1) (((r1) & MMCSD_R1_ERROR_MASK) != 0U)

  Evaluates to `TRUE` if the R1 response contains error flags.
- #define MMCSD_R1_STS(r1) (((r1) >> 9U) & 15U)

  Returns the status field of an R1 response.
- #define MMCSD_R1_IS_CARD_LOCKED(r1) ((((r1) >> 21U) & 1U) != 0U)

  Evaluates to `TRUE` if the R1 response indicates a locked card.

Macro Functions

- #define mmcsdGetCardCapacity(ip) ((ip)->capacity)

  Returns the card capacity in blocks.

Functions

- uint32_t _mmcsd_get_slice (const uint32_t ∗data, uint32_t end, uint32_t start)

  Gets a bit field from a words array.
- uint32_t _mmcsd_get_capacity (const uint32_t ∗csd)

  Extract card capacity from a CSD.
- uint32_t _mmcsd_get_capacity_ext (const uint8_t ∗ext_csd)

  Extract MMC card capacity from EXT_CSD.
- void _mmcsd_unpack_sdc_cid (const MMCSDBlockDevice ∗sdcp, unpacked_sdc_cid_t ∗cidsdc)

  Unpacks SDC CID array in structure.
- void _mmcsd_unpack_mmc_cid (const MMCSDBlockDevice ∗sdcp, unpacked_mmc_cid_t ∗cidmmc)

  Unpacks MMC CID array in structure.
- void _mmcsd_unpack_csd_mmc (const MMCSDBlockDevice ∗sdcp, unpacked_mmc_csd_t ∗csdmmc)

  Unpacks MMC CSD array in structure.
- void _mmcsd_unpack_csd_v10 (const MMCSDBlockDevice ∗sdcp, unpacked_sdc_csd_10_t ∗csd10)

  Unpacks SDC CSD v1.0 array in structure.
- void _mmcsd_unpack_csd_v20 (const MMCSDBlockDevice ∗sdcp, unpacked_sdc_csd_20_t ∗csd20)

  Unpacks SDC CSD v2.0 array in structure.

9.50.1 Detailed Description

MMC/SD cards common header.

This header defines an abstract interface useful to access MMC/SD I/O block devices in a standardized way.

9.51 hal_pal.c File Reference

I/O Ports Abstraction Layer code.

```
#include "hal.h"
```

Functions

- ioportmask_t palReadBus (IOBus ∗bus)

  Read from an I/O bus.
- void palWriteBus (IOBus ∗bus, ioportmask_t bits)

  Write to an I/O bus.
- void palSetBusMode (IOBus ∗bus, iomode_t mode)

  Programs a bus with the specified mode.

9.51.1 Detailed Description

I/O Ports Abstraction Layer code.

9.52 hal_pal.h File Reference

I/O Ports Abstraction Layer macros, types and structures.

```
#include "hal_pal_lld.h"
```

Data Structures

- struct IOBus

  I/O bus descriptor.

Macros

- #define PAL_PORT_BIT(n) ((ioportmask_t)(1U << (n)))

  Port bit helper macro.
- #define PAL_GROUP_MASK(width) ((ioportmask_t)(1U << (width)) - 1U)

  Bits group mask helper.
- #define _IOBUS_DATA(name, port, width, offset) {port, PAL_GROUP_MASK(width), offset}

  Data part of a static I/O bus initializer.
- #define IOBUS_DECL(name, port, width, offset) IOBus name = _IOBUS_DATA(name, port, width, offset)

  Static I/O bus initializer.

Pads mode constants

- #define PAL_MODE_RESET 0U
  After reset state.
- #define PAL_MODE_UNCONNECTED 1U
  *Safe state for **unconnected** pads.*
- #define PAL_MODE_INPUT 2U
  Regular input high-Z pad.
- #define PAL_MODE_INPUT_PULLUP 3U
  Input pad with weak pull up resistor.
- #define PAL_MODE_INPUT_PULLDOWN 4U
  Input pad with weak pull down resistor.
- #define PAL_MODE_INPUT_ANALOG 5U
  Analog input mode.
- #define PAL_MODE_OUTPUT_PUSHPULL 6U
  Push-pull output pad.
- #define PAL_MODE_OUTPUT_OPENDRAIN 7U
  Open-drain output pad.

Logic level constants

- #define PAL_LOW 0U
  Logical low state.
- #define PAL_HIGH 1U
  Logical high state.

PAL event modes

- #define PAL_EVENT_MODE_EDGES_MASK 3U
  Mask of edges field.
- #define PAL_EVENT_MODE_DISABLED 0U
  Channel disabled.
- #define PAL_EVENT_MODE_RISING_EDGE 1U
  Rising edge callback.
- #define PAL_EVENT_MODE_FALLING_EDGE 2U
  Falling edge callback.
- #define PAL_EVENT_MODE_BOTH_EDGES 3U
  Both edges callback.

Macro Functions

- #define palInit(config) pal_lld_init(config)
  PAL subsystem initialization.
- #define palReadPort(port) ((void)(port), 0U)
  Reads the physical I/O port states.
- #define palReadLatch(port) ((void)(port), 0U)
  Reads the output latch.
- #define palWritePort(port, bits) ((void)(port), (void)(bits))
  Writes a bits mask on a I/O port.
- #define palSetPort(port, bits) palWritePort(port, palReadLatch(port) | (bits))
  Sets a bits mask on a I/O port.
- #define palClearPort(port, bits) palWritePort(port, palReadLatch(port) & ∼(bits))
  Clears a bits mask on a I/O port.
- #define palTogglePort(port, bits) palWritePort(port, palReadLatch(port) ^ (bits))
  Toggles a bits mask on a I/O port.
- #define palReadGroup(port, mask, offset) ((palReadPort(port) >> (offset)) & (mask))
  Reads a group of bits.
- #define palWriteGroup(port, mask, offset, bits)
  Writes a group of bits.
- #define palSetGroupMode(port, mask, offset, mode)

Pads group mode setup.
- #define palReadPad(port, pad) ((palReadPort(port) >> (pad)) & 1U)

  Reads an input pad logic state.
- #define palWritePad(port, pad, bit)

  Writes a logic state on an output pad.
- #define palSetPad(port, pad) palSetPort(port, PAL_PORT_BIT(pad))

  Sets a pad logic state to `PAL_HIGH`.
- #define palClearPad(port, pad) palClearPort(port, PAL_PORT_BIT(pad))

  Clears a pad logic state to `PAL_LOW`.
- #define palTogglePad(port, pad) palTogglePort(port, PAL_PORT_BIT(pad))

  Toggles a pad logic state.
- #define palSetPadMode(port, pad, mode) palSetGroupMode(port, PAL_PORT_BIT(pad), 0U, mode)

  Pad mode setup.
- #define palPadEnableEventI(port, pad, mode, callback)

  Pad event enable.
- #define palPadDisableEventI(port, pad)

  Pad event disable.
- #define palReadLine(line) palReadPad(PAL_PORT(line), PAL_PAD(line))

  Reads an input line logic state.
- #define palWriteLine(line, bit) palWritePad(PAL_PORT(line), PAL_PAD(line), bit)

  Writes a logic state on an output line.
- #define palSetLine(line) palSetPad(PAL_PORT(line), PAL_PAD(line))

  Sets a line logic state to `PAL_HIGH`.
- #define palClearLine(line) palClearPad(PAL_PORT(line), PAL_PAD(line))

  Clears a line logic state to `PAL_LOW`.
- #define palToggleLine(line) palTogglePad(PAL_PORT(line), PAL_PAD(line))

  Toggles a line logic state.
- #define palSetLineMode(line, mode) palSetPadMode(PAL_PORT(line), PAL_PAD(line), mode)

  Line mode setup.
- #define palLineEnableEventI(line, mode, callback) palPadEnableEventI(PAL_PORT(line), PAL_PAD(line), mode, callback)

  Line event enable.
- #define palLineDisableEventI(line) palPadDisableEventI(PAL_PORT(line), PAL_PAD(line))

  Line event disable.

Typedefs

- typedef void(∗ palcallback_t) (void)

  Type of a PAL event callback.

Functions

- ioportmask_t palReadBus (IOBus ∗bus)

  Read from an I/O bus.
- void palWriteBus (IOBus ∗bus, ioportmask_t bits)

  Write to an I/O bus.
- void palSetBusMode (IOBus ∗bus, iomode_t mode)

  Programs a bus with the specified mode.

9.52.1 Detailed Description

I/O Ports Abstraction Layer macros, types and structures.

9.53 hal_pal_lld.c File Reference

PLATFORM PAL subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void _pal_lld_init (const PALConfig ∗config)

  STM32 I/O ports configuration.
- void _pal_lld_setgroupmode (ioportid_t port, ioportmask_t mask, iomode_t mode)

  Pads mode setup.

9.53.1 Detailed Description

PLATFORM PAL subsystem low level driver source.

9.54 hal_pal_lld.h File Reference

PLATFORM PAL subsystem low level driver header.

Data Structures

- struct PALConfig

  Generic I/O ports static initializer.

Macros

- #define IOPORT1 0

  First I/O port identifier.
- #define pal_lld_init(config) _pal_lld_init(config)

  Low level PAL subsystem initialization.
- #define pal_lld_readport(port) 0U

  Reads the physical I/O port states.
- #define pal_lld_readlatch(port) 0U

  Reads the output latch.
- #define pal_lld_writeport(port, bits)

  Writes a bits mask on a I/O port.
- #define pal_lld_setport(port, bits)

  Sets a bits mask on a I/O port.
- #define pal_lld_clearport(port, bits)

  Clears a bits mask on a I/O port.
- #define pal_lld_toggleport(port, bits)

  Toggles a bits mask on a I/O port.
- #define pal_lld_readgroup(port, mask, offset) 0U

  Reads a group of bits.
- #define pal_lld_writegroup(port, mask, offset, bits)

  Writes a group of bits.

- #define pal_lld_setgroupmode(port, mask, offset, mode) _pal_lld_setgroupmode(port, mask << offset, mode)

  Pads group mode setup.
- #define pal_lld_readpad(port, pad) PAL_LOW

  Reads a logical state from an I/O pad.
- #define pal_lld_writepad(port, pad, bit)

  Writes a logical state on an output pad.
- #define pal_lld_setpad(port, pad)

  Sets a pad logical state to `PAL_HIGH`.
- #define pal_lld_clearpad(port, pad)

  Clears a pad logical state to `PAL_LOW`.
- #define pal_lld_togglepad(port, pad)

  Toggles a pad logical state.
- #define pal_lld_setpadmode(port, pad, mode)

  Pad mode setup.

Port related definitions

- #define PAL_IOPORTS_WIDTH 16U

  Width, in bits, of an I/O port.
- #define PAL_WHOLE_PORT ((ioportmask_t)0xFFFFU)

  Whole port mask.

Line handling macros

- #define PAL_LINE(port, pad) ((ioline_t)((uint32_t)(port)) | ((uint32_t)(pad)))

  Forms a line identifier.
- #define PAL_PORT(line) ((stm32_gpio_t *)(((uint32_t)(line)) & 0xFFFFFFF0U))

  Decodes a port identifier from a line identifier.
- #define PAL_PAD(line) ((uint32_t)((uint32_t)(line) & 0x0000000FU))

  Decodes a pad identifier from a line identifier.
- #define PAL_NOLINE 0U

  Value identifying an invalid line.

Typedefs

- typedef uint32_t ioportmask_t

  Digital I/O port sized unsigned type.
- typedef uint32_t iomode_t

  Digital I/O modes.
- typedef uint32_t ioline_t

  Type of an I/O line.
- typedef uint32_t ioportid_t

  Port Identifier.

Functions

- void _pal_lld_init (const PALConfig *config)

  STM32 I/O ports configuration.
- void _pal_lld_setgroupmode (ioportid_t port, ioportmask_t mask, iomode_t mode)

  Pads mode setup.

9.54.1 Detailed Description

PLATFORM PAL subsystem low level driver header.

9.55 hal_pwm.c File Reference

PWM Driver code.

```
#include "hal.h"
```

Functions

- void pwmInit (void)

  PWM Driver initialization.
- void pwmObjectInit (PWMDriver ∗pwmp)

  Initializes the standard part of a `PWMDriver` structure.
- void pwmStart (PWMDriver ∗pwmp, const PWMConfig ∗config)

  Configures and activates the PWM peripheral.
- void pwmStop (PWMDriver ∗pwmp)

  Deactivates the PWM peripheral.
- void pwmChangePeriod (PWMDriver ∗pwmp, pwmcnt_t period)

  Changes the period the PWM peripheral.
- void pwmEnableChannel (PWMDriver ∗pwmp, pwmchannel_t channel, pwmcnt_t width)

  Enables a PWM channel.
- void pwmDisableChannel (PWMDriver ∗pwmp, pwmchannel_t channel)

  Disables a PWM channel and its notification.
- void pwmEnablePeriodicNotification (PWMDriver ∗pwmp)

  Enables the periodic activation edge notification.
- void pwmDisablePeriodicNotification (PWMDriver ∗pwmp)

  Disables the periodic activation edge notification.
- void pwmEnableChannelNotification (PWMDriver ∗pwmp, pwmchannel_t channel)

  Enables a channel de-activation edge notification.
- void pwmDisableChannelNotification (PWMDriver ∗pwmp, pwmchannel_t channel)

  Disables a channel de-activation edge notification.

9.55.1 Detailed Description

PWM Driver code.

9.56 hal_pwm.h File Reference

PWM Driver macros and structures.

```
#include "hal_pwm_lld.h"
```

Macros

PWM output mode macros

- #define PWM_OUTPUT_MASK 0x0FU

  Standard output modes mask.
- #define PWM_OUTPUT_DISABLED 0x00U

  Output not driven, callback only.
- #define PWM_OUTPUT_ACTIVE_HIGH 0x01U

  Positive PWM logic, active is logic level one.
- #define PWM_OUTPUT_ACTIVE_LOW 0x02U

  Inverse PWM logic, active is logic level zero.

PWM duty cycle conversion

- #define PWM_FRACTION_TO_WIDTH(pwmp, denominator, numerator)

  Converts from fraction to pulse width.
- #define PWM_DEGREES_TO_WIDTH(pwmp, degrees) PWM_FRACTION_TO_WIDTH(pwmp, 36000, degrees)

  Converts from degrees to pulse width.
- #define PWM_PERCENTAGE_TO_WIDTH(pwmp, percentage) PWM_FRACTION_TO_WIDTH(pwmp, 10000, percentage)

  Converts from percentage to pulse width.

Macro Functions

- #define pwmChangePeriodI(pwmp, value)

  Changes the period the PWM peripheral.
- #define pwmEnableChannelI(pwmp, channel, width)

  Enables a PWM channel.
- #define pwmDisableChannelI(pwmp, channel)

  Disables a PWM channel.
- #define pwmIsChannelEnabledI(pwmp, channel) (((pwmp)->enabled & ((pwmchnmsk_t)1U << (pwmchnmsk_t)(channel))) != 0U)

  Returns a PWM channel status.
- #define pwmEnablePeriodicNotificationI(pwmp) pwm_lld_enable_periodic_notification(pwmp)

  Enables the periodic activation edge notification.
- #define pwmDisablePeriodicNotificationI(pwmp) pwm_lld_disable_periodic_notification(pwmp)

  Disables the periodic activation edge notification.
- #define pwmEnableChannelNotificationI(pwmp, channel) pwm_lld_enable_channel_notification(pwmp, channel)

  Enables a channel de-activation edge notification.
- #define pwmDisableChannelNotificationI(pwmp, channel) pwm_lld_disable_channel_notification(pwmp, channel)

  Disables a channel de-activation edge notification.

Typedefs

- typedef struct PWMDriver PWMDriver

  Type of a structure representing a PWM driver.
- typedef void(∗ pwmcallback_t) (PWMDriver ∗pwmp)

  Type of a PWM notification callback.

Enumerations

Functions

- void pwmInit (void)

  PWM Driver initialization.
- void pwmObjectInit (PWMDriver ∗pwmp)

  Initializes the standard part of a `PWMDriver` structure.
- void pwmStart (PWMDriver ∗pwmp, const PWMConfig ∗config)

  Configures and activates the PWM peripheral.
- void pwmStop (PWMDriver ∗pwmp)

  Deactivates the PWM peripheral.
- void pwmChangePeriod (PWMDriver ∗pwmp, pwmcnt_t period)

  Changes the period the PWM peripheral.
- void pwmEnableChannel (PWMDriver ∗pwmp, pwmchannel_t channel, pwmcnt_t width)

  Enables a PWM channel.
- void pwmDisableChannel (PWMDriver ∗pwmp, pwmchannel_t channel)

  Disables a PWM channel and its notification.
- void pwmEnablePeriodicNotification (PWMDriver ∗pwmp)

  Enables the periodic activation edge notification.
- void pwmDisablePeriodicNotification (PWMDriver ∗pwmp)

  Disables the periodic activation edge notification.
- void pwmEnableChannelNotification (PWMDriver ∗pwmp, pwmchannel_t channel)

  Enables a channel de-activation edge notification.
- void pwmDisableChannelNotification (PWMDriver ∗pwmp, pwmchannel_t channel)

  Disables a channel de-activation edge notification.

9.56.1 Detailed Description

PWM Driver macros and structures.

9.57 hal_pwm_lld.c File Reference

PLATFORM PWM subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void pwm_lld_init (void)

  Low level PWM driver initialization.
- void pwm_lld_start (PWMDriver ∗pwmp)

  Configures and activates the PWM peripheral.
- void pwm_lld_stop (PWMDriver ∗pwmp)

  Deactivates the PWM peripheral.
- void pwm_lld_enable_channel (PWMDriver ∗pwmp, pwmchannel_t channel, pwmcnt_t width)

  Enables a PWM channel.
- void pwm_lld_disable_channel (PWMDriver ∗pwmp, pwmchannel_t channel)

  Disables a PWM channel and its notification.

- void pwm_lld_enable_periodic_notification (PWMDriver ∗pwmp)

  Enables the periodic activation edge notification.
- void pwm_lld_disable_periodic_notification (PWMDriver ∗pwmp)

  Disables the periodic activation edge notification.
- void pwm_lld_enable_channel_notification (PWMDriver ∗pwmp, pwmchannel_t channel)

  Enables a channel de-activation edge notification.
- void pwm_lld_disable_channel_notification (PWMDriver ∗pwmp, pwmchannel_t channel)

  Disables a channel de-activation edge notification.

Variables

- PWMDriver PWMD1

  PWMD1 driver identifier.

9.57.1 Detailed Description

PLATFORM PWM subsystem low level driver source.

9.58 hal_pwm_lld.h File Reference

PLATFORM PWM subsystem low level driver header.

Data Structures

- struct PWMChannelConfig

  Type of a PWM driver channel configuration structure.
- struct PWMConfig

  Type of a PWM driver configuration structure.
- struct PWMDriver

  Structure representing a PWM driver.

Macros

- #define PWM_CHANNELS 4

  Number of PWM channels per PWM driver.
- #define pwm_lld_change_period(pwmp, period)

  Changes the period the PWM peripheral.

PLATFORM configuration options

- #define PLATFORM_PWM_USE_PWM1 FALSE

  PWMD1 driver enable switch.

Typedefs

- typedef uint32_t pwmmode_t

  Type of a PWM mode.
- typedef uint8_t pwmchannel_t

  Type of a PWM channel.
- typedef uint32_t pwmchnmsk_t

  Type of a channels mask.
- typedef uint32_t pwmcnt_t

  Type of a PWM counter.

Functions

- void pwm_lld_init (void)

  Low level PWM driver initialization.
- void pwm_lld_start (PWMDriver ∗pwmp)

  Configures and activates the PWM peripheral.
- void pwm_lld_stop (PWMDriver ∗pwmp)

  Deactivates the PWM peripheral.
- void pwm_lld_enable_channel (PWMDriver ∗pwmp, pwmchannel_t channel, pwmcnt_t width)

  Enables a PWM channel.
- void pwm_lld_disable_channel (PWMDriver ∗pwmp, pwmchannel_t channel)

  Disables a PWM channel and its notification.
- void pwm_lld_enable_periodic_notification (PWMDriver ∗pwmp)

  Enables the periodic activation edge notification.
- void pwm_lld_disable_periodic_notification (PWMDriver ∗pwmp)

  Disables the periodic activation edge notification.
- void pwm_lld_enable_channel_notification (PWMDriver ∗pwmp, pwmchannel_t channel)

  Enables a channel de-activation edge notification.
- void pwm_lld_disable_channel_notification (PWMDriver ∗pwmp, pwmchannel_t channel)

  Disables a channel de-activation edge notification.

9.58.1 Detailed Description

PLATFORM PWM subsystem low level driver header.

9.59 hal_qspi.c File Reference

QSPI Driver code.

```
#include "hal.h"
```

Functions

- void qspiInit (void)

  QSPI Driver initialization.
- void qspiObjectInit (QSPIDriver ∗qspip)

  Initializes the standard part of a `QSPIDriver` structure.
- void qspiStart (QSPIDriver ∗qspip, const QSPIConfig ∗config)

Configures and activates the QSPI peripheral.
- void qspiStop (QSPIDriver ∗qspip)

  Deactivates the QSPI peripheral.
- void qspiStartCommand (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp)

  Sends a command without data phase.
- void qspiStartSend (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp, size_t n, const uint8_t ∗txbuf)

  Sends a command with data over the QSPI bus.
- void qspiStartReceive (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp, size_t n, uint8_t ∗rxbuf)

  Sends a command then receives data over the QSPI bus.
- void qspiCommand (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp)

  Sends a command without data phase.
- void qspiSend (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp, size_t n, const uint8_t ∗txbuf)

  Sends a command with data over the QSPI bus.
- void qspiReceive (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp, size_t n, uint8_t ∗rxbuf)

  Sends a command then receives data over the QSPI bus.
- void qspiMapFlash (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp, uint8_t ∗∗addrp)

  Maps in memory space a QSPI flash device.
- void qspiUnmapFlash (QSPIDriver ∗qspip)

  Maps in memory space a QSPI flash device.
- void qspiAcquireBus (QSPIDriver ∗qspip)

  Gains exclusive access to the QSPI bus.
- void qspiReleaseBus (QSPIDriver ∗qspip)

  Releases exclusive access to the QSPI bus.

9.59.1 Detailed Description

QSPI Driver code.

9.60 hal_qspi.h File Reference

QSPI Driver macros and structures.

```
#include "hal_qspi_lld.h"
```

Data Structures

- struct qspi_command_t

  Type of a QSPI command descriptor.

Macros

Transfer options

- #define **QSPI_CFG_CMD_MASK** (0xFFLU << 0LU)
- #define **QSPI_CFG_CMD**(n) ((n) << 0LU)
- #define **QSPI_CFG_CMD_MODE_MASK** (3LU << 8LU)
- #define **QSPI_CFG_CMD_MODE_NONE** (0LU << 8LU)
- #define **QSPI_CFG_CMD_MODE_ONE_LINE** (1LU << 8LU)
- #define **QSPI_CFG_CMD_MODE_TWO_LINES** (2LU << 8LU)
- #define **QSPI_CFG_CMD_MODE_FOUR_LINES** (3LU << 8LU)
- #define **QSPI_CFG_ADDR_MODE_MASK** (3LU << 10LU)

- #define **QSPI_CFG_ADDR_MODE_NONE** (0LU << 10LU)
- #define **QSPI_CFG_ADDR_MODE_ONE_LINE** (1LU << 10LU)
- #define **QSPI_CFG_ADDR_MODE_TWO_LINES** (2LU << 10LU)
- #define **QSPI_CFG_ADDR_MODE_FOUR_LINES** (3LU << 10LU)
- #define **QSPI_CFG_ADDR_SIZE_MASK** (3LU << 12LU)
- #define **QSPI_CFG_ADDR_SIZE_8** (0LU << 12LU)
- #define **QSPI_CFG_ADDR_SIZE_16** (1LU << 12LU)
- #define **QSPI_CFG_ADDR_SIZE_24** (2LU << 12LU)
- #define **QSPI_CFG_ADDR_SIZE_32** (3LU << 12LU)
- #define **QSPI_CFG_ALT_MODE_MASK** (3LU << 14LU)
- #define **QSPI_CFG_ALT_MODE_NONE** (0LU << 14LU)
- #define **QSPI_CFG_ALT_MODE_ONE_LINE** (1LU << 14LU)
- #define **QSPI_CFG_ALT_MODE_TWO_LINES** (2LU << 14LU)
- #define **QSPI_CFG_ALT_MODE_FOUR_LINES** (3LU << 14LU)
- #define **QSPI_CFG_ALT_SIZE_MASK** (3LU << 16LU)
- #define **QSPI_CFG_ALT_SIZE_8** (0LU << 16LU)
- #define **QSPI_CFG_ALT_SIZE_16** (1LU << 16LU)
- #define **QSPI_CFG_ALT_SIZE_24** (2LU << 16LU)
- #define **QSPI_CFG_ALT_SIZE_32** (3LU << 16LU)
- #define **QSPI_CFG_DUMMY_CYCLES_MASK** (0x1FLU << 18LU)
- #define **QSPI_CFG_DUMMY_CYCLES**(n) ((n) << 18LU)
- #define **QSPI_CFG_DATA_MODE_MASK** (3LU << 24LU)
- #define **QSPI_CFG_DATA_MODE_NONE** (0LU << 24LU)
- #define **QSPI_CFG_DATA_MODE_ONE_LINE** (1LU << 24LU)
- #define **QSPI_CFG_DATA_MODE_TWO_LINES** (2LU << 24LU)
- #define **QSPI_CFG_DATA_MODE_FOUR_LINES** (3LU << 24LU)
- #define **QSPI_CFG_SIOO** (1LU << 28LU)
- #define **QSPI_CFG_DDRM** (1LU << 31LU)

QSPI configuration options

- #define QSPI_USE_WAIT TRUE

  Enables synchronous APIs.
- #define QSPI_USE_MUTUAL_EXCLUSION TRUE

  Enables the `qspiAcquireBus()` and `qspiReleaseBus()` APIs.

Macro Functions

- #define qspiStartCommandI(qspip, cmdp)

  Sends a command without data phase.
- #define qspiStartSendI(qspip, cmdp, n, txbuf)

  Sends data over the QSPI bus.
- #define qspiStartReceiveI(qspip, cmdp, n, rxbuf)

  Receives data from the QSPI bus.
- #define qspiMapFlashI(qspip, cmdp, addrp) qspi_lld_map_flash(qspip, cmdp, addrp)

  Maps in memory space a QSPI flash device.
- #define qspiUnmapFlashI(qspip) qspi_lld_unmap_flash(qspip)

  Maps in memory space a QSPI flash device.

Low level driver helper macros

- #define _qspi_wakeup_isr(qspip)

  Wakes up the waiting thread.
- #define _qspi_isr_code(qspip)

  Common ISR code.

Enumerations

Functions

- void qspiInit (void)

  QSPI Driver initialization.
- void qspiObjectInit (QSPIDriver ∗qspip)

  Initializes the standard part of a `QSPIDriver` structure.
- void qspiStart (QSPIDriver ∗qspip, const QSPIConfig ∗config)

  Configures and activates the QSPI peripheral.
- void qspiStop (QSPIDriver ∗qspip)

  Deactivates the QSPI peripheral.
- void qspiStartCommand (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp)

  Sends a command without data phase.
- void qspiStartSend (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp, size_t n, const uint8_t ∗txbuf)

  Sends a command with data over the QSPI bus.
- void qspiStartReceive (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp, size_t n, uint8_t ∗rxbuf)

  Sends a command then receives data over the QSPI bus.
- void qspiCommand (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp)

  Sends a command without data phase.
- void qspiSend (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp, size_t n, const uint8_t ∗txbuf)

  Sends a command with data over the QSPI bus.
- void qspiReceive (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp, size_t n, uint8_t ∗rxbuf)

  Sends a command then receives data over the QSPI bus.
- void qspiMapFlash (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp, uint8_t ∗∗addrp)

  Maps in memory space a QSPI flash device.
- void qspiUnmapFlash (QSPIDriver ∗qspip)

  Maps in memory space a QSPI flash device.
- void qspiAcquireBus (QSPIDriver ∗qspip)

  Gains exclusive access to the QSPI bus.
- void qspiReleaseBus (QSPIDriver ∗qspip)

  Releases exclusive access to the QSPI bus.

9.60.1 Detailed Description

QSPI Driver macros and structures.

9.61 hal_qspi_lld.c File Reference

PLATFORM QSPI subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void qspi_lld_init (void)

  Low level QSPI driver initialization.
- void qspi_lld_start (QSPIDriver ∗qspip)

  Configures and activates the QSPI peripheral.

- void qspi_lld_stop (QSPIDriver ∗qspip)

  Deactivates the QSPI peripheral.
- void qspi_lld_command (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp)

  Sends a command without data phase.
- void qspi_lld_send (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp, size_t n, const uint8_t ∗txbuf)

  Sends a command with data over the QSPI bus.
- void qspi_lld_receive (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp, size_t n, uint8_t ∗rxbuf)

  Sends a command then receives data over the QSPI bus.
- void qspi_lld_map_flash (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp, uint8_t ∗∗addrp)

  Maps in memory space a QSPI flash device.
- void qspi_lld_unmap_flash (QSPIDriver ∗qspip)

  Maps in memory space a QSPI flash device.

Variables

- QSPIDriver QSPID1

  QSPID1 driver identifier.

9.61.1 Detailed Description

PLATFORM QSPI subsystem low level driver source.

9.62 hal_qspi_lld.h File Reference

PLATFORM QSPI subsystem low level driver header.

Data Structures

- struct QSPIConfig

  Driver configuration structure.
- struct QSPIDriver

  Structure representing an QSPI driver.

Macros

QSPI capabilities

- #define **QSPI_SUPPORTS_MEMMAP** TRUE

Configuration options

- #define PLATFORM_QSPI_USE_QSPI1 FALSE

  QSPID1 driver enable switch.

Typedefs

- typedef struct QSPIDriver QSPIDriver

  Type of a structure representing an QSPI driver.
- typedef void(∗ qspicallback_t) (QSPIDriver ∗qspip)

  Type of a QSPI notification callback.

Functions

- void qspi_lld_init (void)
  Low level QSPI driver initialization.
- void qspi_lld_start (QSPIDriver ∗qspip)
  Configures and activates the QSPI peripheral.
- void qspi_lld_stop (QSPIDriver ∗qspip)
  Deactivates the QSPI peripheral.
- void qspi_lld_command (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp)
  Sends a command without data phase.
- void qspi_lld_send (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp, size_t n, const uint8_t ∗txbuf)
  Sends a command with data over the QSPI bus.
- void qspi_lld_receive (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp, size_t n, uint8_t ∗rxbuf)
  Sends a command then receives data over the QSPI bus.
- void qspi_lld_map_flash (QSPIDriver ∗qspip, const qspi_command_t ∗cmdp, uint8_t ∗∗addrp)
  Maps in memory space a QSPI flash device.
- void qspi_lld_unmap_flash (QSPIDriver ∗qspip)
  Maps in memory space a QSPI flash device.

9.62.1 Detailed Description

PLATFORM QSPI subsystem low level driver header.

9.63 hal_queues.c File Reference

I/O Queues code.

```
#include "hal.h"
```

Functions

- void iqObjectInit (input_queue_t ∗iqp, uint8_t ∗bp, size_t size, qnotify_t infy, void ∗link)
  Initializes an input queue.
- void iqResetI (input_queue_t ∗iqp)
  Resets an input queue.
- msg_t iqPutI (input_queue_t ∗iqp, uint8_t b)
  Input queue write.
- msg_t iqGetTimeout (input_queue_t ∗iqp, systime_t timeout)
  Input queue read with timeout.
- size_t iqReadTimeout (input_queue_t ∗iqp, uint8_t ∗bp, size_t n, systime_t timeout)
  Input queue read with timeout.
- void oqObjectInit (output_queue_t ∗oqp, uint8_t ∗bp, size_t size, qnotify_t onfy, void ∗link)
  Initializes an output queue.
- void oqResetI (output_queue_t ∗oqp)
  Resets an output queue.
- msg_t oqPutTimeout (output_queue_t ∗oqp, uint8_t b, systime_t timeout)
  Output queue write with timeout.
- msg_t oqGetI (output_queue_t ∗oqp)
  Output queue read.
- size_t oqWriteTimeout (output_queue_t ∗oqp, const uint8_t ∗bp, size_t n, systime_t timeout)
  Output queue write with timeout.

9.63.1 Detailed Description

I/O Queues code.

9.64 hal_queues.h File Reference

I/O Queues macros and structures.

Data Structures

- struct io_queue

  Generic I/O queue structure.

Macros

Queue functions returned status value

- #define Q_OK MSG_OK

  Operation successful.
- #define Q_TIMEOUT MSG_TIMEOUT

  Timeout condition.
- #define Q_RESET MSG_RESET

  Queue has been reset.
- #define Q_EMPTY MSG_TIMEOUT

  Queue empty.
- #define Q_FULL MSG_TIMEOUT

  Queue full,.

Macro Functions

- #define qSizeX(qp)

  Returns the queue's buffer size.
- #define qSpaceI(qp) ((qp)->q_counter)

  Queue space.
- #define qGetLink(qp) ((qp)->q_link)

  Returns the queue application-defined link.
- #define iqGetFullI(iqp) qSpaceI(iqp)

  Returns the filled space into an input queue.
- #define iqGetEmptyI(iqp) (qSizeX(iqp) - qSpaceI(iqp))

  Returns the empty space into an input queue.
- #define iqIsEmptyI(iqp) ((bool)(qSpaceI(iqp) == 0U))

  Evaluates to `true` *if the specified input queue is empty.*
- #define iqIsFullI(iqp)

  Evaluates to `true` *if the specified input queue is full.*
- #define iqGet(iqp) iqGetTimeout(iqp, TIME_INFINITE)

  Input queue read.
- #define oqGetFullI(oqp) (qSizeX(oqp) - qSpaceI(oqp))

  Returns the filled space into an output queue.
- #define oqGetEmptyI(oqp) qSpaceI(oqp)

  Returns the empty space into an output queue.
- #define oqIsEmptyI(oqp)

  Evaluates to `true` *if the specified output queue is empty.*
- #define oqIsFullI(oqp) ((bool)(qSpaceI(oqp) == 0U))

  Evaluates to `true` *if the specified output queue is full.*
- #define oqPut(oqp, b) oqPutTimeout(oqp, b, TIME_INFINITE)

  Output queue write.

Typedefs

- typedef struct io_queue io_queue_t

  Type of a generic I/O queue structure.
- typedef void(∗ qnotify_t) (io_queue_t ∗qp)

  Queue notification callback type.
- typedef io_queue_t input_queue_t

  Type of an input queue structure.
- typedef io_queue_t output_queue_t

  Type of an output queue structure.

Functions

- void iqObjectInit (input_queue_t ∗iqp, uint8_t ∗bp, size_t size, qnotify_t infy, void ∗link)

  Initializes an input queue.
- void iqResetI (input_queue_t ∗iqp)

  Resets an input queue.
- msg_t iqPutI (input_queue_t ∗iqp, uint8_t b)

  Input queue write.
- msg_t iqGetTimeout (input_queue_t ∗iqp, systime_t timeout)

  Input queue read with timeout.
- size_t iqReadTimeout (input_queue_t ∗iqp, uint8_t ∗bp, size_t n, systime_t timeout)

  Input queue read with timeout.
- void oqObjectInit (output_queue_t ∗oqp, uint8_t ∗bp, size_t size, qnotify_t onfy, void ∗link)

  Initializes an output queue.
- void oqResetI (output_queue_t ∗oqp)

  Resets an output queue.
- msg_t oqPutTimeout (output_queue_t ∗oqp, uint8_t b, systime_t timeout)

  Output queue write with timeout.
- msg_t oqGetI (output_queue_t ∗oqp)

  Output queue read.
- size_t oqWriteTimeout (output_queue_t ∗oqp, const uint8_t ∗bp, size_t n, systime_t timeout)

  Output queue write with timeout.

9.64.1 Detailed Description

I/O Queues macros and structures.

9.65 hal_rtc.c File Reference

RTC Driver code.

```
#include "hal.h"
```

Functions

- void rtcInit (void)

  RTC Driver initialization.
- void rtcObjectInit (RTCDriver ∗rtcp)

  Initializes a generic RTC driver object.
- void rtcSetTime (RTCDriver ∗rtcp, const RTCDateTime ∗timespec)

  Set current time.
- void rtcGetTime (RTCDriver ∗rtcp, RTCDateTime ∗timespec)

  Get current time.
- void rtcSetAlarm (RTCDriver ∗rtcp, rtcalarm_t alarm, const RTCAlarm ∗alarmspec)

  Set alarm time.
- void rtcGetAlarm (RTCDriver ∗rtcp, rtcalarm_t alarm, RTCAlarm ∗alarmspec)

  Get current alarm.
- void rtcSetCallback (RTCDriver ∗rtcp, rtccb_t callback)

  Enables or disables RTC callbacks.
- void rtcConvertDateTimeToStructTm (const RTCDateTime ∗timespec, struct tm ∗timp, uint32_t ∗tv_msec)

  Convert `RTCDateTime` to broken-down time structure.
- void rtcConvertStructTmToDateTime (const struct tm ∗timp, uint32_t tv_msec, RTCDateTime ∗timespec)

  Convert broken-down time structure to `RTCDateTime`.
- uint32_t rtcConvertDateTimeToFAT (const RTCDateTime ∗timespec)

  Get current time in format suitable for usage in FAT file system.

9.65.1 Detailed Description

RTC Driver code.

9.66 hal_rtc.h File Reference

RTC Driver macros and structures.

```
#include <time.h>
#include "hal_rtc_lld.h"
```

Data Structures

- struct RTCDateTime

  Type of a structure representing an RTC date/time stamp.

Macros

- #define RTC_BASE_YEAR 1980U

  Base year of the calendar.

Date/Time bit masks for FAT format

- #define **RTC_FAT_TIME_SECONDS_MASK** 0x0000001FU
- #define **RTC_FAT_TIME_MINUTES_MASK** 0x000007E0U
- #define **RTC_FAT_TIME_HOURS_MASK** 0x0000F800U
- #define **RTC_FAT_DATE_DAYS_MASK** 0x001F0000U

- #define **RTC_FAT_DATE_MONTHS_MASK** 0x01E00000U
- #define **RTC_FAT_DATE_YEARS_MASK** 0xFE000000U

Day of week encoding

- #define **RTC_DAY_CATURDAY** 0U
- #define **RTC_DAY_MONDAY** 1U
- #define **RTC_DAY_TUESDAY** 2U
- #define **RTC_DAY_WEDNESDAY** 3U
- #define **RTC_DAY_THURSDAY** 4U
- #define **RTC_DAY_FRIDAY** 5U
- #define **RTC_DAY_SATURDAY** 6U
- #define **RTC_DAY_SUNDAY** 7U

Typedefs

- typedef struct RTCDriver RTCDriver

  Type of a structure representing an RTC driver.

Functions

- void rtcInit (void)

  RTC Driver initialization.
- void rtcObjectInit (RTCDriver ∗rtcp)

  Initializes a generic RTC driver object.
- void rtcSetTime (RTCDriver ∗rtcp, const RTCDateTime ∗timespec)

  Set current time.
- void rtcGetTime (RTCDriver ∗rtcp, RTCDateTime ∗timespec)

  Get current time.
- void rtcSetCallback (RTCDriver ∗rtcp, rtccb_t callback)

  Enables or disables RTC callbacks.
- void rtcConvertDateTimeToStructTm (const RTCDateTime ∗timespec, struct tm ∗timp, uint32_t ∗tv_msec)

  Convert `RTCDateTime` to broken-down time structure.
- void rtcConvertStructTmToDateTime (const struct tm ∗timp, uint32_t tv_msec, RTCDateTime ∗timespec)

  Convert broken-down time structure to `RTCDateTime`.
- uint32_t rtcConvertDateTimeToFAT (const RTCDateTime ∗timespec)

  Get current time in format suitable for usage in FAT file system.

9.66.1 Detailed Description

RTC Driver macros and structures.

9.67 hal_rtc_lld.c File Reference

PLATFORM RTC subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void rtc_lld_init (void)

  RTC driver identifier.
- void rtc_lld_set_time (RTCDriver ∗rtcp, const RTCDateTime ∗timespec)

  Set current time.
- void rtc_lld_get_time (RTCDriver ∗rtcp, RTCDateTime ∗timespec)

  Get current time.
- void rtc_lld_set_alarm (RTCDriver ∗rtcp, rtcalarm_t alarm, const RTCAlarm ∗alarmspec)

  Set alarm time.
- void rtc_lld_get_alarm (RTCDriver ∗rtcp, rtcalarm_t alarm, RTCAlarm ∗alarmspec)

  Get alarm time.

9.67.1 Detailed Description

PLATFORM RTC subsystem low level driver source.

9.68 hal_rtc_lld.h File Reference

PLATFORM RTC subsystem low level driver header.

Data Structures

- struct RTCAlarm

  Type of a structure representing an RTC alarm time stamp.
- struct RTCDriverVMT

  `RTCDriver` *virtual methods table.*
- struct RTCDriver

  Structure representing an RTC driver.

Macros

- #define _rtc_driver_methods _file_stream_methods

  FileStream specific methods.

Implementation capabilities

- #define RTC_SUPPORTS_CALLBACKS TRUE

  Callback support int the driver.
- #define RTC_ALARMS 2

  Number of alarms available.
- #define RTC_HAS_STORAGE FALSE

  Presence of a local persistent storage.

PLATFORM configuration options

- #define PLATFORM_RTC_USE_RTC1 FALSE

  RTCD1 driver enable switch.

Typedefs

- typedef uint32_t rtcalarm_t

  Type of an RTC alarm number.
- typedef void(∗ rtccb_t) (RTCDriver ∗rtcp, rtcevent_t event)

  Type of a generic RTC callback.

Enumerations

Functions

- void rtc_lld_init (void)

  RTC driver identifier.
- void rtc_lld_set_time (RTCDriver ∗rtcp, const RTCDateTime ∗timespec)

  Set current time.
- void rtc_lld_get_time (RTCDriver ∗rtcp, RTCDateTime ∗timespec)

  Get current time.
- void rtc_lld_set_alarm (RTCDriver ∗rtcp, rtcalarm_t alarm, const RTCAlarm ∗alarmspec)

  Set alarm time.
- void rtc_lld_get_alarm (RTCDriver ∗rtcp, rtcalarm_t alarm, RTCAlarm ∗alarmspec)

  Get alarm time.

9.68.1 Detailed Description

PLATFORM RTC subsystem low level driver header.

9.69 hal_sdc.c File Reference

SDC Driver code.

```
#include <string.h>
#include "hal.h"
```

Enumerations

Functions

- static bool mode_detect (SDCDriver ∗sdcp)

  Detects card mode.
- static bool mmc_init (SDCDriver ∗sdcp)

  Init procedure for MMC.
- static bool sdc_init (SDCDriver ∗sdcp)

  Init procedure for SDC.
- static uint32_t mmc_cmd6_construct (mmc_switch_t access, uint32_t idx, uint32_t value, uint32_t cmd_set)

  Constructs CMD6 argument for MMC.
- static uint32_t sdc_cmd6_construct (sd_switch_t mode, sd_switch_function_t function, uint32_t value)

  Constructs CMD6 argument for SDC.
- static uint16_t sdc_cmd6_extract_info (sd_switch_function_t function, const uint8_t ∗buf)

  Extracts information from CMD6 answer.

- static bool sdc_cmd6_check_status (sd_switch_function_t function, const uint8_t ∗buf)

  Checks status after switching using CMD6.
- static bool sdc_detect_bus_clk (SDCDriver ∗sdcp, sdcbusclk_t ∗clk)

  Reads supported bus clock and switch SDC to appropriate mode.
- static bool mmc_detect_bus_clk (SDCDriver ∗sdcp, sdcbusclk_t ∗clk)

  Reads supported bus clock and switch MMC to appropriate mode.
- static bool detect_bus_clk (SDCDriver ∗sdcp, sdcbusclk_t ∗clk)

  Reads supported bus clock and switch card to appropriate mode.
- static bool sdc_set_bus_width (SDCDriver ∗sdcp)

  Sets bus width for SDC.
- static bool mmc_set_bus_width (SDCDriver ∗sdcp)

  Sets bus width for MMC.
- bool _sdc_wait_for_transfer_state (SDCDriver ∗sdcp)

  Wait for the card to complete pending operations.
- void sdcInit (void)

  SDC Driver initialization.
- void sdcObjectInit (SDCDriver ∗sdcp)

  Initializes the standard part of a `SDCDriver` structure.
- void sdcStart (SDCDriver ∗sdcp, const SDCConfig ∗config)

  Configures and activates the SDC peripheral.
- void sdcStop (SDCDriver ∗sdcp)

  Deactivates the SDC peripheral.
- bool sdcConnect (SDCDriver ∗sdcp)

  Performs the initialization procedure on the inserted card.
- bool sdcDisconnect (SDCDriver ∗sdcp)

  Brings the driver in a state safe for card removal.
- bool sdcRead (SDCDriver ∗sdcp, uint32_t startblk, uint8_t ∗buf, uint32_t n)

  Reads one or more blocks.
- bool sdcWrite (SDCDriver ∗sdcp, uint32_t startblk, const uint8_t ∗buf, uint32_t n)

  Writes one or more blocks.
- sdcflags_t sdcGetAndClearErrors (SDCDriver ∗sdcp)

  Returns the errors mask associated to the previous operation.
- bool sdcSync (SDCDriver ∗sdcp)

  Waits for card idle condition.
- bool sdcGetInfo (SDCDriver ∗sdcp, BlockDeviceInfo ∗bdip)

  Returns the media info.
- bool sdcErase (SDCDriver ∗sdcp, uint32_t startblk, uint32_t endblk)

  Erases the supplied blocks.

Variables

- static const struct SDCDriverVMT sdc_vmt

  Virtual methods table.

9.69.1 Detailed Description

SDC Driver code.

9.70 hal_sdc.h File Reference

SDC Driver macros and structures.

```
#include "hal_sdc_lld.h"
```

Macros

SD card types

- #define **SDC_MODE_CARDTYPE_MASK** 0xFU
- #define **SDC_MODE_CARDTYPE_SDV11** 0U
- #define **SDC_MODE_CARDTYPE_SDV20** 1U
- #define **SDC_MODE_CARDTYPE_MMC** 2U
- #define **SDC_MODE_HIGH_CAPACITY** 0x10U

SDC bus error conditions

- #define **SDC_NO_ERROR** 0U
- #define **SDC_CMD_CRC_ERROR** 1U
- #define **SDC_DATA_CRC_ERROR** 2U
- #define **SDC_DATA_TIMEOUT** 4U
- #define **SDC_COMMAND_TIMEOUT** 8U
- #define **SDC_TX_UNDERRUN** 16U
- #define **SDC_RX_OVERRUN** 32U
- #define **SDC_STARTBIT_ERROR** 64U
- #define **SDC_OVERFLOW_ERROR** 128U
- #define **SDC_UNHANDLED_ERROR** 0xFFFFFFFFU

SDC configuration options

- #define SDC_INIT_RETRY 100

  Number of initialization attempts before rejecting the card.
- #define SDC_MMC_SUPPORT FALSE

  Include support for MMC cards.
- #define SDC_NICE_WAITING TRUE

  Delays insertions.
- #define SDC_INIT_OCR_V20 0x50FF8000U

  OCR initialization constant for V20 cards.
- #define SDC_INIT_OCR 0x80100000U

  OCR initialization constant for non-V20 cards.

Macro Functions

- #define sdcIsCardInserted(sdcp) (sdc_lld_is_card_inserted(sdcp))

  Returns the card insertion status.
- #define sdcIsWriteProtected(sdcp) (sdc_lld_is_write_protected(sdcp))

  Returns the write protect status.

Enumerations

Functions

- void sdcInit (void)

  SDC Driver initialization.
- void sdcObjectInit (SDCDriver ∗sdcp)

Initializes the standard part of a `SDCDriver` structure.
- void sdcStart (SDCDriver ∗sdcp, const SDCConfig ∗config)

  Configures and activates the SDC peripheral.
- void sdcStop (SDCDriver ∗sdcp)

  Deactivates the SDC peripheral.
- bool sdcConnect (SDCDriver ∗sdcp)

  Performs the initialization procedure on the inserted card.
- bool sdcDisconnect (SDCDriver ∗sdcp)

  Brings the driver in a state safe for card removal.
- bool sdcRead (SDCDriver ∗sdcp, uint32_t startblk, uint8_t ∗buf, uint32_t n)

  Reads one or more blocks.
- bool sdcWrite (SDCDriver ∗sdcp, uint32_t startblk, const uint8_t ∗buf, uint32_t n)

  Writes one or more blocks.
- sdcflags_t sdcGetAndClearErrors (SDCDriver ∗sdcp)

  Returns the errors mask associated to the previous operation.
- bool sdcSync (SDCDriver ∗sdcp)

  Waits for card idle condition.
- bool sdcGetInfo (SDCDriver ∗sdcp, BlockDeviceInfo ∗bdip)

  Returns the media info.
- bool sdcErase (SDCDriver ∗sdcp, uint32_t startblk, uint32_t endblk)

  Erases the supplied blocks.
- bool _sdc_wait_for_transfer_state (SDCDriver ∗sdcp)

  Wait for the card to complete pending operations.

9.70.1 Detailed Description

SDC Driver macros and structures.

9.71 hal_sdc_lld.c File Reference

PLATFORM SDC subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void sdc_lld_init (void)

  Low level SDC driver initialization.
- void sdc_lld_start (SDCDriver ∗sdcp)

  Configures and activates the SDC peripheral.
- void sdc_lld_stop (SDCDriver ∗sdcp)

  Deactivates the SDC peripheral.
- void sdc_lld_start_clk (SDCDriver ∗sdcp)

  Starts the SDIO clock and sets it to init mode (400kHz or less).
- void sdc_lld_set_data_clk (SDCDriver ∗sdcp, sdcbusclk_t clk)

  Sets the SDIO clock to data mode (25MHz or less).
- void sdc_lld_stop_clk (SDCDriver ∗sdcp)

  Stops the SDIO clock.
- void sdc_lld_set_bus_mode (SDCDriver ∗sdcp, sdcbusmode_t mode)

Switches the bus to 4 bits mode.
- void sdc_lld_send_cmd_none (SDCDriver ∗sdcp, uint8_t cmd, uint32_t arg)

  Sends an SDIO command with no response expected.
- bool sdc_lld_send_cmd_short (SDCDriver ∗sdcp, uint8_t cmd, uint32_t arg, uint32_t ∗resp)

  Sends an SDIO command with a short response expected.
- bool sdc_lld_send_cmd_short_crc (SDCDriver ∗sdcp, uint8_t cmd, uint32_t arg, uint32_t ∗resp)

  Sends an SDIO command with a short response expected and CRC.
- bool sdc_lld_send_cmd_long_crc (SDCDriver ∗sdcp, uint8_t cmd, uint32_t arg, uint32_t ∗resp)

  Sends an SDIO command with a long response expected and CRC.
- bool sdc_lld_read (SDCDriver ∗sdcp, uint32_t startblk, uint8_t ∗buf, uint32_t n)

  Reads one or more blocks.
- bool sdc_lld_write (SDCDriver ∗sdcp, uint32_t startblk, const uint8_t ∗buf, uint32_t n)

  Writes one or more blocks.
- bool sdc_lld_sync (SDCDriver ∗sdcp)

  Waits for card idle condition.

Variables

- SDCDriver SDCD1

  SDCD1 driver identifier.

9.71.1 Detailed Description

PLATFORM SDC subsystem low level driver source.

9.72 hal_sdc_lld.h File Reference

PLATFORM SDC subsystem low level driver header.

Data Structures

- struct SDCConfig

  Driver configuration structure.
- struct SDCDriverVMT

  `SDCDriver` virtual methods table.
- struct SDCDriver

  Structure representing an SDC driver.

Macros

- #define _sdc_driver_methods _mmcsd_block_device_methods

  `SDCDriver` specific methods.

PLATFORM configuration options

- #define PLATFORM_SDC_USE_SDC1 FALSE

  PWMD1 driver enable switch.

Typedefs

- typedef uint32_t sdcmode_t

  Type of card flags.
- typedef uint32_t sdcflags_t

  SDC Driver condition flags type.
- typedef struct SDCDriver SDCDriver

  Type of a structure representing an SDC driver.

Functions

- void sdc_lld_init (void)

  Low level SDC driver initialization.
- void sdc_lld_start (SDCDriver ∗sdcp)

  Configures and activates the SDC peripheral.
- void sdc_lld_stop (SDCDriver ∗sdcp)

  Deactivates the SDC peripheral.
- void sdc_lld_start_clk (SDCDriver ∗sdcp)

  Starts the SDIO clock and sets it to init mode (400kHz or less).
- void sdc_lld_set_data_clk (SDCDriver ∗sdcp, sdcbusclk_t clk)

  Sets the SDIO clock to data mode (25MHz or less).
- void sdc_lld_stop_clk (SDCDriver ∗sdcp)

  Stops the SDIO clock.
- void sdc_lld_set_bus_mode (SDCDriver ∗sdcp, sdcbusmode_t mode)

  Switches the bus to 4 bits mode.
- void sdc_lld_send_cmd_none (SDCDriver ∗sdcp, uint8_t cmd, uint32_t arg)

  Sends an SDIO command with no response expected.
- bool sdc_lld_send_cmd_short (SDCDriver ∗sdcp, uint8_t cmd, uint32_t arg, uint32_t ∗resp)

  Sends an SDIO command with a short response expected.
- bool sdc_lld_send_cmd_short_crc (SDCDriver ∗sdcp, uint8_t cmd, uint32_t arg, uint32_t ∗resp)

  Sends an SDIO command with a short response expected and CRC.
- bool sdc_lld_send_cmd_long_crc (SDCDriver ∗sdcp, uint8_t cmd, uint32_t arg, uint32_t ∗resp)

  Sends an SDIO command with a long response expected and CRC.
- bool sdc_lld_read (SDCDriver ∗sdcp, uint32_t startblk, uint8_t ∗buf, uint32_t n)

  Reads one or more blocks.
- bool sdc_lld_write (SDCDriver ∗sdcp, uint32_t startblk, const uint8_t ∗buf, uint32_t n)

  Writes one or more blocks.
- bool sdc_lld_sync (SDCDriver ∗sdcp)

  Waits for card idle condition.

9.72.1 Detailed Description

PLATFORM SDC subsystem low level driver header.

9.73 hal_serial.c File Reference

Serial Driver code.

```
#include "hal.h"
```

Functions

- void sdInit (void)

  Serial Driver initialization.
- void sdObjectInit (SerialDriver ∗sdp, qnotify_t inotify, qnotify_t onotify)

  Initializes a generic full duplex driver object.
- void sdStart (SerialDriver ∗sdp, const SerialConfig ∗config)

  Configures and starts the driver.
- void sdStop (SerialDriver ∗sdp)

  Stops the driver.
- void sdIncomingDataI (SerialDriver ∗sdp, uint8_t b)

  Handles incoming data.
- msg_t sdRequestDataI (SerialDriver ∗sdp)

  Handles outgoing data.
- bool sdPutWouldBlock (SerialDriver ∗sdp)

  Direct output check on a `SerialDriver`.
- bool sdGetWouldBlock (SerialDriver ∗sdp)

  Direct input check on a `SerialDriver`.

9.73.1 Detailed Description

Serial Driver code.

9.74 hal_serial.h File Reference

Serial Driver macros and structures.

`#include "hal_serial_lld.h"`

Data Structures

- struct SerialDriverVMT

  `SerialDriver` virtual methods table.
- struct SerialDriver

  Full duplex serial driver class.

Macros

- #define _serial_driver_methods _base_asynchronous_channel_methods

  `SerialDriver` specific methods.

Serial status flags

- #define SD_PARITY_ERROR (eventflags_t)32

  Parity.
- #define SD_FRAMING_ERROR (eventflags_t)64

  Framing.
- #define SD_OVERRUN_ERROR (eventflags_t)128

  Overflow.
- #define SD_NOISE_ERROR (eventflags_t)256

Line noise.
- #define SD_BREAK_DETECTED (eventflags_t)512
  LIN Break.
- #define SD_QUEUE_FULL_ERROR (eventflags_t)1024
  Queue full.

Serial configuration options

- #define SERIAL_DEFAULT_BITRATE 38400
  Default bit rate.
- #define SERIAL_BUFFERS_SIZE 16
  Serial buffers size.

Macro Functions

- #define sdPut(sdp, b) oqPut(&(sdp)->oqueue, b)
  Direct write to a `SerialDriver`.
- #define sdPutTimeout(sdp, b, t) oqPutTimeout(&(sdp)->oqueue, b, t)
  Direct write to a `SerialDriver` with timeout specification.
- #define sdGet(sdp) iqGet(&(sdp)->iqueue)
  Direct read from a `SerialDriver`.
- #define sdGetTimeout(sdp, t) iqGetTimeout(&(sdp)->iqueue, t)
  Direct read from a `SerialDriver` with timeout specification.
- #define sdWrite(sdp, b, n) oqWriteTimeout(&(sdp)->oqueue, b, n, TIME_INFINITE)
  Direct blocking write to a `SerialDriver`.
- #define sdWriteTimeout(sdp, b, n, t) oqWriteTimeout(&(sdp)->oqueue, b, n, t)
  Direct blocking write to a `SerialDriver` with timeout specification.
- #define sdAsynchronousWrite(sdp, b, n) oqWriteTimeout(&(sdp)->oqueue, b, n, TIME_IMMEDIATE)
  Direct non-blocking write to a `SerialDriver`.
- #define sdRead(sdp, b, n) iqReadTimeout(&(sdp)->iqueue, b, n, TIME_INFINITE)
  Direct blocking read from a `SerialDriver`.
- #define sdReadTimeout(sdp, b, n, t) iqReadTimeout(&(sdp)->iqueue, b, n, t)
  Direct blocking read from a `SerialDriver` with timeout specification.
- #define sdAsynchronousRead(sdp, b, n) iqReadTimeout(&(sdp)->iqueue, b, n, TIME_IMMEDIATE)
  Direct non-blocking read from a `SerialDriver`.

Typedefs

- typedef struct SerialDriver SerialDriver
  Structure representing a serial driver.

Enumerations

Functions

- void sdInit (void)
  Serial Driver initialization.
- void sdObjectInit (SerialDriver ∗sdp, qnotify_t inotify, qnotify_t onotify)
  Initializes a generic full duplex driver object.
- void sdStart (SerialDriver ∗sdp, const SerialConfig ∗config)
  Configures and starts the driver.
- void sdStop (SerialDriver ∗sdp)
  Stops the driver.
- void sdIncomingDataI (SerialDriver ∗sdp, uint8_t b)
  Handles incoming data.

- msg_t sdRequestDataI (SerialDriver ∗sdp)

  Handles outgoing data.
- bool sdPutWouldBlock (SerialDriver ∗sdp)

  Direct output check on a `SerialDriver`.
- bool sdGetWouldBlock (SerialDriver ∗sdp)

  Direct input check on a `SerialDriver`.

9.74.1 Detailed Description

Serial Driver macros and structures.

9.75 hal_serial_lld.c File Reference

PLATFORM serial subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void sd_lld_init (void)

  Low level serial driver initialization.
- void sd_lld_start (SerialDriver ∗sdp, const SerialConfig ∗config)

  Low level serial driver configuration and (re)start.
- void sd_lld_stop (SerialDriver ∗sdp)

  Low level serial driver stop.

Variables

- SerialDriver SD1

  USART1 serial driver identifier.
- static const SerialConfig default_config

  Driver default configuration.

9.75.1 Detailed Description

PLATFORM serial subsystem low level driver source.

9.76 hal_serial_lld.h File Reference

PLATFORM serial subsystem low level driver header.

Data Structures

- struct SerialConfig

  PLATFORM Serial Driver configuration structure.

Macros

- #define _serial_driver_data

  `SerialDriver` *specific data.*

PLATFORM configuration options

- #define PLATFORM_SERIAL_USE_USART1 FALSE

  USART1 driver enable switch.

Functions

- void sd_lld_init (void)

  Low level serial driver initialization.
- void sd_lld_start (SerialDriver ∗sdp, const SerialConfig ∗config)

  Low level serial driver configuration and (re)start.
- void sd_lld_stop (SerialDriver ∗sdp)

  Low level serial driver stop.

9.76.1 Detailed Description

PLATFORM serial subsystem low level driver header.

9.77 hal_serial_usb.c File Reference

Serial over USB Driver code.

```
#include "hal.h"
```

Functions

- static void ibnotify (io_buffers_queue_t ∗bqp)

  Notification of empty buffer released into the input buffers queue.
- static void obnotify (io_buffers_queue_t ∗bqp)

  Notification of filled buffer inserted into the output buffers queue.
- void sduInit (void)

  Serial Driver initialization.
- void sduObjectInit (SerialUSBDriver ∗sdup)

  Initializes a generic full duplex driver object.
- void sduStart (SerialUSBDriver ∗sdup, const SerialUSBConfig ∗config)

  Configures and starts the driver.
- void sduStop (SerialUSBDriver ∗sdup)

  Stops the driver.
- void sduSuspendHookI (SerialUSBDriver ∗sdup)

  USB device suspend handler.
- void sduWakeupHookI (SerialUSBDriver ∗sdup)

  USB device wakeup handler.
- void sduConfigureHookI (SerialUSBDriver ∗sdup)

  USB device configured handler.
- bool sduRequestsHook (USBDriver ∗usbp)

Default requests hook.
- void sduSOFHookI (SerialUSBDriver ∗sdup)

  SOF handler.
- void sduDataTransmitted (USBDriver ∗usbp, usbep_t ep)

  Default data transmitted callback.
- void sduDataReceived (USBDriver ∗usbp, usbep_t ep)

  Default data received callback.
- void sduInterruptTransmitted (USBDriver ∗usbp, usbep_t ep)

  Default data received callback.

9.77.1 Detailed Description

Serial over USB Driver code.

9.78 hal_serial_usb.h File Reference

Serial over USB Driver macros and structures.

`#include "hal_usb_cdc.h"`

Data Structures

- struct SerialUSBConfig

  Serial over USB Driver configuration structure.
- struct SerialUSBDriverVMT

  `SerialDriver` virtual methods table.
- struct SerialUSBDriver

  Full duplex serial driver class.

Macros

- #define _serial_usb_driver_data

  `SerialDriver` specific data.
- #define _serial_usb_driver_methods _base_asynchronous_channel_methods

  `SerialUSBDriver` specific methods.

SERIAL_USB configuration options

- #define SERIAL_USB_BUFFERS_SIZE 256

  Serial over USB buffers size.
- #define SERIAL_USB_BUFFERS_NUMBER 2

  Serial over USB number of buffers.

Typedefs

- typedef struct SerialUSBDriver SerialUSBDriver

  Structure representing a serial over USB driver.

Enumerations

Functions

- void sduInit (void)

  Serial Driver initialization.
- void sduObjectInit (SerialUSBDriver ∗sdup)

  Initializes a generic full duplex driver object.
- void sduStart (SerialUSBDriver ∗sdup, const SerialUSBConfig ∗config)

  Configures and starts the driver.
- void sduStop (SerialUSBDriver ∗sdup)

  Stops the driver.
- void sduSuspendHookI (SerialUSBDriver ∗sdup)

  USB device suspend handler.
- void sduWakeupHookI (SerialUSBDriver ∗sdup)

  USB device wakeup handler.
- void sduConfigureHookI (SerialUSBDriver ∗sdup)

  USB device configured handler.
- bool sduRequestsHook (USBDriver ∗usbp)

  Default requests hook.
- void sduSOFHookI (SerialUSBDriver ∗sdup)

  SOF handler.
- void sduDataTransmitted (USBDriver ∗usbp, usbep_t ep)

  Default data transmitted callback.
- void sduDataReceived (USBDriver ∗usbp, usbep_t ep)

  Default data received callback.
- void sduInterruptTransmitted (USBDriver ∗usbp, usbep_t ep)

  Default data received callback.

9.78.1 Detailed Description

Serial over USB Driver macros and structures.

9.79 hal_spi.c File Reference

SPI Driver code.

`#include "hal.h"`

Functions

- void spiInit (void)

  SPI Driver initialization.
- void spiObjectInit (SPIDriver ∗spip)

  Initializes the standard part of a `SPIDriver` structure.
- void spiStart (SPIDriver ∗spip, const SPIConfig ∗config)

  Configures and activates the SPI peripheral.
- void spiStop (SPIDriver ∗spip)

  Deactivates the SPI peripheral.

- void spiSelect (SPIDriver ∗spip)
  Asserts the slave select signal and prepares for transfers.
- void spiUnselect (SPIDriver ∗spip)
  Deasserts the slave select signal.
- void spiStartIgnore (SPIDriver ∗spip, size_t n)
  Ignores data on the SPI bus.
- void spiStartExchange (SPIDriver ∗spip, size_t n, const void ∗txbuf, void ∗rxbuf)
  Exchanges data on the SPI bus.
- void spiStartSend (SPIDriver ∗spip, size_t n, const void ∗txbuf)
  Sends data over the SPI bus.
- void spiStartReceive (SPIDriver ∗spip, size_t n, void ∗rxbuf)
  Receives data from the SPI bus.
- void spiIgnore (SPIDriver ∗spip, size_t n)
  Ignores data on the SPI bus.
- void spiExchange (SPIDriver ∗spip, size_t n, const void ∗txbuf, void ∗rxbuf)
  Exchanges data on the SPI bus.
- void spiSend (SPIDriver ∗spip, size_t n, const void ∗txbuf)
  Sends data over the SPI bus.
- void spiReceive (SPIDriver ∗spip, size_t n, void ∗rxbuf)
  Receives data from the SPI bus.
- void spiAcquireBus (SPIDriver ∗spip)
  Gains exclusive access to the SPI bus.
- void spiReleaseBus (SPIDriver ∗spip)
  Releases exclusive access to the SPI bus.

9.79.1 Detailed Description

SPI Driver code.

9.80 hal_spi.h File Reference

SPI Driver macros and structures.

```
#include "hal_spi_lld.h"
```

Macros

SPI configuration options

- #define SPI_USE_WAIT TRUE
  Enables synchronous APIs.
- #define SPI_USE_MUTUAL_EXCLUSION TRUE
  Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.

Macro Functions

- #define spiSelectI(spip)
  Asserts the slave select signal and prepares for transfers.
- #define spiUnselectI(spip)
  Deasserts the slave select signal.
- #define spiStartIgnoreI(spip, n)

Ignores data on the SPI bus.
- #define spiStartExchangeI(spip, n, txbuf, rxbuf)

  Exchanges data on the SPI bus.
- #define spiStartSendI(spip, n, txbuf)

  Sends data over the SPI bus.
- #define spiStartReceiveI(spip, n, rxbuf)

  Receives data from the SPI bus.
- #define spiPolledExchange(spip, frame) spi_lld_polled_exchange(spip, frame)

  Exchanges one frame using a polled wait.

Low level driver helper macros

- #define _spi_wakeup_isr(spip)

  Wakes up the waiting thread.
- #define _spi_isr_code(spip)

  Common ISR code.

Enumerations

Functions

- void spiInit (void)

  SPI Driver initialization.
- void spiObjectInit (SPIDriver ∗spip)

  Initializes the standard part of a `SPIDriver` structure.
- void spiStart (SPIDriver ∗spip, const SPIConfig ∗config)

  Configures and activates the SPI peripheral.
- void spiStop (SPIDriver ∗spip)

  Deactivates the SPI peripheral.
- void spiSelect (SPIDriver ∗spip)

  Asserts the slave select signal and prepares for transfers.
- void spiUnselect (SPIDriver ∗spip)

  Deasserts the slave select signal.
- void spiStartIgnore (SPIDriver ∗spip, size_t n)

  Ignores data on the SPI bus.
- void spiStartExchange (SPIDriver ∗spip, size_t n, const void ∗txbuf, void ∗rxbuf)

  Exchanges data on the SPI bus.
- void spiStartSend (SPIDriver ∗spip, size_t n, const void ∗txbuf)

  Sends data over the SPI bus.
- void spiStartReceive (SPIDriver ∗spip, size_t n, void ∗rxbuf)

  Receives data from the SPI bus.
- void spiIgnore (SPIDriver ∗spip, size_t n)

  Ignores data on the SPI bus.
- void spiExchange (SPIDriver ∗spip, size_t n, const void ∗txbuf, void ∗rxbuf)

  Exchanges data on the SPI bus.
- void spiSend (SPIDriver ∗spip, size_t n, const void ∗txbuf)

  Sends data over the SPI bus.
- void spiReceive (SPIDriver ∗spip, size_t n, void ∗rxbuf)

  Receives data from the SPI bus.
- void spiAcquireBus (SPIDriver ∗spip)

  Gains exclusive access to the SPI bus.
- void spiReleaseBus (SPIDriver ∗spip)

  Releases exclusive access to the SPI bus.

9.80.1 Detailed Description

SPI Driver macros and structures.

9.81 hal_spi_lld.c File Reference

PLATFORM SPI subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void spi_lld_init (void)

  Low level SPI driver initialization.
- void spi_lld_start (SPIDriver ∗spip)

  Configures and activates the SPI peripheral.
- void spi_lld_stop (SPIDriver ∗spip)

  Deactivates the SPI peripheral.
- void spi_lld_select (SPIDriver ∗spip)

  Asserts the slave select signal and prepares for transfers.
- void spi_lld_unselect (SPIDriver ∗spip)

  Deasserts the slave select signal.
- void spi_lld_ignore (SPIDriver ∗spip, size_t n)

  Ignores data on the SPI bus.
- void spi_lld_exchange (SPIDriver ∗spip, size_t n, const void ∗txbuf, void ∗rxbuf)

  Exchanges data on the SPI bus.
- void spi_lld_send (SPIDriver ∗spip, size_t n, const void ∗txbuf)

  Sends data over the SPI bus.
- void spi_lld_receive (SPIDriver ∗spip, size_t n, void ∗rxbuf)

  Receives data from the SPI bus.
- uint16_t spi_lld_polled_exchange (SPIDriver ∗spip, uint16_t frame)

  Exchanges one frame using a polled wait.

Variables

- SPIDriver SPID1

  SPI1 driver identifier.

9.81.1 Detailed Description

PLATFORM SPI subsystem low level driver source.

9.82 hal_spi_lld.h File Reference

PLATFORM SPI subsystem low level driver header.

Data Structures

- struct SPIConfig

  Driver configuration structure.
- struct SPIDriver

  Structure representing an SPI driver.

Macros

PLATFORM configuration options

- #define PLATFORM_SPI_USE_SPI1 FALSE

  SPI1 driver enable switch.

Typedefs

- typedef struct SPIDriver SPIDriver

  Type of a structure representing an SPI driver.
- typedef void(∗ spicallback_t) (SPIDriver ∗spip)

  SPI notification callback type.

Functions

- void spi_lld_init (void)

  Low level SPI driver initialization.
- void spi_lld_start (SPIDriver ∗spip)

  Configures and activates the SPI peripheral.
- void spi_lld_stop (SPIDriver ∗spip)

  Deactivates the SPI peripheral.
- void spi_lld_select (SPIDriver ∗spip)

  Asserts the slave select signal and prepares for transfers.
- void spi_lld_unselect (SPIDriver ∗spip)

  Deasserts the slave select signal.
- void spi_lld_ignore (SPIDriver ∗spip, size_t n)

  Ignores data on the SPI bus.
- void spi_lld_exchange (SPIDriver ∗spip, size_t n, const void ∗txbuf, void ∗rxbuf)

  Exchanges data on the SPI bus.
- void spi_lld_send (SPIDriver ∗spip, size_t n, const void ∗txbuf)

  Sends data over the SPI bus.
- void spi_lld_receive (SPIDriver ∗spip, size_t n, void ∗rxbuf)

  Receives data from the SPI bus.
- uint16_t spi_lld_polled_exchange (SPIDriver ∗spip, uint16_t frame)

  Exchanges one frame using a polled wait.

9.82.1 Detailed Description

PLATFORM SPI subsystem low level driver header.

9.83 hal_st.c File Reference

ST Driver code.

```
#include "hal.h"
```

Functions

- void stInit (void)

  ST Driver initialization.
- void stStartAlarm (systime_t abstime)

  Starts the alarm.
- void stStopAlarm (void)

  Stops the alarm interrupt.
- void stSetAlarm (systime_t abstime)

  Sets the alarm time.
- systime_t stGetAlarm (void)

  Returns the current alarm time.

9.83.1 Detailed Description

ST Driver code.

9.84 hal_st.h File Reference

ST Driver macros and structures.

```
#include "hal_st_lld.h"
```

Macros

Macro Functions

- #define stGetCounter() st_lld_get_counter()

  Returns the time counter value.
- #define stIsAlarmActive() st_lld_is_alarm_active()

  Determines if the alarm is active.

Functions

- void stInit (void)

  ST Driver initialization.
- void stStartAlarm (systime_t abstime)

  Starts the alarm.
- void stStopAlarm (void)

  Stops the alarm interrupt.
- void stSetAlarm (systime_t abstime)

  Sets the alarm time.
- systime_t stGetAlarm (void)

  Returns the current alarm time.

9.84.1 Detailed Description

ST Driver macros and structures.

This header is designed to be include-able without having to include other files from the HAL.

9.85 hal_st_lld.c File Reference

PLATFORM ST subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void st_lld_init (void)

  Low level ST driver initialization.

9.85.1 Detailed Description

PLATFORM ST subsystem low level driver source.

9.86 hal_st_lld.h File Reference

PLATFORM ST subsystem low level driver header.

Functions

- void st_lld_init (void)

  Low level ST driver initialization.
- static systime_t st_lld_get_counter (void)

  Returns the time counter value.
- static void st_lld_start_alarm (systime_t abstime)

  Starts the alarm.
- static void st_lld_stop_alarm (void)

  Stops the alarm interrupt.
- static void st_lld_set_alarm (systime_t abstime)

  Sets the alarm time.
- static systime_t st_lld_get_alarm (void)

  Returns the current alarm time.
- static bool st_lld_is_alarm_active (void)

  Determines if the alarm is active.

9.86.1 Detailed Description

PLATFORM ST subsystem low level driver header.

This header is designed to be include-able without having to include other files from the HAL.

9.87 hal_streams.h File Reference

Data streams.

Data Structures

- struct BaseSequentialStreamVMT

    `BaseSequentialStream` virtual methods table.
- struct BaseSequentialStream

    Base stream class.

Macros

- #define _base_sequential_stream_methods

    BaseSequentialStream specific methods.
- #define _base_sequential_stream_data

    `BaseSequentialStream` specific data.

Streams return codes

- #define **STM_OK** MSG_OK
- #define **STM_TIMEOUT** MSG_TIMEOUT
- #define **STM_RESET** MSG_RESET

Macro Functions (BaseSequentialStream)

- #define streamWrite(ip, bp, n) ((ip)->vmt->write(ip, bp, n))

    Sequential Stream write.
- #define streamRead(ip, bp, n) ((ip)->vmt->read(ip, bp, n))

    Sequential Stream read.
- #define streamPut(ip, b) ((ip)->vmt->put(ip, b))

    Sequential Stream blocking byte write.
- #define streamGet(ip) ((ip)->vmt->get(ip))

    Sequential Stream blocking byte read.

9.87.1 Detailed Description

Data streams.

This header defines abstract interfaces useful to access generic data streams in a standardized way.

9.88 hal_uart.c File Reference

UART Driver code.

```
#include "hal.h"
```

Functions

- void uartInit (void)

    UART Driver initialization.
- void uartObjectInit (UARTDriver ∗uartp)

    Initializes the standard part of a `UARTDriver` structure.
- void uartStart (UARTDriver ∗uartp, const UARTConfig ∗config)

    Configures and activates the UART peripheral.
- void uartStop (UARTDriver ∗uartp)

    Deactivates the UART peripheral.
- void uartStartSend (UARTDriver ∗uartp, size_t n, const void ∗txbuf)

    Starts a transmission on the UART peripheral.
- void uartStartSendI (UARTDriver ∗uartp, size_t n, const void ∗txbuf)

    Starts a transmission on the UART peripheral.
- size_t uartStopSend (UARTDriver ∗uartp)

    Stops any ongoing transmission.
- size_t uartStopSendI (UARTDriver ∗uartp)

    Stops any ongoing transmission.
- void uartStartReceive (UARTDriver ∗uartp, size_t n, void ∗rxbuf)

    Starts a receive operation on the UART peripheral.
- void uartStartReceiveI (UARTDriver ∗uartp, size_t n, void ∗rxbuf)

    Starts a receive operation on the UART peripheral.
- size_t uartStopReceive (UARTDriver ∗uartp)

    Stops any ongoing receive operation.
- size_t uartStopReceiveI (UARTDriver ∗uartp)

    Stops any ongoing receive operation.
- msg_t uartSendTimeout (UARTDriver ∗uartp, size_t ∗np, const void ∗txbuf, systime_t timeout)

    Performs a transmission on the UART peripheral.
- msg_t uartSendFullTimeout (UARTDriver ∗uartp, size_t ∗np, const void ∗txbuf, systime_t timeout)

    Performs a transmission on the UART peripheral.
- msg_t uartReceiveTimeout (UARTDriver ∗uartp, size_t ∗np, void ∗rxbuf, systime_t timeout)

    Performs a receive operation on the UART peripheral.
- void uartAcquireBus (UARTDriver ∗uartp)

    Gains exclusive access to the UART bus.
- void uartReleaseBus (UARTDriver ∗uartp)

    Releases exclusive access to the UART bus.

9.88.1 Detailed Description

UART Driver code.

9.89 hal_uart.h File Reference

UART Driver macros and structures.

```
#include "hal_uart_lld.h"
```

Macros

UART status flags

- #define UART_NO_ERROR 0
  No pending conditions.
- #define UART_PARITY_ERROR 4
  Parity error happened.
- #define UART_FRAMING_ERROR 8
  Framing error happened.
- #define UART_OVERRUN_ERROR 16
  Overflow happened.
- #define UART_NOISE_ERROR 32
  Noise on the line.
- #define UART_BREAK_DETECTED 64
  Break detected.

UART configuration options

- #define UART_USE_WAIT FALSE
  Enables synchronous APIs.
- #define UART_USE_MUTUAL_EXCLUSION FALSE
  Enables the `uartAcquireBus()` and `uartReleaseBus()` APIs.

Low level driver helper macros

- #define _uart_wakeup_tx1_isr(uartp)
  Wakes up the waiting thread in case of early TX complete.
- #define _uart_wakeup_tx2_isr(uartp)
  Wakes up the waiting thread in case of late TX complete.
- #define _uart_wakeup_rx_complete_isr(uartp)
  Wakes up the waiting thread in case of RX complete.
- #define _uart_wakeup_rx_error_isr(uartp)
  Wakes up the waiting thread in case of RX error.
- #define _uart_wakeup_rx_timeout_isr(uartp)
  Wakes up the waiting thread in case of RX timeout.
- #define _uart_tx1_isr_code(uartp)
  Common ISR code for early TX.
- #define _uart_tx2_isr_code(uartp)
  Common ISR code for late TX.
- #define _uart_rx_complete_isr_code(uartp)
  Common ISR code for RX complete.
- #define _uart_rx_error_isr_code(uartp, errors)
  Common ISR code for RX error.
- #define _uart_rx_idle_code(uartp)
  Common ISR code for RX on idle.
- #define _uart_timeout_isr_code(uartp)
  Timeout ISR code for receiver.

Enumerations

Functions

- void uartInit (void)
  UART Driver initialization.
- void uartObjectInit (UARTDriver ∗uartp)
  Initializes the standard part of a `UARTDriver` structure.

- void uartStart (UARTDriver ∗uartp, const UARTConfig ∗config)
    Configures and activates the UART peripheral.
- void uartStop (UARTDriver ∗uartp)
    Deactivates the UART peripheral.
- void uartStartSend (UARTDriver ∗uartp, size_t n, const void ∗txbuf)
    Starts a transmission on the UART peripheral.
- void uartStartSendI (UARTDriver ∗uartp, size_t n, const void ∗txbuf)
    Starts a transmission on the UART peripheral.
- size_t uartStopSend (UARTDriver ∗uartp)
    Stops any ongoing transmission.
- size_t uartStopSendI (UARTDriver ∗uartp)
    Stops any ongoing transmission.
- void uartStartReceive (UARTDriver ∗uartp, size_t n, void ∗rxbuf)
    Starts a receive operation on the UART peripheral.
- void uartStartReceiveI (UARTDriver ∗uartp, size_t n, void ∗rxbuf)
    Starts a receive operation on the UART peripheral.
- size_t uartStopReceive (UARTDriver ∗uartp)
    Stops any ongoing receive operation.
- size_t uartStopReceiveI (UARTDriver ∗uartp)
    Stops any ongoing receive operation.
- msg_t uartSendTimeout (UARTDriver ∗uartp, size_t ∗np, const void ∗txbuf, systime_t timeout)
    Performs a transmission on the UART peripheral.
- msg_t uartSendFullTimeout (UARTDriver ∗uartp, size_t ∗np, const void ∗txbuf, systime_t timeout)
    Performs a transmission on the UART peripheral.
- msg_t uartReceiveTimeout (UARTDriver ∗uartp, size_t ∗np, void ∗rxbuf, systime_t timeout)
    Performs a receive operation on the UART peripheral.
- void uartAcquireBus (UARTDriver ∗uartp)
    Gains exclusive access to the UART bus.
- void uartReleaseBus (UARTDriver ∗uartp)
    Releases exclusive access to the UART bus.

9.89.1 Detailed Description

UART Driver macros and structures.

9.90 hal_uart_lld.c File Reference

PLATFORM UART subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void uart_lld_init (void)
    Low level UART driver initialization.
- void uart_lld_start (UARTDriver ∗uartp)
    Configures and activates the UART peripheral.
- void uart_lld_stop (UARTDriver ∗uartp)
    Deactivates the UART peripheral.

- void uart_lld_start_send (UARTDriver ∗uartp, size_t n, const void ∗txbuf)

  Starts a transmission on the UART peripheral.
- size_t uart_lld_stop_send (UARTDriver ∗uartp)

  Stops any ongoing transmission.
- void uart_lld_start_receive (UARTDriver ∗uartp, size_t n, void ∗rxbuf)

  Starts a receive operation on the UART peripheral.
- size_t uart_lld_stop_receive (UARTDriver ∗uartp)

  Stops any ongoing receive operation.

Variables

- UARTDriver UARTD1

  UART1 driver identifier.

9.90.1 Detailed Description

PLATFORM UART subsystem low level driver source.

9.91 hal_uart_lld.h File Reference

PLATFORM UART subsystem low level driver header.

Data Structures

- struct UARTConfig

  Driver configuration structure.
- struct UARTDriver

  Structure representing an UART driver.

Macros

PLATFORM configuration options

- #define PLATFORM_UART_USE_UART1 FALSE

  UART driver enable switch.

Typedefs

- typedef uint32_t uartflags_t

  UART driver condition flags type.
- typedef struct UARTDriver UARTDriver

  Type of structure representing an UART driver.
- typedef void(∗ uartcb_t) (UARTDriver ∗uartp)

  Generic UART notification callback type.
- typedef void(∗ uartccb_t) (UARTDriver ∗uartp, uint16_t c)

  Character received UART notification callback type.
- typedef void(∗ uartecb_t) (UARTDriver ∗uartp, uartflags_t e)

  Receive error UART notification callback type.

Functions

- void uart_lld_init (void)

  Low level UART driver initialization.
- void uart_lld_start (UARTDriver ∗uartp)

  Configures and activates the UART peripheral.
- void uart_lld_stop (UARTDriver ∗uartp)

  Deactivates the UART peripheral.
- void uart_lld_start_send (UARTDriver ∗uartp, size_t n, const void ∗txbuf)

  Starts a transmission on the UART peripheral.
- size_t uart_lld_stop_send (UARTDriver ∗uartp)

  Stops any ongoing transmission.
- void uart_lld_start_receive (UARTDriver ∗uartp, size_t n, void ∗rxbuf)

  Starts a receive operation on the UART peripheral.
- size_t uart_lld_stop_receive (UARTDriver ∗uartp)

  Stops any ongoing receive operation.

9.91.1 Detailed Description

PLATFORM UART subsystem low level driver header.

9.92 hal_usb.c File Reference

USB Driver code.

```
#include <string.h>
#include "hal.h"
```

Functions

- static void set_address (USBDriver ∗usbp)

  SET ADDRESS transaction callback.
- static bool default_handler (USBDriver ∗usbp)

  Standard requests handler.
- void usbInit (void)

  USB Driver initialization.
- void usbObjectInit (USBDriver ∗usbp)

  Initializes the standard part of a `USBDriver` structure.
- void usbStart (USBDriver ∗usbp, const USBConfig ∗config)

  Configures and activates the USB peripheral.
- void usbStop (USBDriver ∗usbp)

  Deactivates the USB peripheral.
- void usbInitEndpointI (USBDriver ∗usbp, usbep_t ep, const USBEndpointConfig ∗epcp)

  Enables an endpoint.
- void usbDisableEndpointsI (USBDriver ∗usbp)

  Disables all the active endpoints.
- void usbStartReceiveI (USBDriver ∗usbp, usbep_t ep, uint8_t ∗buf, size_t n)

  Starts a receive transaction on an OUT endpoint.
- void usbStartTransmitI (USBDriver ∗usbp, usbep_t ep, const uint8_t ∗buf, size_t n)

Starts a transmit transaction on an IN endpoint.
- msg_t usbReceive (USBDriver ∗usbp, usbep_t ep, uint8_t ∗buf, size_t n)

  Performs a receive transaction on an OUT endpoint.
- msg_t usbTransmit (USBDriver ∗usbp, usbep_t ep, const uint8_t ∗buf, size_t n)

  Performs a transmit transaction on an IN endpoint.
- bool usbStallReceiveI (USBDriver ∗usbp, usbep_t ep)

  Stalls an OUT endpoint.
- bool usbStallTransmitI (USBDriver ∗usbp, usbep_t ep)

  Stalls an IN endpoint.
- void _usb_reset (USBDriver ∗usbp)

  USB reset routine.
- void _usb_suspend (USBDriver ∗usbp)

  USB suspend routine.
- void _usb_wakeup (USBDriver ∗usbp)

  USB wake-up routine.
- void _usb_ep0setup (USBDriver ∗usbp, usbep_t ep)

  Default EP0 SETUP callback.
- void _usb_ep0in (USBDriver ∗usbp, usbep_t ep)

  Default EP0 IN callback.
- void _usb_ep0out (USBDriver ∗usbp, usbep_t ep)

  Default EP0 OUT callback.

9.92.1 Detailed Description

USB Driver code.

9.93 hal_usb.h File Reference

USB Driver macros and structures.

```
#include "hal_usb_lld.h"
```

Data Structures

- struct USBDescriptor

  Type of an USB descriptor.

Macros

- #define USB_USE_WAIT FALSE

  Enables synchronous APIs.

Helper macros for USB descriptors

- #define USB_DESC_INDEX(i) ((uint8_t)(i))

  Helper macro for index values into descriptor strings.
- #define USB_DESC_BYTE(b) ((uint8_t)(b))

  Helper macro for byte values into descriptor strings.
- #define USB_DESC_WORD(w)

  Helper macro for word values into descriptor strings.

- #define USB_DESC_BCD(bcd)
  Helper macro for BCD values into descriptor strings.
- #define **USB_DESC_DEVICE_SIZE** 18U
- #define USB_DESC_DEVICE(bcdUSB, bDeviceClass, bDeviceSubClass, bDeviceProtocol, b↩ MaxPacketSize, idVendor, idProduct, bcdDevice, iManufacturer, iProduct, iSerialNumber, bNum↩ Configurations)
  Device Descriptor helper macro.
- #define USB_DESC_CONFIGURATION_SIZE 9U
  Configuration Descriptor size.
- #define USB_DESC_CONFIGURATION(wTotalLength, bNumInterfaces, bConfigurationValue, i↩ Configuration, bmAttributes, bMaxPower)
  Configuration Descriptor helper macro.
- #define USB_DESC_INTERFACE_SIZE 9U
  Interface Descriptor size.
- #define USB_DESC_INTERFACE(bInterfaceNumber, bAlternateSetting, bNumEndpoints, bInterface↩ Class, bInterfaceSubClass, bInterfaceProtocol, iInterface)
  Interface Descriptor helper macro.
- #define USB_DESC_INTERFACE_ASSOCIATION_SIZE 8U
  Interface Association Descriptor size.
- #define USB_DESC_INTERFACE_ASSOCIATION(bFirstInterface, bInterfaceCount, bFunctionClass, b↩ FunctionSubClass, bFunctionProcotol, iInterface)
  Interface Association Descriptor helper macro.
- #define USB_DESC_ENDPOINT_SIZE 7U
  Endpoint Descriptor size.
- #define USB_DESC_ENDPOINT(bEndpointAddress, bmAttributes, wMaxPacketSize, bInterval)
  Endpoint Descriptor helper macro.

Endpoint types and settings

- #define USB_EP_MODE_TYPE 0x0003U
- #define USB_EP_MODE_TYPE_CTRL 0x0000U
- #define USB_EP_MODE_TYPE_ISOC 0x0001U
- #define USB_EP_MODE_TYPE_BULK 0x0002U
- #define USB_EP_MODE_TYPE_INTR 0x0003U

Macro Functions

- #define usbGetDriverStateI(usbp) ((usbp)->state)
  Returns the driver state.
- #define usbConnectBus(usbp) usb_lld_connect_bus(usbp)
  Connects the USB device.
- #define usbDisconnectBus(usbp) usb_lld_disconnect_bus(usbp)
  Disconnect the USB device.
- #define usbGetFrameNumberX(usbp) usb_lld_get_frame_number(usbp)
  Returns the current frame number.
- #define usbGetTransmitStatusI(usbp, ep) (((usbp)->transmitting & (uint16_t)((unsigned)1U << (un-signed)(ep))) != 0U)
  Returns the status of an IN endpoint.
- #define usbGetReceiveStatusI(usbp, ep) (((usbp)->receiving & (uint16_t)((unsigned)1U << (un-signed)(ep))) != 0U)
  Returns the status of an OUT endpoint.
- #define usbGetReceiveTransactionSizeX(usbp, ep) usb_lld_get_transaction_size(usbp, ep)
  Returns the exact size of a receive transaction.
- #define usbSetupTransfer(usbp, buf, n, endcb)
  Request transfer setup.
- #define usbReadSetup(usbp, ep, buf) usb_lld_read_setup(usbp, ep, buf)
  Reads a setup packet from the dedicated packet buffer.

Low level driver helper macros

- #define _usb_isr_invoke_event_cb(usbp, evt)

  Common ISR code, usb event callback.
- #define _usb_isr_invoke_sof_cb(usbp)

  Common ISR code, SOF callback.
- #define _usb_isr_invoke_setup_cb(usbp, ep)

  Common ISR code, setup packet callback.
- #define _usb_isr_invoke_in_cb(usbp, ep)

  Common ISR code, IN endpoint callback.
- #define _usb_isr_invoke_out_cb(usbp, ep)

  Common ISR code, OUT endpoint event.

Typedefs

- typedef struct USBDriver USBDriver

  Type of a structure representing an USB driver.
- typedef uint8_t usbep_t

  Type of an endpoint identifier.
- typedef void(∗ usbcallback_t) (USBDriver ∗usbp)

  Type of an USB generic notification callback.
- typedef void(∗ usbepcallback_t) (USBDriver ∗usbp, usbep_t ep)

  Type of an USB endpoint callback.
- typedef void(∗ usbeventcb_t) (USBDriver ∗usbp, usbevent_t event)

  Type of an USB event notification callback.
- typedef bool(∗ usbreqhandler_t) (USBDriver ∗usbp)

  Type of a requests handler callback.
- typedef const USBDescriptor ∗(∗ usbgetdescriptor_t) (USBDriver ∗usbp, uint8_t dtype, uint8_t dindex, uint16_t lang)

  Type of an USB descriptor-retrieving callback.

Enumerations

Functions

- void usbInit (void)

  USB Driver initialization.
- void usbObjectInit (USBDriver ∗usbp)

  Initializes the standard part of a `USBDriver` structure.
- void usbStart (USBDriver ∗usbp, const USBConfig ∗config)

  Configures and activates the USB peripheral.
- void usbStop (USBDriver ∗usbp)

  Deactivates the USB peripheral.
- void usbInitEndpointI (USBDriver ∗usbp, usbep_t ep, const USBEndpointConfig ∗epcp)

  Enables an endpoint.
- void usbDisableEndpointsI (USBDriver ∗usbp)

  Disables all the active endpoints.
- void usbStartReceiveI (USBDriver ∗usbp, usbep_t ep, uint8_t ∗buf, size_t n)

  Starts a receive transaction on an OUT endpoint.
- void usbStartTransmitI (USBDriver ∗usbp, usbep_t ep, const uint8_t ∗buf, size_t n)

  Starts a transmit transaction on an IN endpoint.
- msg_t usbReceive (USBDriver ∗usbp, usbep_t ep, uint8_t ∗buf, size_t n)

  Performs a receive transaction on an OUT endpoint.
- msg_t usbTransmit (USBDriver ∗usbp, usbep_t ep, const uint8_t ∗buf, size_t n)

Performs a transmit transaction on an IN endpoint.
- bool usbStallReceiveI (USBDriver ∗usbp, usbep_t ep)

  Stalls an OUT endpoint.
- bool usbStallTransmitI (USBDriver ∗usbp, usbep_t ep)

  Stalls an IN endpoint.
- void _usb_reset (USBDriver ∗usbp)

  USB reset routine.
- void _usb_suspend (USBDriver ∗usbp)

  USB suspend routine.
- void _usb_wakeup (USBDriver ∗usbp)

  USB wake-up routine.
- void _usb_ep0setup (USBDriver ∗usbp, usbep_t ep)

  Default EP0 SETUP callback.
- void _usb_ep0in (USBDriver ∗usbp, usbep_t ep)

  Default EP0 IN callback.
- void _usb_ep0out (USBDriver ∗usbp, usbep_t ep)

  Default EP0 OUT callback.

9.93.1 Detailed Description

USB Driver macros and structures.

9.94 hal_usb_cdc.h File Reference

USB CDC macros and structures.

Data Structures

- struct cdc_linecoding_t

  Type of Line Coding structure.

Macros

CDC specific messages.

- #define **CDC_SEND_ENCAPSULATED_COMMAND** 0x00U
- #define **CDC_GET_ENCAPSULATED_RESPONSE** 0x01U
- #define **CDC_SET_COMM_FEATURE** 0x02U
- #define **CDC_GET_COMM_FEATURE** 0x03U
- #define **CDC_CLEAR_COMM_FEATURE** 0x04U
- #define **CDC_SET_AUX_LINE_STATE** 0x10U
- #define **CDC_SET_HOOK_STATE** 0x11U
- #define **CDC_PULSE_SETUP** 0x12U
- #define **CDC_SEND_PULSE** 0x13U
- #define **CDC_SET_PULSE_TIME** 0x14U
- #define **CDC_RING_AUX_JACK** 0x15U
- #define **CDC_SET_LINE_CODING** 0x20U
- #define **CDC_GET_LINE_CODING** 0x21U
- #define **CDC_SET_CONTROL_LINE_STATE** 0x22U
- #define **CDC_SEND_BREAK** 0x23U
- #define **CDC_SET_RINGER_PARMS** 0x30U
- #define **CDC_GET_RINGER_PARMS** 0x31U
- #define **CDC_SET_OPERATION_PARMS** 0x32U

- #define **CDC_GET_OPERATION_PARMS** 0x33U

CDC classes

- #define **CDC_COMMUNICATION_INTERFACE_CLASS** 0x02U
- #define **CDC_DATA_INTERFACE_CLASS** 0x0AU

CDC subclasses

- #define **CDC_ABSTRACT_CONTROL_MODEL** 0x02U

CDC descriptors

- #define **CDC_CS_INTERFACE** 0x24U

CDC subdescriptors

- #define **CDC_HEADER** 0x00U
- #define **CDC_CALL_MANAGEMENT** 0x01U
- #define **CDC_ABSTRACT_CONTROL_MANAGEMENT** 0x02U
- #define **CDC_UNION** 0x06U

Line Control bit definitions.

- #define **LC_STOP_1** 0U
- #define **LC_STOP_1P5** 1U
- #define **LC_STOP_2** 2U
- #define **LC_PARITY_NONE** 0U
- #define **LC_PARITY_ODD** 1U
- #define **LC_PARITY_EVEN** 2U
- #define **LC_PARITY_MARK** 3U
- #define **LC_PARITY_SPACE** 4U

9.94.1 Detailed Description

USB CDC macros and structures.

9.95 hal_usb_lld.c File Reference

PLATFORM USB subsystem low level driver source.

```
#include "hal.h"
```

Functions

- void usb_lld_init (void)

  Low level USB driver initialization.
- void usb_lld_start (USBDriver ∗usbp)

  Configures and activates the USB peripheral.
- void usb_lld_stop (USBDriver ∗usbp)

  Deactivates the USB peripheral.
- void usb_lld_reset (USBDriver ∗usbp)

  USB low level reset routine.
- void usb_lld_set_address (USBDriver ∗usbp)

Sets the USB address.
- void usb_lld_init_endpoint (USBDriver ∗usbp, usbep_t ep)

  Enables an endpoint.
- void usb_lld_disable_endpoints (USBDriver ∗usbp)

  Disables all the active endpoints except the endpoint zero.
- usbepstatus_t usb_lld_get_status_out (USBDriver ∗usbp, usbep_t ep)

  Returns the status of an OUT endpoint.
- usbepstatus_t usb_lld_get_status_in (USBDriver ∗usbp, usbep_t ep)

  Returns the status of an IN endpoint.
- void usb_lld_read_setup (USBDriver ∗usbp, usbep_t ep, uint8_t ∗buf)

  Reads a setup packet from the dedicated packet buffer.
- void usb_lld_prepare_receive (USBDriver ∗usbp, usbep_t ep)

  Prepares for a receive operation.
- void usb_lld_prepare_transmit (USBDriver ∗usbp, usbep_t ep)

  Prepares for a transmit operation.
- void usb_lld_start_out (USBDriver ∗usbp, usbep_t ep)

  Starts a receive operation on an OUT endpoint.
- void usb_lld_start_in (USBDriver ∗usbp, usbep_t ep)

  Starts a transmit operation on an IN endpoint.
- void usb_lld_stall_out (USBDriver ∗usbp, usbep_t ep)

  Brings an OUT endpoint in the stalled state.
- void usb_lld_stall_in (USBDriver ∗usbp, usbep_t ep)

  Brings an IN endpoint in the stalled state.
- void usb_lld_clear_out (USBDriver ∗usbp, usbep_t ep)

  Brings an OUT endpoint in the active state.
- void usb_lld_clear_in (USBDriver ∗usbp, usbep_t ep)

  Brings an IN endpoint in the active state.

Variables

- USBDriver USBD1

  USB1 driver identifier.
- union {
  USBInEndpointState in
  IN EP0 state.
  USBOutEndpointState out
  OUT EP0 state.
  } ep0_state

  EP0 state.
- static const USBEndpointConfig ep0config

  EP0 initialization structure.

9.95.1 Detailed Description

PLATFORM USB subsystem low level driver source.

9.95.2 Variable Documentation

9.95.2.1 USBInEndpointState in

IN EP0 state.

9.95.2.2 USBOutEndpointState out

OUT EP0 state.

9.96 hal_usb_lld.h File Reference

PLATFORM USB subsystem low level driver header.

Data Structures

- struct USBInEndpointState

  Type of an IN endpoint state structure.
- struct USBOutEndpointState

  Type of an OUT endpoint state structure.
- struct USBEndpointConfig

  Type of an USB endpoint configuration structure.
- struct USBConfig

  Type of an USB driver configuration structure.
- struct USBDriver

  Structure representing an USB driver.

Macros

- #define USB_MAX_ENDPOINTS 4

  Maximum endpoint address.
- #define USB_EP0_STATUS_STAGE USB_EP0_STATUS_STAGE_SW

  Status stage handling method.
- #define USB_SET_ADDRESS_MODE USB_LATE_SET_ADDRESS

  The address can be changed immediately upon packet reception.
- #define USB_SET_ADDRESS_ACK_HANDLING USB_SET_ADDRESS_ACK_SW

  Method for set address acknowledge.
- #define usb_lld_get_frame_number(usbp) 0

  Returns the current frame number.
- #define usb_lld_get_transaction_size(usbp, ep) ((usbp)->epc[ep]->out_state->rxcnt)

  Returns the exact size of a receive transaction.
- #define usb_lld_connect_bus(usbp)

  Connects the USB device.
- #define usb_lld_disconnect_bus(usbp)

  Disconnect the USB device.

PLATFORM configuration options

- #define PLATFORM_USB_USE_USB1 FALSE

  USB driver enable switch.

Functions

- void usb_lld_init (void)

    Low level USB driver initialization.
- void usb_lld_start (USBDriver ∗usbp)

    Configures and activates the USB peripheral.
- void usb_lld_stop (USBDriver ∗usbp)

    Deactivates the USB peripheral.
- void usb_lld_reset (USBDriver ∗usbp)

    USB low level reset routine.
- void usb_lld_set_address (USBDriver ∗usbp)

    Sets the USB address.
- void usb_lld_init_endpoint (USBDriver ∗usbp, usbep_t ep)

    Enables an endpoint.
- void usb_lld_disable_endpoints (USBDriver ∗usbp)

    Disables all the active endpoints except the endpoint zero.
- usbepstatus_t usb_lld_get_status_in (USBDriver ∗usbp, usbep_t ep)

    Returns the status of an IN endpoint.
- usbepstatus_t usb_lld_get_status_out (USBDriver ∗usbp, usbep_t ep)

    Returns the status of an OUT endpoint.
- void usb_lld_read_setup (USBDriver ∗usbp, usbep_t ep, uint8_t ∗buf)

    Reads a setup packet from the dedicated packet buffer.
- void usb_lld_prepare_receive (USBDriver ∗usbp, usbep_t ep)

    Prepares for a receive operation.
- void usb_lld_prepare_transmit (USBDriver ∗usbp, usbep_t ep)

    Prepares for a transmit operation.
- void usb_lld_start_out (USBDriver ∗usbp, usbep_t ep)

    Starts a receive operation on an OUT endpoint.
- void usb_lld_start_in (USBDriver ∗usbp, usbep_t ep)

    Starts a transmit operation on an IN endpoint.
- void usb_lld_stall_out (USBDriver ∗usbp, usbep_t ep)

    Brings an OUT endpoint in the stalled state.
- void usb_lld_stall_in (USBDriver ∗usbp, usbep_t ep)

    Brings an IN endpoint in the stalled state.
- void usb_lld_clear_out (USBDriver ∗usbp, usbep_t ep)

    Brings an OUT endpoint in the active state.
- void usb_lld_clear_in (USBDriver ∗usbp, usbep_t ep)

    Brings an IN endpoint in the active state.

9.96.1 Detailed Description

PLATFORM USB subsystem low level driver header.

9.97 hal_wdg.c File Reference

WDG Driver code.

```
#include "hal.h"
```

Functions

- void wdgInit (void)

  WDG Driver initialization.
- void wdgStart (WDGDriver ∗wdgp, const WDGConfig ∗config)

  Configures and activates the WDG peripheral.
- void wdgStop (WDGDriver ∗wdgp)

  Deactivates the WDG peripheral.
- void wdgReset (WDGDriver ∗wdgp)

  Resets WDG's counter.

9.97.1 Detailed Description

WDG Driver code.

9.98 hal_wdg.h File Reference

WDG Driver macros and structures.

```
#include "hal_wdg_lld.h"
```

Macros

- #define wdgResetI(wdgp) wdg_lld_reset(wdgp)

  Resets WDG's counter.

Enumerations

Functions

- void wdgInit (void)

  WDG Driver initialization.
- void wdgStart (WDGDriver ∗wdgp, const WDGConfig ∗config)

  Configures and activates the WDG peripheral.
- void wdgStop (WDGDriver ∗wdgp)

  Deactivates the WDG peripheral.
- void wdgReset (WDGDriver ∗wdgp)

  Resets WDG's counter.

9.98.1 Detailed Description

WDG Driver macros and structures.

9.99 halconf.h File Reference

HAL configuration header.

```
#include "mcuconf.h"
```

Macros

Drivers enable switches

- #define HAL_USE_PAL TRUE
  Enables the PAL subsystem.
- #define HAL_USE_ADC TRUE
  Enables the ADC subsystem.
- #define HAL_USE_CAN TRUE
  Enables the CAN subsystem.
- #define HAL_USE_DAC FALSE
  Enables the DAC subsystem.
- #define HAL_USE_EXT TRUE
  Enables the EXT subsystem.
- #define HAL_USE_GPT TRUE
  Enables the GPT subsystem.
- #define HAL_USE_I2C TRUE
  Enables the I2C subsystem.
- #define HAL_USE_I2S TRUE
  Enables the I2S subsystem.
- #define HAL_USE_ICU TRUE
  Enables the ICU subsystem.
- #define HAL_USE_MAC TRUE
  Enables the MAC subsystem.
- #define HAL_USE_MMC_SPI TRUE
  Enables the MMC_SPI subsystem.
- #define HAL_USE_PWM TRUE
  Enables the PWM subsystem.
- #define HAL_USE_QSPI TRUE
  Enables the QSPI subsystem.
- #define HAL_USE_RTC TRUE
  Enables the RTC subsystem.
- #define HAL_USE_SDC TRUE
  Enables the SDC subsystem.
- #define HAL_USE_SERIAL TRUE
  Enables the SERIAL subsystem.
- #define HAL_USE_SERIAL_USB TRUE
  Enables the SERIAL over USB subsystem.
- #define HAL_USE_SPI TRUE
  Enables the SPI subsystem.
- #define HAL_USE_UART TRUE
  Enables the UART subsystem.
- #define HAL_USE_USB TRUE
  Enables the USB subsystem.
- #define HAL_USE_WDG TRUE
  Enables the WDG subsystem.

ADC driver related setting

- #define ADC_USE_WAIT TRUE
  Enables synchronous APIs.
- #define ADC_USE_MUTUAL_EXCLUSION TRUE
  Enables the `adcAcquireBus()` and `adcReleaseBus()` APIs.

CAN driver related setting

- #define CAN_USE_SLEEP_MODE TRUE
  Sleep mode related APIs inclusion switch.

I2C driver related setting

- #define I2C_USE_MUTUAL_EXCLUSION TRUE
  Enables the mutual exclusion APIs on the I2C bus.

MAC driver related setting

- #define MAC_USE_ZERO_COPY TRUE
  Enables an event sources for incoming packets.
- #define MAC_USE_EVENTS TRUE
  Enables an event sources for incoming packets.

MMC_SPI driver related setting

- #define MMC_NICE_WAITING TRUE
  Delays insertions.

SDC driver related setting

- #define SDC_INIT_RETRY 100
  Number of initialization attempts before rejecting the card.
- #define SDC_MMC_SUPPORT TRUE
  Include support for MMC cards.
- #define SDC_NICE_WAITING TRUE
  Delays insertions.

SERIAL driver related setting

- #define SERIAL_DEFAULT_BITRATE 38400
  Default bit rate.
- #define SERIAL_BUFFERS_SIZE 16
  Serial buffers size.

SERIAL_USB driver related setting

- #define SERIAL_USB_BUFFERS_SIZE 256
  Serial over USB buffers size.
- #define SERIAL_USB_BUFFERS_NUMBER 2
  Serial over USB number of buffers.

SPI driver related setting

- #define SPI_USE_WAIT TRUE
  Enables synchronous APIs.
- #define SPI_USE_MUTUAL_EXCLUSION TRUE
  Enables the `spiAcquireBus()` and `spiReleaseBus()` APIs.

UART driver related setting

- #define UART_USE_WAIT TRUE
  Enables synchronous APIs.
- #define UART_USE_MUTUAL_EXCLUSION TRUE
  Enables the `uartAcquireBus()` and `uartReleaseBus()` APIs.

USB driver related setting

- #define USB_USE_WAIT TRUE
  Enables synchronous APIs.

9.99.1 Detailed Description

HAL configuration header.

HAL configuration file, this file allows to enable or disable the various device drivers from your application. You may also use this file in order to override the device drivers default settings.

Index