www.qorvo.com
lpw.support@qorvo.com

k8e Reference Manual

API Description

Version 2.10.2.0
November 15, 2021

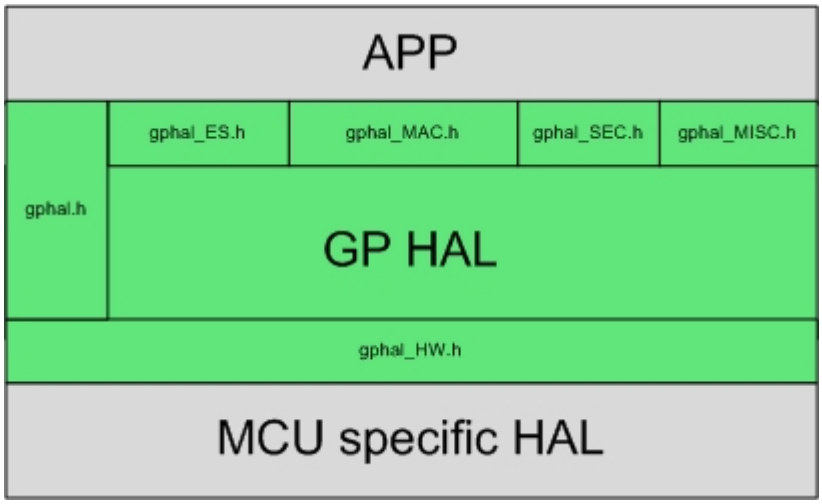
Contents

Chapter 1

Introduction

This document describes in a formal manner the API interface that can be used to control all the functionalities of the GP chip from a microcontroller. The GPHAL API is implemented by the GP HAL component.

The API is split up into several parts. Each part regroups all the functions related to one functional block of the GP chip. The different interfaces structure is shown in the next figure.



Interface Diagram

The HAL code is subdivided in different C-files as the chip itself consists of different blocks.This way the user can quickly use the block he needs, and discard the functionalities of the blocks that are not used by the application thus limiting code size, complexity, etc. If, for instance, the user does not require encryption, the functionalities of the GP chip security processor included in the gphal_SEC files can be deleted from the application.

1.1 gpHal.h

The general functions of the HAL including initialization, reset and interrupt mask control and microcontroller settings are included in the file gpHal.h.

1.2 gpHal_Pbm.h

The API to set and retrieve data and properties from PBM (packet buffer memory).

1.3 gpHal_Coex.h

This file contains the different functions for controlling the Co-Existence interface.

1.4 gpHal_DP.h

This file contains the functions for setting the datapending flag for different remote addresses.

1.5 gpHal_HW.h

The gphal_HW_XXX files contain the microcontroller specific functions (for instance SPI access is included in the gphal_HW_SPI.h file, etc.). These files are microcontroller specific and need to be altered when using the HAL with a different microcontroller then the one included with the evaluation kit.

1.6 gpHal_ES.h

This file contains function prototypes, variables and enumerations to access the hardware event scheduler included in the GP chip. Events can be triggered by three sources:

- **Absolute Events:** An absolute event is triggered at a given time (time is set by the application). The internal time base of the GP chip is compared with the execution time. When they match, the event is executed.
- **Relative Events:** A relative event is triggered a certain time after the event was scheduled in the GP chip (delay is set by the application).
- **External Events:** An external event is triggered when the WKUP pin of the GP chip is triggered.

Four types of events can be scheduled:

- **Transmiting a packet:** When triggered, the GP chip will transmit any packets in its queue.
- **Enable the radio:** When triggered, the GP chip will enable its radio.
- **Disable the radio:** When triggered, the GP chip will disable its radio.
- **Wake up the microcontroller (dummy event):** It is possible to schedule a dummy event in the GP chip. When it is triggered, the INTOUTn pin will be assessed to wake up the microcontroller from a sleep mode.

1.7 gpHal_MAC.h

The GP chip includes a real-time MAC layer. The gphal_MAC.h file contains the functions to access this hardware MAC layer. Its main features are:

- Setting Transmit Power Level
- Setting Channel (Channels 11-26 of the IEEE802.15.4 specification are possible)
- Setting CCA Mode: CCA is done based on received energy, based on modulation of received packet or based on a combination of both

- Purging packets from the transmit queue
- Automatic frame filtering
- Automatic acknowledgement reception/transmission

1.8 gpHal_SEC.h

This file contains the functions to access the integrated security processor of the GP chip. It is possible to do CCM encryption and decryption and AES encryption.

1.9 gpHal_MISC.h

This file contains functions to set/get the GPIO's of the GP chip. It also includes functionality for the ADC and random block of the GP chip.

Chapter 2

Data Structure Documentation

2.1 ble_mgr_start_event_args_t Struct Reference

Data Fields

- UInt8 event_nr
  Number of the ES HW event to be used for this BLE event.
- UInt8 event_type
  Type of the BLE event. Same format as EVENT_TYPE.
- UInt16 info_ptr
  Pointer to the info structure for that BLE event.
- UInt32 schedule_time
  First execution time to be scheduled for the BLE event. in microseconds.

2.1.1 Field Documentation

info_ptr

```
UInt16 ble_mgr_start_event_args_t::info_ptr
```

Note

Value of this pointer must be in the GPMicro address space. Use #TO_GPM_ADDR macro to convert.

2.2 ble_mgr_stat_es_trigger_too_late_t Struct Reference

Data Fields

- UInt8 trigger_type
  Which trigger type was executed too late. one of the ES_TRIGGER_TYPE_∗.
- UInt8 event_type
  Which event type was executed too late. Same format as EVENT_TYPE.
- UInt16 t_too_late
  amount of time in microseconds the trigger was too late.

2.2.1 Field Documentation

t_too_late

`UInt16 ble_mgr_stat_es_trigger_too_late_t::t_too_late`

Warning

> Accuracy of this value is not that good. Because this value is calculated by sampling the symbol counter at ES interrupt entry, and this might be blocked by a higher priority interrupt (which may take up to 200 us).

2.3 gpHal_AbsoluteEventDescriptor Struct Reference

The gpHal_AbsoluteEventDescriptor structure specifying the parameters of an Absolute Event.

2.4 gpHal_AbsoluteEventDescriptor_t Struct Reference

Data Fields

- UInt32 exTime
- UInt32 recPeriod
- UInt16 recAmount
- UInt16 customData
- UInt8 executionOptions
- UInt8 interruptOptions
- UInt8 control
- UInt8 type

2.4.1 Field Documentation

control

`UInt8 gpHal_AbsoluteEventDescriptor_t::control`

This field contains the Event state (see enum gpHal_EventState_t) and the Event result (see enum gpHal_EventResult_t). The macro GP_ES_SET_EVENT_STATE() (resp. GP_ES_SET_EVENT_RESULT()) should be used in order to extract the information and the macro GP_ES_SET_EVENT_STATE() (resp. GP_ES_SET_EVENT_RESULT()) in order to initialize this field.

Preferably the result field is written to INVALID at schedule time. After the event was triggered, this field will return information about the execution status.

customData

`UInt16 gpHal_AbsoluteEventDescriptor_t::customData`

This field contains Custom Data that can be associated with the event. This data can be read when using the gpHal_MonitorAbsoluteEvent() function.

executionOptions

`UInt8 gpHal_AbsoluteEventDescriptor_t::executionOptions`

This field contains the bitfield specifying the execution of the event (see GP_ES_EXECUTION_OPTIONS_MASK).

exTime

`UInt32 gpHal_AbsoluteEventDescriptor_t::exTime`

This field contains the absolute time (absolute to the symbol counter of the GP chip) at which the event should be executed.

interruptOptions

`UInt8 gpHal_AbsoluteEventDescriptor_t::interruptOptions`

This field contains the bitfield specifying the interrupts given by the event (see GP_ES_INTERRUPT_OPTIONS_M

recAmount

`UInt16 gpHal_AbsoluteEventDescriptor_t::recAmount`

This field contains the amount of recurrences of the event. If set to 0 = 1 execution, set to 1 = 2 executions. Value 0xFFFF indicates an endless recurrent event.

recPeriod

`UInt32 gpHal_AbsoluteEventDescriptor_t::recPeriod`

This field contains the period between different periodic executions of this event. This value will only be used by the GP chip when recAmount $>$ 0.

type

`UInt8 gpHal_AbsoluteEventDescriptor_t::type`

This field contains the Event type (see enum gpHal_EventType_t). This specifies what action needs to be performed on execution of the event.

2.5 gpHal_AddressInfo_t Union Reference

Data Fields

- gpHal_RfAddress_t **address**
- UInt16 **panId**
- gpHal_AddressMode_t **addressMode**

2.6 gpHal_BleValidationInputParameters_t Struct Reference

Data Fields

- UInt32 **accessAddress**
- UInt16 **threshHold_low**
- UInt16 **threshHold_med**
- UInt16 **threshHold_hig**

- UInt8 **max_validation_n**
- Bool **isHighDataRate**

2.7 gpHal_BleValidationParameters_t Struct Reference

Data Fields

- UInt8 **scores** [GP_HAL_BLE_NR_OF_VALIDATION_SETTINGS]
- UInt8 **numberOfSimilarities** [GP_HAL_BLE_NR_OF_VALIDATION_SETTINGS]
- UInt8 **firstSimilar** [GP_HAL_BLE_NR_OF_VALIDATION_SETTINGS]
- UInt8 **similarScore** [GP_HAL_BLE_NR_OF_VALIDATION_SETTINGS]
- UInt8 **validationStartIndex**
- UInt16 **validationThresh**
- Bool **fakePreambleFlag**
- UInt8 **fakePreambleStartIndex**
- Bool **isReliableAccessAddress**

2.8 gpHal_BleValidationParameters_test_t Struct Reference

Data Fields

- UInt32 **accessAddress**
- UInt8 **scores** [GP_HAL_BLE_NR_OF_VALIDATION_SETTINGS]
- UInt8 **numberOfSimilarities** [GP_HAL_BLE_NR_OF_VALIDATION_SETTINGS]
- UInt8 **firstSimilar** [GP_HAL_BLE_NR_OF_VALIDATION_SETTINGS]
- UInt8 **similarScore** [GP_HAL_BLE_NR_OF_VALIDATION_SETTINGS]
- UInt8 **validationStartIndex**
- UInt16 **validationThresh**
- Bool **fakePreambleFlag**
- UInt8 **fakePreambleStartIndex**

2.9 gpHal_CalibrationTask_t Struct Reference

Calibration task descriptor.

Data Fields

- gpHal_CalibrationFlags_t flags

  Bitmask of GP_HAL_CALIBRATION_FLAG_xxx.
- UInt16 temperatureThreshold

  Maximum temperature deviation in Q8_8 format (with GP_HAL_CALIBRATION_FLAG_TEMPERATURE_SENSITIVE).
- UInt32 calibrationPeriod

  Maximum time between calibrations in microseconds (with GP_HAL_CALIBRATION_FLAG_PERIODIC).
- Q8_8 temperature

  Temperature at which calibration is triggered.
- void * pUserData

  May be used to pass additional data to the calibration task.

2.10 gpHal_DataReqOptions_t Struct Reference

These options dictate the way a data packet should be transmitted.

Data Fields

- gpHal_MacScenario_t **macScenario**
- gpHal_SourceIdentifier_t **srcId**

2.10.1 Detailed Description

Parameters

gpHal_MacScenario_t	

2.11 gpHal_ExternalEventDescriptor Struct Reference

The gpHal_ExternalEventDescriptor structure specifying the External Event.

2.12 gpHal_ExternalEventDescriptor_t Struct Reference

Data Fields

- gpHal_EventType_t type

2.12.1 Field Documentation

type

`gpHal_EventType_t gpHal_ExternalEventDescriptor_t::type`

This field contains the Event type (see enum gpHal_EventType_t). This specifies what action needs to be performed on execution of the event.

2.13 gpHal_IpcBackupRestoreFlags_t Struct Reference

Data Fields

- UInt8 **interruptFlags**

2.14 gpHal_PaSettings_t Struct Reference

Data Fields

- UInt8 **PbmSettingAntselInt**
- Int8 **internalDbmSetting**
- Bool **pa_low**
- Bool **pa_ultralow**

2.15 gpHal_RfAddress_t Union Reference

Data Fields

- UInt16 Short
- MACAddress_t Extended

2.15.1 Field Documentation

Extended

`MACAddress_t gpHal_RfAddress_t::Extended`

The extended address (MAC address), 8 bytes.

Short

`UInt16 gpHal_RfAddress_t::Short`

The short address (2 bytes).

2.16 gpHal_RxInfo_t Struct Reference

Data Fields

- UInt8 rxChannel

  The channel the packet was received on.

2.17 gpHal_StatisticsCntPrio_t Struct Reference

Data Fields

- UInt16 **prio0**
- UInt16 **prio1**
- UInt16 **prio2**
- UInt16 **prio3**

2.18 gpHal_StatisticsCoexCounter_t Struct Reference

Data Fields

- gpHal_StatisticsCntPrio_t **coexReq**
- gpHal_StatisticsCntPrio_t **coexGrant**

2.19 gpHal_StatisticsMacCounter_t Struct Reference

Data Fields

- UInt16 **ccaFails**
- UInt16 **txRetries**
- UInt16 **failTxNoAck**
- UInt16 **failTxChannelAccess**
- UInt16 **successTx**
- UInt16 **totalRx**
- UInt16 **pbmOverflow**

2.20 rangedescription Struct Reference

Data Fields

- struct rangelist * **rlp**
- UInt16 **rangesize**

2.21 rangelist Struct Reference

Data Fields

- gpHal_Address_t **startAddress**
- gpHal_Address_t **endAddress**

Chapter 3

File Documentation

3.1 gp_global.h File Reference

Contains general definitions used in the different blocks.

Data Structures

- struct gpHal_RxInfo_t

Typedefs

- typedef void(∗ gpHal_AbsoluteEventCallback_t) (void)

  The gpHal_AbsoluteEventCallback_t callback type definition defines the callback prototype of an Absolute Event interrupt.
- typedef void(∗ gpHal_ExternalEventCallback_t) (void)

  The gpHal_ExternalEventCallback_t callback typedef defines the callback prototype of the External Event interrupt.
- typedef void(∗ **gpHal_LowBatteryCallback_t**) (void)

Functions

- void gpHal_cbExternalEvent (void)

  This callback is called when an absolute event has occured.

gpHal_Result_t

- #define gpHal_ResultSuccess 0x0

  The function returned successful.
- #define gpHal_ResultInvalidParameter 0x5

  An invalid parameter was given as a parameter to this function.
- #define gpHal_ResultRxOn 0x6

  The GP chip is in receive mode.
- #define gpHal_ResultBusy 0x7

  The GP chip is busy.
- #define gpHal_ResultTrxOff 0x8

  The GP chip radio is off.

- #define gpHal_ResultTxOn 0x9

  The GP chip radio is transmitting.
- #define gpHal_ResultGrantTimeout 0xa

  The GP chip has timed out waiting for the COEX Grant signal.
- #define gpHal_ResultUnsupported 0xb

  The GP chip unsupported operation.
- #define gpHal_ResultInvalidRequest 0xc2

  The request was invalid (event not present, ...)
- #define gpHal_ResultInvalidHandle 0xe7

  The handle given as parameter cannot be found.
- #define gpHal_ResultCCAFailure 0xe1

  Channel access failure.
- #define gpHal_ResultNoAck 0xe9

  Ack was required but no ack received.
- typedef UInt8 gpHal_Result_t

  The gpHal_Result_t type defines the result of various HAL functions.

3.1.1 Detailed Description

The file gp_global.h contains general definitions used in the different blocks. The result enumeration for the gphal functions and different callbacks can be found here.

3.1.2 Function Documentation

gpHal_cbExternalEvent()

```
void gpHal_cbExternalEvent (
            void  )
```

This callback is called when an absolute event has occured.

This callback has to be implemented by the software layer that is using the GPHAL (only required when using the fixed callbacks).

3.2 gpHal.h File Reference

The file gpHal.h contains the general functions of the HAL (init, reset, interrupts).

Macros

- #define **GP_DIVERSITY_NR_OF_STACKS** 1
- #define **GP_HAL_DEFAULT_TIMEOUT** 10000UL
- #define **GP_HAL_MAXIMUM_TIMEOUT** 2097119UL
- #define **GP_HAL_TIME_DIFF**(t1, t2) (t2 <= t1 ? (t1 - t2) : (0xFFFFFFFF - (t1 - t2)))
- #define GP_HAL_TIME_COMPARE_LOWER(t1, t2) (!((UInt32)((t1) - (t2))/*&(0xFFFFFFFF)*/ < (0x80000000LU)))

  Compares times from the chip's timebase - check if t1 < t2.
- #define GP_HAL_TIME_COMPARE_BIGGER_EQUAL(t1, t2) (((UInt32)((t1) - (t2))/*&(0xFFFFFFFF)*/ < (0x80000000LU)))

Compares times from the chip's timebase - check if t1 >= t2.

- #define GP_HAL_TIME_COMPARE_BIGGER(t1, t2) (!((UInt32)((t2) - (t1))/*&(0xFFFFFFFF)*/ < (0x80000000LU)))

  Compares times from the chip's timebase - check if t1 > t2.
- #define GP_HAL_TIME_COMPARE_LOWER_EQUAL(t1, t2) (((UInt32)((t2) - (t1))/*&(0xFFFFFFFF)*/ < (0x80000000LU)))

  Compares times from the chip's timebase - check if t1 <= t2.
- #define gpHal_GetChipId() (GP_HAL_EXPECTED_CHIP_ID)

  Getter method for the chip ID.
- #define gpHal_EnableInterrupts(enable) GP_HAL_ENABLE_INTERRUPTS(enable)

  Thie functions enables the interrupt line of the GP chip.

Functions

- UInt8 gpHal_ReadReg (gpHal_Address_t Register)

  Reads a register of the GP chip.
- void gpHal_ReadRegs (gpHal_Address_t Address, void *pBuffer, UInt8 Length)

  Reads a block of registers of the GP chip.
- void gpHal_WriteReg (gpHal_Address_t Register, UInt8 Data)

  Write a register of the GP chip.
- void gpHal_WriteRegs (gpHal_Address_t Address, void *pBuffer, UInt8 Length)

  Writes a block of registers to the GP chip.
- void gpHal_ReadModifyWriteReg (gpHal_Address_t Register, UInt8 Mask, UInt8 Data)

  Reads a register, modifies the data with a certain mask and data, writes the register back.
- Bool gpHal_CheckMsi (void)

  Checks if MSI communication is possible and correct by reading a known register.
- UInt16 gpHal_GetHWVersionId (void)

  Returns the version information of the chip.
- UInt8 gpHal_GetChipVersion (void)

  Getter method for the chip version.
- UInt8 gpHal_GetRomBlVersion (void)

  Getter method for the ROM BL version.
- void gpHal_Init (Bool timedMAC)

  Initializes HAL variables and sets basic GP chip register values.
- void gpHal_AdvancedInit (void)

  Initializes Advanced HAL variables.
- void gpHal_Reset (void)

  This function performs a reset of the chip.
- UInt8 gpHal_IsRadioAccessible (void)

  This function checks if radio is awake.
- Bool gpHal_DidGPReset (void)

  This function detects if a reset of the GP chip has occurred.
- void gpHal_Interrupt (void)

  The interrupt service routine to be called when the interrupt of the GP chip is seen.
- void gpHal_GoToSleepWhenIdle (Bool enable)

  This function regulates the GP chip sleep behaviour.

3.2.1 Detailed Description

The general functions of the HAL including initialization, reset and interrupt mask control and MCU settings are included in the file gpHal.h

3.2.2 Macro Definition Documentation

gpHal_EnableInterrupts

```
#define gpHal_EnableInterrupts(
                enable ) GP_HAL_ENABLE_INTERRUPTS(enable)
```

Sets the main interrupt mask of the GP chip.

Parameters

enable	If set to true: main interrupt mask is switched on.

gpHal_GetChipId

```
#define gpHal_GetChipId( ) (GP_HAL_EXPECTED_CHIP_ID)
```

Returns

The identifier of the silicon.

3.2.3 Function Documentation

gpHal_AdvancedInit()

```
void gpHal_AdvancedInit (
                void  )
```

The function has to be called at the end of the application initialization. In comparison with the basic gpHal_Init() method, the settings triggered by this initialization could interfere with initializations of other components.

gpHal_DidGPReset()

```
Bool gpHal_DidGPReset (
                void  )
```

This function will return true if the GP chip has resetted since its first startup.

gpHal_GetChipVersion()

```
UInt8 gpHal_GetChipVersion (
                void  )
```

Returns

The metal fix version of the chip.

gpHal_GetHWVersionId()

```
UInt16 gpHal_GetHWVersionId (
              void  )
```

This function returns the version information of the chip. It indicates the current revision of the chip and other information

gpHal_GetRomBlVersion()

```
UInt8 gpHal_GetRomBlVersion (
              void  )
```

Note that this function returns -on purpose- only the MSB of the ROM bootloader version.

Returns

The version of the ROM bootloader.

gpHal_GoToSleepWhenIdle()

```
void gpHal_GoToSleepWhenIdle (
              Bool enable )
```

This function enables the sleep mode of the GP chip when the GP chip is idle. Enabling GoToSleepWhenIdle will cause the GP chip to go to sleep mode when nothing is busy (TX/RX/Receiver on/...) Be aware that the GP chip remains in sleep if no wakeup trigger or event is enabled and registered.

It also regulates the GP chip sleep behaviour by keeping track of a "stay awake counter". When the counter reaches '0' the GP chip is put into sleep. It always needs to be used in pairs, one to wake up the GP chip when its needed and one to set it to sleep when the GP chip is no longer of use. In this way different functions can use the sleep functionality without overriding each others settings.

Parameters

enable	• if set to true : The GP chip will be put to sleep if no other wake requests are pending, counter decremented • if set to false: The GP chip will be kept awake, counter incremented

gpHal_Init()

```
void gpHal_Init (
              Bool timedMAC )
```

The function has to be called at the beginning of the application as it initializes variables that are used throughout operation. At startup a decision is made to use the timed operation of the GP chip or not, by setting the parameter timedMAC.

Parameters

timedMAC	If set to true, the GP chip will be used with a timed MAC. When a timed MAC is used all transmission is done using scheduled triggers from the Event Scheduler (ES).

gpHal_Interrupt()

```
void gpHal_Interrupt (
              void  )
```

This ISR needs to be called when the interrupt line (INTOUTn) goes low (active low signal). It will handle the pending interrupt according to the callback functions registered to the different sources.

Parameters

ID	no functionality.

gpHal_IsRadioAccessible()

```
UInt8 gpHal_IsRadioAccessible (
              void  )
```

Returns

Returns if value is awake. If value is 0, chip is asleep else the device is awake.

gpHal_ReadReg()

```
UInt8 gpHal_ReadReg (
              gpHal_Address_t Register )
```

Reads a register of the GP chip.

Parameters

Register	The register address to read data from.

gpHal_ReadRegs()

```
void gpHal_ReadRegs (
              gpHal_Address_t Address,
```

```
            void * pBuffer,
            UInt8 Length )
```

Reads a block of registers of the GP chip.

Parameters

Address	The register address where the block read starts.
pBuffer	The pointer to a byte buffer where the read data will be stored.
Length	The number of bytes to be read.

gpHal_Reset()

```
void gpHal_Reset (
            void  )
```

This function performs a reset of the functional registers. All registers are reverted to their default value and all memories are cleared.

gpHal_WriteReg()

```
void gpHal_WriteReg (
            gpHal_Address_t Register,
            UInt8 Data )
```

Write a register of the GP chip.

Parameters

Register	The register address to write to.
Data	The data to write to the register.

gpHal_WriteRegs()

```
void gpHal_WriteRegs (
            gpHal_Address_t Address,
            void * pBuffer,
            UInt8 Length )
```

Writes a block of registers to the GP chip.

Parameters

Address	The register address where the block write starts.
pBuffer	The pointer to a byte buffer where the data to be written are stored.
Length	The number of bytes to be written.

3.3 gpHal_Coex.h File Reference

Typedefs

- typedef void(∗ **gpHal_CoexCbDataConfirm_t**) (UInt8 result, UInt8 retries, UInt8 priority)
- typedef UInt8(∗ **gpHal_CoexCbGpioInt_t**) (UInt8 interruptsMasked)
- typedef void(∗ **gpHal_CoexCbCSMARetry_t**) (UInt8 result, void ∗pCSMA_CA_State)

Functions

- gpHal_Result_t gpHal_Set_MAC_RX_Packet (Bool request, UInt8 priority)

  Set the coexistence parameters for 802.15.4 packet RX.
- gpHal_Result_t gpHal_Set_MAC_TX_ACK (Bool request, UInt8 priority, gpHal_Coex_MAC_TX_ACK_NotGran txAckNotGrantedAction)

  Set the coexistence parameters for 802.15.4 ACK TX.
- gpHal_Result_t gpHal_Set_MAC_TX_Packet (Bool request, UInt8 priority, gpHal_Coex_MAC_TX_Packet_NotG txNotGrantedAction)

  Set the coexistence parameters for 802.15.4 packet TX.
- gpHal_Result_t gpHal_Set_MAC_RX_ACK (Bool request, UInt8 priority)

  Set the coexistence parameters for 802.15.4 ACK RX.
- gpHal_Result_t gpHal_Set_MAC_RX_ReqExt (gpHal_MAC_ReqExtTrigger_t trigger, UInt8 priority)

  Configure the request extensions.
- gpHal_Result_t gpHal_Set_GainControl (gpHal_GainControl_Mode_t gainControlMode, gpHal_AttLna_t attLnaLow, gpHal_AttLna_t attLnaHigh)

  Set the gain control.
- gpHal_Result_t gpHal_Set_MAC_EarlyPreambleDetect (Bool enableEarlyPreambleDetect)

  Enable early preamble detection.
- gpHal_Result_t gpHal_Set_MAC_ExtensionTimeout (UInt32 extCoexTimeout)

  Set extension timeout in multiples of 16 us.
- gpHal_Result_t gpHal_Set_MAC_MacRetriesTreshold (UInt8 retriesCnt)

  Set the number of consecutive mac retries before raising the priority of the packet TX. Setting to 0 deactivate it.
- gpHal_Result_t gpHal_Set_MAC_CcaRetriesTreshold (UInt8 retriesCnt)

  Set the number of consecutive cca retries before raising the priority of the packet TX. Setting to 0 deactivate it.
- gpHal_Result_t gpHal_Set_MAC_ReRequest (gpHal_MAC_ReRequestTrigger_t trigger, UInt8 offTime, UInt8 onTime)

  Configure if the request line should be re-requested on case of grant not given or lost for over x time.
- gpHal_Result_t gpHal_Set_MAC_IndTxPriorityBoost (Bool enable, UInt8 priority)

  Configure if an 802.15.4 indirect packet TX should have a different priority level as normal TX.
- UInt8 gpHal_CompletePossibleGrantimeoutByConfigRestore (UInt8 result)

  Finalize handling after timeout on "Wait for COEX GRANT" Actions: Restore saved configuration for PA (internal and, if applcable, external PA) Modify result accordingly.

- #define gpHal_MAC_ReRequestTrigger_None 0x00

  No action.
- #define gpHal_MAC_ReRequestTrigger_NoGrant 0x01

Re-Request if grant is lost or not given.

- #define gpHal_MAC_ReRequestTrigger_PrioChange 0x02

  Re-Request on priority change during ongoing request.
- typedef UInt8 gpHal_MAC_ReRequestTrigger_t

  Mask with the ReRequest triggers.

- #define gpHal_MAC_TX_Packet_Ignore 0x00

  No action, grant not aware.
- #define gpHal_Coex_MAC_TX_Packet_DisablePa 0x01

  If not granted, then disable PA.
- #define gpHal_Coex_MAC_TX_Packet_CcaFailure 0x02

  If not granted, then trigger a CMSA_CA failure. Only applicable to TX_packet.
- #define gpHal_Coex_MAC_TX_Packet_CcaHold 0x04

  If not granted, CSMA-CA delay backoff counter get frozen. Once grant is received, conter proceeds. Can be used to force wait for grant before CCA measurement. When using SW CSMA-CA, backoff delay is handled by SW, so this option is ignored.
- #define gpHal_Coex_MAC_TX_Packet_DelayedStart 0x08

  If not granted, TX state machine is hold after doing CSMA-CA, but before starting the TX. Once Grant is received, TX starts imediatelly.
- typedef UInt8 gpHal_Coex_MAC_TX_Packet_NotGrantedActions_t

  Mask that sets the MAC packet TX actions upon not having grant.

- #define gpHal_MAC_TX_ACK_Ignore 0x00

  No action, grant not aware.
- #define gpHal_Coex_MAC_TX_ACK_DisablePa 0x01

  If not granted, then disable PA.
- #define gpHal_Coex_MAC_TX_ACK_SkipTx 0x02

  If not granted, skip sending of the ACK.
- typedef UInt8 gpHal_Coex_MAC_TX_ACK_NotGrantedActions_t

  Mask that sets the MAC ACK TX actions upon not having grant.

- #define gpHal_MAC_ExtensionTriggers_None 0x00

  No extension.
- #define gpHal_MAC_ExtensionTriggers_Preamble 0x01

  Extend on preamble detect.
- #define gpHal_MAC_ExtensionTriggers_SFD 0x02

  Extend on SFD reception.
- #define gpHal_MAC_ExtensionTriggers_PacketAbort 0x04

  Extend on macfilter.
- #define gpHal_MAC_ExtensionTriggers_FCSERR 0x08

  Extend on CRC failure.
- typedef UInt8 gpHal_MAC_ReqExtTrigger_t

  Mask that sets request extension triggers.

- #define gpHal_GainControl_Mode_Default 0x00

  Use default fixed gain control level LNA0.
- #define gpHal_GainControl_Mode_Fixed 0x01

  Gain control levels will be fixed to the specified lowLnaAtt setting.
- #define gpHal_GainControl_Mode_RssiBasedAgc 0x02

Gain control levels will be controlled by the internal AGC.

- #define gpHal_GainControl_Mode_GpioBasedAgc 0x03
  Gain control levels will be controlled by the configured COEX_ATT_CTRL BSP setting.
- typedef UInt8 gpHal_GainControl_Mode_t
  Enumeration specifying the gain control mode.
- #define gpHal_AttLna_LNA0 0x00
  Select LNA0.
- #define gpHal_AttLna_LNA1 0x01
  Select LNA1.
- #define gpHal_AttLna_LNA2 0x02
  Select LNA2.
- #define gpHal_AttLna_LNA3 0x03
  Select LNA3.
- #define gpHal_AttLna_LNA4 0x04
  Select LNA4.
- #define gpHal_AttLna_LNA5 0x05
  Select LNA5.
- #define gpHal_AttLna_Ignore 0xFF
  Don't use/update LNA setting.
- typedef UInt8 gpHal_AttLna_t
  Selection of predefined LNA condif.

3.3.1 Detailed Description

gpHal Coexistence subcomponent
Declarations of the public functions and enumerations of gpHal_Coex.

3.3.2 Function Documentation

gpHal_CompletePossibleGrantimeoutByConfigRestore()

```
UInt8 gpHal_CompletePossibleGrantimeoutByConfigRestore (
              UInt8 result )
```

gpHal_Set_GainControl()

```
gpHal_Result_t gpHal_Set_GainControl (
              gpHal_GainControl_Mode_t gainControlMode,
              gpHal_AttLna_t attLnaLow,
              gpHal_AttLna_t attLnaHigh )
```

Parameters

gainControlMode	
attLnaLow	
attLnaHigh	

Returns

result

gpHal_Set_MAC_CcaRetriesTreshold()

```
gpHal_Result_t gpHal_Set_MAC_CcaRetriesTreshold (
               UInt8 retriesCnt )
```

Parameters

retriesCnt	

Returns

result

gpHal_Set_MAC_EarlyPreambleDetect()

```
gpHal_Result_t gpHal_Set_MAC_EarlyPreambleDetect (
               Bool enableEarlyPreambleDetect )
```

Parameters

enableEarlyPreambleDetect	

Returns

result

gpHal_Set_MAC_ExtensionTimeout()

```
gpHal_Result_t gpHal_Set_MAC_ExtensionTimeout (
               UInt32 extCoexTimeout )
```

Parameters

extCoexTimeout	

Returns

result Returns false when value out of range, else sucess.

gpHal_Set_MAC_IndTxPriorityBoost()

```
gpHal_Result_t gpHal_Set_MAC_IndTxPriorityBoost (
```

```
Bool enable,
UInt8 priority )
```

Parameters

enable	Enable the priority boost for indirect TX. When enabled, all indirect TX will use the priority defined by this API. When disabled, the priority applied is the same on for a normal packet TX.
priority	Priority level to be used when enabled.

Returns

result

gpHal_Set_MAC_MacRetriesTreshold()

```
gpHal_Result_t gpHal_Set_MAC_MacRetriesTreshold (
UInt8 retriesCnt )
```

Parameters

retriesCnt	

Returns

result

gpHal_Set_MAC_ReRequest()

```
gpHal_Result_t gpHal_Set_MAC_ReRequest (
gpHal_MAC_ReRequestTrigger_t trigger,
UInt8 offTime,
UInt8 onTime )
```

Parameters

trigger	Enable re-request upon no-grant or priority change
offTime	Set the time in uS that request should be off during a re-request toggle.
onTime	Set the time in uS it should wait for the grant, before turning request off to turn it on again.

Returns

result

gpHal_Set_MAC_RX_ACK()

```
gpHal_Result_t gpHal_Set_MAC_RX_ACK (
               Bool request,
               UInt8 priority )
```

Parameters

request	
priority	

Returns

result

gpHal_Set_MAC_RX_Packet()

```
gpHal_Result_t gpHal_Set_MAC_RX_Packet (
               Bool request,
               UInt8 priority )
```

Parameters

request	
priority	

Returns

result

gpHal_Set_MAC_RX_ReqExt()

```
gpHal_Result_t gpHal_Set_MAC_RX_ReqExt (
               gpHal_MAC_ReqExtTrigger_t trigger,
               UInt8 priority )
```

Parameters

trigger	
priority	

Returns

result

gpHal_Set_MAC_TX_ACK()

```
gpHal_Result_t gpHal_Set_MAC_TX_ACK (
               Bool request,
               UInt8 priority,
               gpHal_Coex_MAC_TX_ACK_NotGrantedActions_t txAckNotGrantedAction )
```

Parameters

request	
priority	
txAckNotGrantedAction	

Returns

result

gpHal_Set_MAC_TX_Packet()

```
gpHal_Result_t gpHal_Set_MAC_TX_Packet (
               Bool request,
               UInt8 priority,
               gpHal_Coex_MAC_TX_Packet_NotGrantedActions_t txNotGrantedAction )
```

Parameters

request	
priority	
txNotGrantedAction	

Returns

result

3.4 gpHal_DP.h File Reference

This file contains all the functions needed for DataPending functionality.

Data Structures

- union gpHal_RfAddress_t
- union gpHal_AddressInfo_t

Functions

- gpHal_Result_t **gpHal_DpClearEntries** (UInt8 id)
- gpHal_Result_t gpHal_DpAddEntry (gpHal_AddressInfo_t *pAddressInfo, UInt8 id)

  Add an entry to the list of addresses for which a data packet is pending transission.
- gpHal_Result_t gpHal_DpRemoveEntry (gpHal_AddressInfo_t *pAddressInfo, UInt8 id)

  Remove an entry from the list of addresses for which a data packet is pending transission.
- Bool gpHal_DPEntriesPending (void)

  Check if there are datapending entries in the list.
- #define gpHal_AddressModeNoAddress 0

  No Address.
- #define gpHal_AddressModeReserved 1

  Reserved.
- #define gpHal_AddressModeShortAddress 2

  Short (i.e. 16-bit) address.
- #define gpHal_AddressModeExtendedAddress 3

  Extended (i.e. 8-byte) address.
- typedef UInt8 gpHal_AddressMode_t

  Selection of the address mode.

3.4.1 Function Documentation

gpHal_DpAddEntry()

```
gpHal_Result_t gpHal_DpAddEntry (
               gpHal_AddressInfo_t * pAddressInfo,
               UInt8 id )
```

Parameters

pAddressInfo	The address for which a data packet is pending
id	The stack id which has the data packet pending

Returns

result The return parameter indicating success or the failure code

gpHal_DPEntriesPending()

```
Bool gpHal_DPEntriesPending (
               void  )
```

Returns

result A boolean indicating if any stack has pending data packets in the list.

gpHal_DpRemoveEntry()

```
gpHal_Result_t gpHal_DpRemoveEntry (
               gpHal_AddressInfo_t * pAddressInfo,
               UInt8 id )
```

Parameters

pAddressInfo	The address for which a data packet is no longer pending
id	The stack id which had the data packet pending

Returns

result The return parameter indicating success or the failure code

3.5 gpHal_DPI.h File Reference

This file contains functions for Deep Packet Inspection.

Functions

- void gpHal_DpiPrepareForConfig (void)
  This function prepares the DPI block for further configuration.
- Bool gpHal_DpiCheckPreConditions (void)
  Returns true if enabling without conflicts is allowed.
- Bool gpHal_DpiIsDpiRunning (void)
  Returns true if DPI is running.
- gpHal_Result_t gpHal_DpiEnable (void)
  Enable the DPI block - only call when gpHal_DpiCheckPreConditions() returned true.
- gpHal_Result_t gpHal_DpiDisable (void)
  Disable the DPI block.
- void gpHal_DpiAddPattern (UInt8 *pPattr, UInt8 length)
  Add a pattern to be checked by DPI.
- void gpHal_DpiAddDevice (UInt32 *pFrameCnt, UInt16 *pShortAddr, MACAddress_t *pLongAddr, UInt8 *pSecKey)
  Add a device to be checked by DPI.

3.6 gpHal_DPI_ZB.h File Reference

This file contains functions for Deep Packet Inspection on ZB networks, to allow HW buffering of packets before handling it.

Functions

- void gpHal_DpiZbSetBuffering (UInt8 packetsBuffered)
  This function configures the low level DPI filtering and buffering of zigbee packets.
- void gpHal_DpiZbEnable (Bool enable)
  This function set the low level DPI filtering and buffering of zigbee packets.

3.6.1 Function Documentation

gpHal_DpiZbEnable()

```
void gpHal_DpiZbEnable (
              Bool enable )
```

Parameters

enale	True to enale filtering mode, false to disable it.

gpHal_DpiZbSetBuffering()

```
void gpHal_DpiZbSetBuffering (
              UInt8 packetsBuffered )
```

Parameters

packetsBuffered	Amount of packets that arrives before enabling the interrupt.

3.7 gpHal_ES.h File Reference

All functions for the event scheduler and sleep modes.

Data Structures

- struct gpHal_AbsoluteEventDescriptor_t
- struct gpHal_ExternalEventDescriptor_t

Macros

- #define GP_ES_EXECUTION_OPTIONS_MASK 0xF
  Event exection options mask.
- #define GP_ES_EXECUTION_OPTIONS_EXECUTE_IF_TOO_LATE 0x4
  Event exection options : Execute the event even if the trigger time has passed.
- #define GP_ES_EXECUTION_OPTIONS_PROHIBIT_STANDBY 0x8
  Event exection options : Prohibit standby as long as event is pending.
- #define GP_ES_EXECUTION_OPTIONS_NOT_EXECUTE_IF_TOO_LATE 0x0
  Event exection options : Execute the event only if the event was triggered on time.
- #define GP_ES_INTERRUPT_OPTIONS_MASK 0x3F
  Event interrupt options mask (enables all interrupt options).
- #define GP_ES_INTERRUPT_OPTIONS_ON_FIRST_ON_TIME 0x01
  Event interrupt option: generate interrupt on first event execution that is on time.
- #define GP_ES_INTERRUPT_OPTIONS_ON_OTHERS_ON_TIME 0x02

Event interrupt option: generate interrupt on event execution other then first or last execution that is on time.

- #define GP_ES_INTERRUPT_OPTIONS_ON_LAST_ON_TIME 0x04

  Event interrupt option: generate interrupt on last event execution that is on time.
- #define GP_ES_INTERRUPT_OPTIONS_ON_FIRST_TOO_LATE 0x08

  Event interrupt option: generate interrupt on first event execution that is too late.
- #define GP_ES_INTERRUPT_OPTIONS_ON_OTHERS_TOO_LATE 0x10

  Event interrupt option: generate interrupt on execution other then first or last execution that is too late.
- #define GP_ES_INTERRUPT_OPTIONS_ON_LAST_TOO_LATE 0x20

  Event interrupt option: generate interrupt on last event execution that is too late.
- #define GPHAL_ES_32KHZ_SLEEP_DEFAULT_CALIB 0x3D090000

  Default value for the calibration of the 32kHz crystal.
- #define GPHAL_ES_16MHZ_SLEEP_DEFAULT_CALIB 0x40000000

  Default value for the calibration of the 16MHz crystal.
- #define GP_ES_GET_EVENT_RESULT(control) (((control)>>4) & 0xF)

  Get the event result (top 4 bits of the control field).
- #define GP_ES_GET_EVENT_STATE(control) ((control) & 0xF)

  Get the event state (last 4 bits of the control field).
- #define GP_ES_SET_EVENT_RESULT(control, result) (control = (control & 0x0F) | (((result)<<4) & 0xF0))

  Set the event result (top 4 bits of the control field).
- #define GP_ES_SET_EVENT_STATE(control, state) (control = (control & 0xF0) | ((state) & 0x0F))

  Set the event state (last 4 bits of the control field).
- #define **GPHAL_ES_ABSOLUTE_EVENT_ID_INVALID** 0xFF
- #define **gpHal_SleepClockMeasurementStatusNotStarted** 0x00
- #define **gpHal_SleepClockMeasurementStatusPending** 0x01
- #define **gpHal_SleepClockMeasurementStatusNotStable** 0x02
- #define **gpHal_SleepClockMeasurementStatusStable** 0x03
- #define gpHal_EnableExternalEventCallbackInterrupt(enable) GP_HAL_ENABLE_EXTERNAL_EVENT_INTE

  This function enables the interrupt line of the External Event interrupt.

Typedefs

- typedef UInt8 **gpHal_SleepClockMeasurementStatus_t**

gpHal_EventType_t

- typedef UInt8 gpHal_EventType_t

  The gpHal_EventType_t type defines the event type.

gpHal_AbsoluteEventId_t

- typedef UInt8 gpHal_AbsoluteEventId_t

  The gpHal_AbsoluteEventId_t type holds an absolue event index.

gpHal_SleepMode_t

- typedef UInt8 gpHal_SleepMode_t

  The gpHal_SleepMode_t type defines the GP chip sleep mode.

Functions

- void gpHal_EnableAbsoluteEventCallbackInterrupt (UInt8 eventNbr, Bool enable)

  This function enables the interrupt line of an Absolute Event interrupt.
- void gpHal_RegisterAbsoluteEventCallback (gpHal_AbsoluteEventCallback_t callback, UInt8 eventNbr)

  Registers the callback for an Absolute Event.
- gpHal_ExternalEventCallback_t gpHal_RegisterExternalEventCallback (gpHal_ExternalEventCallback_t callback)

  Registers the callback for a External Event.
- void gpHal_ResetTime (void)

  Resets the timebase of the GP chip.
- void gpHal_GetTime (UInt32 *pTime)

  Gets the time of the GP chip.
- void gpHal_ApplyCalibration (Int32 phaseAdjustment, UInt32 frequency)

  Calibrates the timer of the GP chip.
- void gpHal_ScheduleAbsoluteEvent (gpHal_AbsoluteEventDescriptor_t *pAbsoluteEventDescriptor, gpHal_AbsoluteEventId_t eventNbr)

  Schedules an Absolute Event in the GP chip.
- gpHal_AbsoluteEventId_t gpHal_GetAbsoluteEvent (void)

  Allocates an available absolute event id.
- void gpHal_FreeAbsoluteEvent (gpHal_AbsoluteEventId_t EventId)

  Frees an allocated absolute event id.
- void gpHal_RefreshAbsoluteEvent (gpHal_AbsoluteEventId_t eventNbr, UInt32 absTime, UInt8 control)

  Refreshes an Absolute Event in the GP chip.
- gpHal_EventState_t gpHal_UnscheduleAbsoluteEvent (gpHal_AbsoluteEventId_t eventNbr)

  Unschedules an Absolute Event.
- gpHal_Result_t gpHal_MonitorAbsoluteEvent (UInt8 eventNbr, gpHal_AbsoluteEventDescriptor_t *pAbsoluteEvent)

  Returns all information about a registered event.
- void gpHal_ScheduleImmediateEvent (gpHal_EventType_t type)

  Schedules an immediate event trigger in the GP chip.
- void gpHal_ScheduleExternalEvent (gpHal_ExternalEventDescriptor_t *pExternalEventDescriptor)

  Schedules the External Event in the GP chip.
- gpHal_Result_t gpHal_UnscheduleExternalEvent (void)

  Unschedules the External Event.
- gpHal_Result_t gpHal_MonitorExternalEvent (gpHal_ExternalEventDescriptor_t *pExternalEventDescriptor)

  Returns all information about the External Event.
- gpHal_Result_t gpHal_SetSleepMode (gpHal_SleepMode_t mode)

  Sets the sleep mode of the GP chip.
- gpHal_SleepMode_t gpHal_GetSleepMode (void)

  Gets the sleep mode of the GP chip.
- gpHal_SleepClockMeasurementStatus_t gpHal_GetMeasuredSleepClockFrequency (gpHal_SleepMode_t mode, UInt32 *frequencymHz)

  defined(GP_DIVERSITY_GPHAL_COPROC)
- UInt16 gpHal_GetSleepClockAccuracy (void)

  Returns the average sleep clock accuracy of the currently selected sleep clock.
- UInt16 gpHal_GetWorstSleepClockAccuracy (void)

  Returns the worst-case sleep clock accuracy of the currently selected sleep clock.

gpHal_EventState_t

- #define gpHal_EventStateInvalid GPHAL_ENUM_EVENT_STATE_INVALID

  The event is invalid, will not be executed if execution time is reached.
- #define gpHal_EventStateScheduled GPHAL_ENUM_EVENT_STATE_SCHEDULED

  The event is scheduled, it will be executed if execution time is reached.
- #define gpHal_EventStateScheduledForImmediate GPHAL_ENUM_EVENT_STATE_SCHEDULED_FOR_IMI

  The event is scheduled for immediate, it will be executed as soon as possible.
- #define gpHal_EventStateReScheduled GPHAL_ENUM_EVENT_STATE_RESCHEDULED

  The event is rescheduled after being triggered before. It will be executed if execution time is reached.
- #define gpHal_EventStateDone GPHAL_ENUM_EVENT_STATE_DONE

  The event has been executed.
- typedef UInt8 gpHal_EventState_t

  The gpHal_EventState_t type defines the Absolute Event state.

gpHal_EventResult_t

- #define gpHal_EventResultInvalid GPHAL_ENUM_EVENT_RESULT_UNKNOWN

  Event not yet executed.
- #define gpHal_EventResultOnTime GPHAL_ENUM_EVENT_RESULT_EXECUTED_ON_TIME

  Event was executed on time.
- #define gpHal_EventResultTooLate GPHAL_ENUM_EVENT_RESULT_EXECUTED_TOO_LATE

  Event was executed too late and GP_ES_EXECUTION_OPTIONS_EXECUTE_IF_TOO_LATE was set.
- #define gpHal_EventResultMissed GPHAL_ENUM_EVENT_RESULT_MISSED_TOO_LATE

  Event was executed too late and GP_ES_EXECUTION_OPTIONS_NOT_EXECUTE_IF_TOO_LATE was set.
- typedef UInt8 gpHal_EventResult_t

  The gpHal_EventResult_t type defines the event result.

3.7.1 Detailed Description

This file defines all functions for the event scheduler and sleep modes. These functions can be used to schedule certain actions : an interrupt, TX of a packet, etc. The different sleep and wakeup modes can also be initialized and used with these functions.

3.7.2 Macro Definition Documentation

gpHal_EnableExternalEventCallbackInterrupt

```
#define gpHal_EnableExternalEventCallbackInterrupt(
                enable ) GP_HAL_ENABLE_EXTERNAL_EVENT_INTERRUPT(enable)
```

This function enables the interrupt line of the External Event interrupt by setting the interrupt mask of the External Event interrupt.

Parameters

enable	Enables the interrupt source if true.

3.7.3 Function Documentation

gpHal_ApplyCalibration()

```
void gpHal_ApplyCalibration (
              Int32 phaseAdjustment,
              UInt32 frequency )
```

This function calibrates the GP chip time base by applying a correction of the current time and an adjustment of the timer slope.

Parameters

phaseAdjustment	The phase adjustment to be applied to the GP chip timer.
frequency	The desired frequency of the timer/slope of the counter.

gpHal_EnableAbsoluteEventCallbackInterrupt()

```
void gpHal_EnableAbsoluteEventCallbackInterrupt (
              UInt8 eventNbr,
              Bool enable )
```

This function enables the interrupt line of an Absolute Event interrupt by setting the interrupt mask of the Absolute Event with index eventNbr.

Parameters

eventNbr	The index of the Absolute Event (1..16).
enable	Enables the interrupt source if true.

gpHal_GetMeasuredSleepClockFrequency()

```
gpHal_SleepClockMeasurementStatus_t gpHal_GetMeasuredSleepClockFrequency (
              gpHal_SleepMode_t mode,
              UInt32 * frequencymHz )
```

Gets the actual (measured) deviation of a sleep clock with respect to the 32 MHz clock

This function returns whether the measurements for the requested sleep mode have been performed and what the measured frequency is.

gpHal_GetSleepMode()

```
gpHal_SleepMode_t gpHal_GetSleepMode (
              void  )
```

This function returns which sleep mode is currently set. The return mode is returned as enumerated under the enumeration gpHal_SleepMode.

gpHal_GetTime()

```
void gpHal_GetTime (
                UInt32 * pTime )
```

This function returns the current time of the GP chip in us.

Parameters

pTime	Pointer to the variable where the time will be stored.

gpHal_MonitorAbsoluteEvent()

```
gpHal_Result_t gpHal_MonitorAbsoluteEvent (
                UInt8 eventNbr,
                gpHal_AbsoluteEventDescriptor_t * pAbsoluteEvent )
```

This function returns the AbsoluteEventDescriptor_t structure of an Absolute Event.

This function needs to be used carefully, because it temporarilly disables the event and the event could be missed.

Possible results are:

- gpHal_ResultSuccess
- gpHal_ResultInvalidHandle (no Absolute Event registered at given index)

Parameters

eventNbr	The index of the Absolute Event (1..16).
pAbsoluteEvent	The pointer where the AbsoluteEventDescriptor_t structure is returned.

gpHal_MonitorExternalEvent()

```
gpHal_Result_t gpHal_MonitorExternalEvent (
                gpHal_ExternalEventDescriptor_t * pExternalEventDescriptor )
```

This funcion returns the gpHal_ExternalEventDescriptor_t structure of the External Event. The contents of the structure are only valid in case the function returns gpHal_ResultSuccess.

Possible results are:

- gpHal_ResultSuccess (valid External Event found)
- gpHal_ResultInvalidRequest (no valid External Event present)

Parameters

pExternalEventDescriptor	The pointer where the gpHal_ExternalEventDescriptor_t structure is returned.

gpHal_RefreshAbsoluteEvent()

```
void gpHal_RefreshAbsoluteEvent (
                gpHal_AbsoluteEventId_t eventNbr,
                UInt32 absTime,
                UInt8 control )
```

This function refreshes an already prepared Absolute Event. The event descriptor gpHal_AbsoluteEventDescriptor needs to be written as part of the preparation. Writing the gpHal_AbsoluteEventDescriptor_t can be done with GP_ES_WRITE_EVENT_DESCRIPTOR().

Parameters

eventNbr	The index of the Absolute Event (1..16).
absTime	The absolute execution time of the event (in us)
control	The control field of the event descriptor (see gpHal_AbsoluteEventDescriptor_t).

gpHal_RegisterAbsoluteEventCallback()

```
void gpHal_RegisterAbsoluteEventCallback (
                gpHal_AbsoluteEventCallback_t callback,
                UInt8 eventNbr )
```

This function registers the callback for an Absolute Event. The callback will be executed when the Absolute Event is triggered. The Absolute Event with the correct index needs to be enabled.

Parameters

callback	The pointer to the callback function.
eventNbr	The index of the Absolute Event (1..16).

gpHal_RegisterExternalEventCallback()

```
gpHal_ExternalEventCallback_t gpHal_RegisterExternalEventCallback (
                gpHal_ExternalEventCallback_t callback )
```

This function registers the callback for a External Event. It returns the callback that was registered earlier or NULL if none was registered. Multiple External Event handlers can be threaded by calling previously registered handler from the new handler.

The callback will be executed when the External Event is triggered. The External Event interrupt must be enabled.

Parameters

callback	The pointer to the callback function.

gpHal_ScheduleAbsoluteEvent()

```
void gpHal_ScheduleAbsoluteEvent (
                gpHal_AbsoluteEventDescriptor_t * pAbsoluteEventDescriptor,
                gpHal_AbsoluteEventId_t eventNbr )
```

This function uploads and activates an event in the GP chip Event Scheduler. To facilitate a callback on the execution of the event one must register the callback using gpHal_RegisterAbsoluteEventCallback() and enable the interrupt using gpHal_EnableAbsoluteEventCallbackInterrupt().

Parameters

pAbsoluteEventDescriptor	Pointer to the AbsoluteEventDescriptor_t structure containing the Event options.
eventNbr	The index of the Absolute Event (1..16).

gpHal_ScheduleExternalEvent()

```
void gpHal_ScheduleExternalEvent (
                gpHal_ExternalEventDescriptor_t * pExternalEventDescriptor )
```

This function uploads and activates the External Event in the GP chip Event Scheduler. To facilitate a callback on the execution of the event the callback must be registered using gpHal_RegisterExternalEventCallba and the interrupt enabled using gpHal_EnableExternalEventCallbackInterrupt().

Parameters

pExternalEventDescriptor	Pointer to the gpHal_ExternalEventDescriptor_t structure containing the Event options.

gpHal_ScheduleImmediateEvent()

```
void gpHal_ScheduleImmediateEvent (
                gpHal_EventType_t type )
```

This function uploads and activates the Relative Event in the GP chip Event Scheduler immediately.

Parameters

type	Type of event to execute without delay

gpHal_SetSleepMode()

```
gpHal_Result_t gpHal_SetSleepMode (
                gpHal_SleepMode_t mode )
```

This function sets the sleep mode of the GP chip. As enumerated under the enumeration gpHal_SleepMode the GP chip can be put into 4 different sleep modes. The desired setting can be made using this function.

Parameters

mode	The sleepmode enumerated in gpHal_SleepMode.

Returns

gpHal_ResultSuccess only if the operation was successful

gpHal_UnscheduleAbsoluteEvent()

```
gpHal_EventState_t gpHal_UnscheduleAbsoluteEvent (
            gpHal_AbsoluteEventId_t eventNbr )
```

This function disables the Absolute Event in the GP chip and returns the current EventState. Possible Event States are enumerated in the enumeration gpHal_EventState.

Parameters

eventNbr	The index of the Absolute Event (1..16).

gpHal_UnscheduleExternalEvent()

```
gpHal_Result_t gpHal_UnscheduleExternalEvent (
            void  )
```

This function disables the External Event in the GP chip.
Possible results are:

- gpHal_ResultSuccess

3.8 gpHal_HW.h File Reference

This file switches between the HW access modes (SPI, I2C, ...)

Macros

- #define GP_LOG(fmt, ...) do {} while (false)

  A macro that is used to print log messages.
- #define GP_HAL_WRITE_PROPTO(Byte, Reg, Property, Value)

  A macro to modify a property in a cached version of a GP chip register.
- #define GP_HAL_WRITE_PROP_OFFSET_AUX(Offset, Property, Value)
- #define GP_HAL_WRITE_PROP_OFFSET(Offset, Property, Value) GP_HAL_WRITE_PROP_OFFSET_AUX(Property, Value)

  A macro to write a specific field (=property) in a register, for a property that is offset based.
- #define GP_HAL_WRITE_PROP(Property, Value) GP_HAL_WRITE_PROP_OFFSET(0, Property, Value)

  A macro to write a specific field (= property) in a register .

- #define GP_HAL_UNSAFE_WRITE_PROP(Property, Value) GP_HAL_WRITE_PROP_OFFSET_AUX(0, Property, Value)
- #define **GP_HAL_TDC_ENABLE**(Property, Value)
- #define GP_HAL_READ_PROPFROM(Byte, Reg, Property)

  Read a property from a byte buffer. A macro to read a specific bit field (=property) of a byte buffer.
- #define GP_HAL_READ_PROP_OFFSET_AUX(Offset, Property) GP_HAL_BASE_READ_PROPFROM(GP_
  (Offset)) + Property##_REGISTER), Property)
- #define GP_HAL_READ_PROP_OFFSET(Offset, Property) GP_HAL_READ_PROP_OFFSET_AUX(Offset, Property)

  A macro to read a specific field (= property) of a register, for a property that is offset based.
- #define GP_HAL_READ_PROP(Property) GP_HAL_READ_PROP_OFFSET(0, Property)

  A macro to read a specific field (= property) of a register.
- #define GP_HAL_READMODIFYWRITE_PROP(Prop, Mask, Data)

  A macro to modify some bits of a specific field (= property) in a register .
- #define GP_HAL_READ_REGS16(Address, pBuffer) do { GP_HAL_READ_TWO_BYTES(Address, pBuffer); RF_TO_HOST_UINT16(pBuffer); } while(false)

  A macro to read a 16 bit value.
- #define GP_HAL_READ_REGS32(Address, pBuffer) do { GP_HAL_READ_FOUR_BYTES(Address, pBuffer); RF_TO_HOST_UINT32(pBuffer); } while(false)

  A macro to read a 32 bit value.
- #define GP_HAL_READ_REGS64(Address, pBuffer) do { GP_HAL_READ_EIGHT_BYTES(Address, pBuffer); RF_TO_HOST_UINT64(pBuffer); } while(false)

  A macro to read a 64 bit value.
- #define GP_HAL_WRITE_REGS16(Address, pBuffer) do { HOST_TO_RF_UINT16(pBuffer); GP_HAL_WRITE_TWO_BYTES(Address, pBuffer); RF_TO_HOST_UINT16(pBuffer); } while(false)

  A macro to write a 16 bit value.
- #define GP_HAL_WRITE_REGS24(Address, pBuffer) do { HOST_TO_RF_UINT32(pBuffer); GP_HAL_WRITE_THREE_BYTES(Address, pBuffer); RF_TO_HOST_UINT32(pBuffer); } while(false)

  A macro to write a 24 bit value.
- #define GP_HAL_WRITE_REGS32(Address, pBuffer) do { HOST_TO_RF_UINT32(pBuffer); GP_HAL_WRITE_FOUR_BYTES(Address, pBuffer); RF_TO_HOST_UINT32(pBuffer); } while(false)

  A macro to write a 32 bit value.
- #define GP_HAL_WRITE_REGS64(Address, pBuffer) do { HOST_TO_RF_UINT64(pBuffer); GP_HAL_WRITE_EIGHT_BYTES(Address, pBuffer); RF_TO_HOST_UINT64(pBuffer); } while(false)

  A macro to write a 64 bit value.
- #define ENABLE_GP_GLOBAL_INT() HAL_ENABLE_GLOBAL_INT()

  Enable the interrupts on the micro processor.
- #define DISABLE_GP_GLOBAL_INT() HAL_DISABLE_GLOBAL_INT()

  Disables the interrupts on the micro processor.

3.8.1 Macro Definition Documentation

GP_HAL_READ_PROP

```
#define GP_HAL_READ_PROP(
                Property ) GP_HAL_READ_PROP_OFFSET(0, Property)
```

A macro to read a specific field (= property) of a register. Register definitions are needed to preprocess these instructions.

Parameters

Property	Property name (only use with regprop definition).

GP_HAL_READ_PROP_OFFSET

```
#define GP_HAL_READ_PROP_OFFSET(
                Offset,
                Property ) GP_HAL_READ_PROP_OFFSET_AUX(Offset, Property)
```

A macro to read a specific field (= property) of a register, for a property that is offset based. Register definitions are needed to preprocess these instructions.

Parameters

Offset	Base address of the entry to which the property applies.
Property	Property name (only use with regprop definition).

GP_HAL_READ_PROP_OFFSET_AUX

```
#define GP_HAL_READ_PROP_OFFSET_AUX(
                Offset,
                Property ) GP_HAL_BASE_READ_PROPFROM(GP_HAL_READ_REG(((UInt16) (Offset)) + Property##_REGISTER),
Property)
```

main code in auxiliary macro to make property substitution work as intended

GP_HAL_READ_PROPFROM

```
#define GP_HAL_READ_PROPFROM(
                Byte,
                Reg,
                Property )
```

Value:

```
( GP_HAL_CHECK_PROP_MATCHES_REG(Property, Reg), \
      GP_HAL_BASE_READ_PROPFROM(Byte, Property)     \
      )
```

Parameters

Byte	The byte buffer.
Reg	The register of the property to access. This parameter is used in order to check the consistency between the byte buffer and the accessed property.
Property	Property name (only use with regprop definition).

GP_HAL_READ_REGS16

```
#define GP_HAL_READ_REGS16(
                Address,
                pBuffer ) do { GP_HAL_READ_TWO_BYTES(Address, pBuffer); RF_TO_HOST_UINT16(pBuffer); } while(false)
```

A macro to read a 16 bit value. The macro takes care of the endianness of the host processor.

Parameters

Address	The address of the value to read
pBuffer	The buffer where the data is returned.

GP_HAL_READ_REGS32

```
#define GP_HAL_READ_REGS32(
                Address,
                pBuffer ) do { GP_HAL_READ_FOUR_BYTES(Address, pBuffer); RF_TO_HOST_UINT32(pBuffer); } while(false)
```

A macro to read a 32 bit value. The macro takes care of the endianness of the host processor.

Parameters

Address	The address of the value to read
pBuffer	The buffer where the data is returned.

GP_HAL_READ_REGS64

```
#define GP_HAL_READ_REGS64(
                Address,
                pBuffer ) do { GP_HAL_READ_EIGHT_BYTES(Address, pBuffer); RF_TO_HOST_UINT64(pBuffer); }
while(false)
```

A macro to read a 64 bit value. The macro takes care of the endianness of the host processor.

Parameters

Address	The address of the value to read
pBuffer	The buffer where the data is returned.

GP_HAL_READMODIFYWRITE_PROP

```
#define GP_HAL_READMODIFYWRITE_PROP(
                Prop,
                Mask,
                Data )
```

Value:

```
do              \
```

```
{                                                                   \
    UInt8 newData = (GP_HAL_READ_PROP(Prop) & ~(Mask)) | ((Data) & (Mask));    \
    GP_HAL_WRITE_PROP(Prop, newData);                               \
} while (false)
```

A macro to modify some bits in a specific field (= property) in a register. Register definitions are needed to preprocess these instructions .

Parameters

Prop	Property name (only use with regprop definition).
Mask	The read-modify-write mask.
Data	The Value to write.

GP_HAL_TDC_ENABLE

```
#define GP_HAL_TDC_ENABLE(
                Property,
                Value )
```

Value:

```
do                                                      \
    {                                                       \
        GP_HAL_WRITE_PROP(Property, Value);                 \
        GP_HAL_UNSAFE_WRITE_PROP(Property##_TDC_ENABLE, true);  \
    } while (false)
```

GP_HAL_UNSAFE_WRITE_PROP

```
#define GP_HAL_UNSAFE_WRITE_PROP(
                Property,
                Value ) GP_HAL_WRITE_PROP_OFFSET_AUX(0, Property, Value)
```

internal

GP_HAL_WRITE_PROP

```
#define GP_HAL_WRITE_PROP(
                Property,
                Value ) GP_HAL_WRITE_PROP_OFFSET(0, Property, Value)
```

A macro to write a specific field (= property) in a register. Register definitions are needed to preprocess these instructions .

Parameters

Property	Property name (only use with regprop definition).
Value	Value to be set.

GP_HAL_WRITE_PROP_OFFSET

```
#define GP_HAL_WRITE_PROP_OFFSET(
                Offset,
                Property,
                Value ) GP_HAL_WRITE_PROP_OFFSET_AUX(Offset, Property, Value)
```

A macro to write a specific field (= property) in a register, for a property that is offset based. Register definitions are needed to preprocess these instructions. The register must be relative to a base address: for use with PBM, ES and other repeating structures.

Parameters

Offset	Base address of the entry to which the property applies.
Property	Property name (only use with regprop definition).
Value	Value to be set.

GP_HAL_WRITE_PROP_OFFSET_AUX

```
#define GP_HAL_WRITE_PROP_OFFSET_AUX(
                Offset,
                Property,
                Value )
```

Value:

```
do {                                                                    \
        COMPILE_TIME_ASSERT(Property##_WRITABLE==1);                    \
        /* check if the value fits in the property*/                    \
        GP_ASSERT_EARLY(((Value) & (Property##_MASK >> Property##_LSB)) == (Value)); \
        if (Property##_READABLE && !(GP_HAL_IS_ONLY_PROP_IN_REG(Property))) \
        {                                                               \
            GP_HAL_READMODIFYWRITE_REG( ((UInt16) (Offset)) + Property##_REGISTER,  Property##_MASK,
      GP_HAL_VAL2PLACEDVAL(Value, Property) );\
        }                                                               \
        else                                                            \
        {  /* not readable/write to whole register*/   \
            GP_HAL_WRITE_REG( ((UInt16) (Offset)) + Property##_REGISTER, GP_HAL_VAL2PLACEDVAL(Value,
      Property));    \
        }                                                               \
    } while (0)
```

internal usage, main code in auxiliary macro to make property substitution work as intended

GP_HAL_WRITE_PROPTO

```
#define GP_HAL_WRITE_PROPTO(
                Byte,
                Reg,
                Property,
                Value )
```

Value:

```
do {                                                \
        GP_HAL_CHECK_PROP_MATCHES_REG(Property, Reg);       \
        GP_HAL_BASE_WRITE_PROPTO(Byte, Property, Value);    \
    } while (0)
```

A macro to modify a property in a cached version of a GP chip register. This cached memory has to written to the real register to have any effect. Use this to change several properties in the same register in an efficient manner.

Parameters

Byte	the cached version register in memory.
Property	Property name (only use with regprop definition).
Value	Value to be set.
Reg	The GP chip register name that the Byte parameter represents.

GP_HAL_WRITE_REGS16

```
#define GP_HAL_WRITE_REGS16(
                Address,
                pBuffer ) do { HOST_TO_RF_UINT16(pBuffer); GP_HAL_WRITE_TWO_BYTES(Address, pBuffer); RF_TO_HOST_UINT16(pBuf
} while(false)
```

A macro to write a 16 bit value. The macro takes care of the endianness of the host processor.

Parameters

Address	The address of the value to write
pBuffer	The buffer where the data to write is read from.

GP_HAL_WRITE_REGS24

```
#define GP_HAL_WRITE_REGS24(
                Address,
                pBuffer ) do { HOST_TO_RF_UINT32(pBuffer); GP_HAL_WRITE_THREE_BYTES(Address, pBuffer); RF_TO_HOST_UINT32(pB
} while(false)
```

A macro to write a 24 bit value. The macro takes care of the endianness of the host processor. The argument should be a 32-bit value.

Parameters

Address	The address of the value to write
pBuffer	The buffer where the data to write is read from.

GP_HAL_WRITE_REGS32

```
#define GP_HAL_WRITE_REGS32(
                Address,
                pBuffer ) do { HOST_TO_RF_UINT32(pBuffer); GP_HAL_WRITE_FOUR_BYTES(Address, pBuffer); RF_TO_HOST_UINT32(pBu
} while(false)
```

A macro to write a 32 bit value. The macro takes care of the endianness of the host processor.

Parameters

Address	The address of the value to write

Parameters

pBuffer	The buffer where the data to write is read from.

GP_HAL_WRITE_REGS64

```
#define GP_HAL_WRITE_REGS64(
                Address,
                pBuffer ) do { HOST_TO_RF_UINT64(pBuffer); GP_HAL_WRITE_EIGHT_BYTES(Address, pBuffer); RF_TO_HOST_UINT64(pB
} while(false)
```

A macro to write a 64 bit value. The macro takes care of the endianness of the host processor.

Parameters

Address	The address of the value to write
pBuffer	The buffer where the data to write is read from.

GP_LOG

```
#define GP_LOG(
                fmt,
                ...  ) do {} while (false)
```

A macro that is used to print log messages inside of the GPHAL. This macro is inserted in the GPHAL where the log messages are useful. By default, this macro is empty and it is up to the GPHAL user to implement it.

Parameters

fmt	The format string of the print message.
...	List of parameters for the print message. The first parameter of this list should be the length of the parameters in bytes.

3.9 gpHal_HW_MM.h File Reference

Macros

- #define **GP_COMPONENT_ID** GP_COMPONENT_ID_GPHAL
- #define **GP_COMPONENT_ID_DEFINED_IN_HEADER**
- #define WBPTR(x) ((volatile UInt8 ∗) (x))

  Read and write register access macros.
- #define **REG**(x) (∗WBPTR((UIntPtr) (x)))
- #define **GP_HAL_READ_REG**(Register) REG(Register)
- #define **GP_HAL_WRITE_REG**(Register, Data) REG(Register) = (Data)
- #define **GP_HAL_HAVE_READ_TWO_BYTES**
- #define **GP_HAL_HAVE_READ_FOUR_BYTES**

- #define **GP_HAL_HAVE_READ_EIGHT_BYTES**
- #define **GP_HAL_HAVE_WRITE_TWO_BYTES**
- #define **GP_HAL_HAVE_WRITE_FOUR_BYTES**
- #define **GP_HAL_HAVE_WRITE_EIGHT_BYTES**
- #define **BLKCPR** "blkcp.r "
- #define **BLKCPI** "blkcp.i "
- #define **WRITE_N_BYTES**(Address, pData, type) ∗ ((volatile type ∗) (Address)) = ∗((type ∗) (unsigned char ∗) (pData))
- #define **GP_HAL_WRITE_BYTE_STREAM**(Address, pBuffer, Length)
- #define **READ_N_BYTES**(Address, pData, type) ∗((type ∗) (unsigned char ∗) (pData)) = ∗ ((volatile type ∗) (Address))
- #define **GP_HAL_READ_BYTE_STREAM**(Address, pBuffer, Length)

Typedefs

- typedef UIntPtr **gpHal_Address_t**

Functions

- ALWAYS_INLINE void **GP_HAL_READMODIFYWRITE_REG** (UIntPtr Register, UInt16 Mask, UInt16 Data)
- ALWAYS_INLINE void **GP_HAL_READ_TWO_BYTES** (UIntPtr Address, UInt16 ∗pData)
- ALWAYS_INLINE void **GP_HAL_READ_THREE_BYTES** (UIntPtr Address, UInt32 ∗pData)
- ALWAYS_INLINE void **GP_HAL_READ_FOUR_BYTES** (UIntPtr Address, UInt32 ∗pData)
- ALWAYS_INLINE void **GP_HAL_READ_EIGHT_BYTES** (UIntPtr Address, UInt64Struct_t ∗pData)
- ALWAYS_INLINE void **GP_HAL_WRITE_TWO_BYTES** (UIntPtr Address, const UInt16 ∗pData)
- ALWAYS_INLINE void **GP_HAL_WRITE_THREE_BYTES** (UIntPtr Address, const UInt32 ∗pData)
- ALWAYS_INLINE void **GP_HAL_WRITE_FOUR_BYTES** (UIntPtr Address, const UInt32 ∗pData)
- ALWAYS_INLINE void **GP_HAL_WRITE_EIGHT_BYTES** (UIntPtr Address, const UInt64Struct_t ∗pData)
- ALWAYS_INLINE void **VOLATILE_MEMCPY_VAR** (void ∗dst, const void ∗src, UInt16 Length)
- ALWAYS_INLINE void **VOLATILE_MEMCPY_FIX** (void ∗dst, const void ∗src, UInt16 Length)
- ALWAYS_INLINE void **VOLATILE_MEMCPY** (void ∗dst, const void ∗src, UInt16 Length)
- ALWAYS_INLINE void **BLOCKREAD** (UIntPtr Address, void ∗pBuffer, UInt16 Length)
- ALWAYS_INLINE void **BLOCKWRITE** (UIntPtr Address, const void ∗pBuffer, UInt16 Length)

3.9.1 Detailed Description

This file contains the functions dependent on the choice of MCU : memory mapped registers, Interrupt handling,...

3.9.2 Macro Definition Documentation

GP_HAL_READ_BYTE_STREAM

```
#define GP_HAL_READ_BYTE_STREAM(
                Address,
                pBuffer,
                Length )
```

Value:

```
do                                                  \
    {                                               \
        if (__builtin_constant_p(Length))           \
        {                                           \
            if (Length == 2)                        \
            {                                       \
                READ_N_BYTES(Address, pBuffer, UInt16); \
            }                                       \
            else if (Length == 4)                   \
            {                                       \
                READ_N_BYTES(Address, pBuffer, UInt32); \
            }                                       \
            else if (Length == 8)                   \
            {                                       \
                READ_N_BYTES(Address, pBuffer, UInt64); \
            }                                       \
            else                                    \
            {                                       \
                BLOCKREAD(Address, pBuffer, Length);    \
            }                                       \
        }                                           \
        else                                        \
        {                                           \
            BLOCKREAD(Address, pBuffer, Length);    \
        }                                           \
    } while (false)
```

GP_HAL_WRITE_BYTE_STREAM

```
#define GP_HAL_WRITE_BYTE_STREAM(
                Address,
                pBuffer,
                Length )
```

Value:

```
do                                                  \
    {                                               \
        if (__builtin_constant_p(Length))           \
        {                                           \
            if (Length == 2)                        \
            {                                       \
                WRITE_N_BYTES(Address, pBuffer, UInt16);    \
            }                                       \
            else if (Length == 4)                   \
            {                                       \
                WRITE_N_BYTES(Address, pBuffer, UInt32);    \
            }                                       \
            else if (Length == 8)                   \
            {                                       \
                WRITE_N_BYTES(Address, pBuffer, UInt64);    \
            }                                       \
            else                                    \
            {                                       \
                BLOCKWRITE(Address, pBuffer, Length);       \
            }                                       \
        }                                           \
        else                                        \
        {                                           \
            BLOCKWRITE(Address, pBuffer, Length);           \
        }                                           \
    } while (false)
```

WBPTR

```
#define WBPTR(
              x ) ((volatile UInt8 *) (x))
```

These are the raw macros for accessing GP chip registers.

Depending on the processor the macros are used on, access to the registers is direct (embedded processor) or using SPI/I2C (external processor).

3.10 gpHal_HW_MSI.h File Reference

This file contains the functions dependent on the choice of MCU : MSI, Interrupts,... This particular file does it using the MSI protcol + our own hal functionality.

Macros

- #define HAL_GP_HW_INIT() { HAL_WKUP_START(); HAL_SET_RESET_INACTIVE(); GP_MSI_INIT(); CONFIG_INTERRUPT_LINE(); }

  Micro processor dependent macro to initialize communication interface and reset,interrupt and wakeup pins.
- #define **HAL_ISR_RADIO_INTERRUPT_START**(x)
- #define HAL_GP_INTERRUPT_INIT()

  Intializes the interrupt line at micro processor side.
- #define HAL_GP_SET_WAKEUP() HAL_SET_WKUP_ACTIVE()

  Puts a high signal on the WKUP pin of the GP chip.
- #define HAL_GP_CLR_WAKEUP() HAL_SET_WKUP_INACTIVE()

  Puts a low signal on the WKUP pin of the GP chip.
- #define **HAL_GP_PULSE_WAKEUP**()
- #define HAL_GP_SET_RESET(level) HAL_SET_RESET_ACTIVE(level)

  Sets the reset of the GP chip active.
- #define HAL_GP_CLR_RESET() HAL_SET_RESET_INACTIVE()

  Puts the reset of the GP chip inactive.

3.10.1 Macro Definition Documentation

HAL_GP_CLR_RESET

```
#define HAL_GP_CLR_RESET( ) HAL_SET_RESET_INACTIVE()
```

This macro releases the reset of the GP chip.

HAL_GP_CLR_WAKEUP

```
#define HAL_GP_CLR_WAKEUP( ) HAL_SET_WKUP_INACTIVE()
```

Wakeup of the GP chip is done on a falling edge of the WKUP pin.

HAL_GP_INTERRUPT_INIT

```
#define HAL_GP_INTERRUPT_INIT( )
```

Value:

```
do {                                                \
        HAL_ISR_RADIO_INTERRUPT_START( gpHal_ISR_Interrupt );  \
        HAL_RADIO_INTERRUPT_START( gpHal_Interrupt );  \
    } while(0)
```

This macro configures the pin on which the GP chip INTOUTn pin is connected as an input pin and configures the interrupt attached to this pin as a low level interrupt.

HAL_GP_PULSE_WAKEUP

```
#define HAL_GP_PULSE_WAKEUP( )
```

Value:

```
do{                     \
    HAL_GP_SET_WAKEUP();\
    HAL_GP_CLR_WAKEUP();\
}while(false);
```

HAL_GP_SET_RESET

```
#define HAL_GP_SET_RESET(
                level ) HAL_SET_RESET_ACTIVE(level)
```

This macro sets and holds the GP chip in a reset state.

HAL_GP_SET_WAKEUP

```
#define HAL_GP_SET_WAKEUP( ) HAL_SET_WKUP_ACTIVE()
```

Wakeup of the GP chip is done on a falling edge of the WKUP pin.

3.11 gpHal_kx_MSI.h File Reference

This file defines the SPI protocol as implemented by k7.

Macros

- #define GP_HAL_MSI_MAX_BLOCK_SIZE 127

  Maximum amount of bytes to be transferred in a MSI read/write block.
- #define GP_MSI_READCMD 0x00

  GP short MSI read command.
- #define GP_MSI_WRITECMD 0x40

  GP short MSI write command.
- #define GP_MSI_BLOCKCMD 0x80

  GP short MSI block command.
- #define GP_MSI_MASKEDWRITECMD 0xC0

  GP short MSI masked write command.
- #define GP_MSI_BLOCKREADCMD GP_MSI_BLOCKCMD

  GP short MSI read option for block command.

- #define **GP_MSI_BLOCKREADEXTCMD** 0x00
- #define GP_MSI_BLOCKWRITECMD GP_MSI_BLOCKCMD

  GP short MSI write option for block command.
- #define **GP_MSI_BLOCKWRITEEXTCMD** 0x80
- #define GP_MSI_WRITECONFIRM 0x01

  GP MSI write returncode.
- #define GP_MSI_READCONFIRM 0x03

  GP MSI read returncode.
- #define GP_MSI_WRITEBLOCKCONFIRM 0x05

  GP MSI write block returncode.
- #define GP_MSI_READBLOCKCONFIRM 0x07

  GP MSI read block returncode.
- #define GP_MSI_IS_VALID_ADDR(addr) (((addr) >= 0) && ((addr) < 0x1000000))
- #define GP_MSI_MOSI_BYTECMD(msi_errno, cmd, a2, a1, a0)
- #define GP_MSI_MOSI_BLOCKCMD(msi_errno, cmd, len, a2, a1, a0)
- #define GP_MSI_MOSI_MASKEDWRITE(msi_errno, mask, a2, a1, a0)

3.11.1 Macro Definition Documentation

GP_MSI_IS_VALID_ADDR

```
#define GP_MSI_IS_VALID_ADDR(
                addr ) (((addr) >= 0) && ((addr) < 0x1000000))
```

check if address fits into protocol. two bits for cmd, 14 for address

GP_MSI_MOSI_BLOCKCMD

```
#define GP_MSI_MOSI_BLOCKCMD(
                msi_errno,
                cmd,
                len,
                a2,
                a1,
                a0 )
```

Value:

```
do {                                                    \
        GP_MSI_MOSI_BYTE(msi_errno, GP_MSI_BLOCKCMD | (a2));    \
        GP_MSI_MOSI_BYTE(msi_errno, a1);                \
        GP_MSI_MOSI_BYTE(msi_errno, a0);                \
        GP_MSI_MOSI_BYTE(msi_errno, cmd | (len));       \
    } while (false)
```

Master -> Slave one block command (without data) ah: address high, msb of address al: address low, lsb of address

GP_MSI_MOSI_BYTECMD

```
#define GP_MSI_MOSI_BYTECMD(
                msi_errno,
                cmd,
                a2,
                a1,
                a0 )
```

Value:

```
do {                                            \
        GP_MSI_MOSI_BYTE(msi_errno, (cmd) | (a2));   \
        GP_MSI_MOSI_BYTE(msi_errno, a1);             \
        GP_MSI_MOSI_BYTE(msi_errno, a0);             \
    } while (false)
```

Master -> Slave one byte command (without data) ah: address high, msb of address al: address low, lsb of address

GP_MSI_MOSI_MASKEDWRITE

```
#define GP_MSI_MOSI_MASKEDWRITE(
                msi_errno,
                mask,
                a2,
                a1,
                a0 )
```

Value:

```
do {                                                    \
        GP_MSI_MOSI_BYTE(msi_errno, GP_MSI_MASKEDWRITECMD | (a2)); \
        GP_MSI_MOSI_BYTE(msi_errno, a1);                    \
        GP_MSI_MOSI_BYTE(msi_errno, a0);                    \
        GP_MSI_MOSI_BYTE(msi_errno, mask);                  \
    } while (false)
```

Master -> Slave one masked write command (without data) mask: used mask ah: address high, msb of address al: address low, lsb of address

3.12 gpHal_MAC.h File Reference

This file contains all the functions needed for MAC functionality.

Data Structures

- struct gpHal_DataReqOptions_t

  These options dictate the way a data packet should be transmitted.

Macros

- #define GPHAL_TTL_START_VALUE 20

  Default value of the time to live setting of a PBM entry.
- #define GP_HAL_MULTICHANNEL_MAX_CHANNELS 3

  Maximum amount of channels used to do multi channel retries.
- #define GP_HAL_MAX_NUM_OF_SLOTS 6

  Maximum number of slots (simultaneously active RX channels)

- #define GP_HAL_MULTICHANNEL_INVALID_CHANNEL 0xFF

  Define to ignore channel used in the multiChannel options struct.
- #define GPHAL_ACK_REQ_LSB 5

  Offset in IEEE packet to check if Ack Request is required.
- #define **GPHAL_POLL_REQ_MAX_WAIT_TIME** 0x7C2
- #define **GPHAL_MAX_15_4_FCS_LENGTH** 2
- #define **GPHAL_MAX_15_4_PACKET_LENGTH** 127UL
- #define GPHAL_MAX_15_4_PACKET_LENGTH_NO_FCS (GPHAL_MAX_15_4_PACKET_LENGTH - GPHAL_MAX_15_4_FCS_LENGTH)

  Maximum length of a payload that can be written into a PBM entry.
- #define gpHal_EnablePrimitiveCallbackInterrupt(enable) GP_HAL_ENABLE_PIO_INT(enable)

  Enables the interrupt line of the MAC and SEC operations.
- #define gpHal_EnableEmptyQueueCallbackInterrupt(enable) GP_HAL_ENABLE_EMPTY_QUEUE_CALLBAC

  Enables the interrupt line of the Empty Queue interrupt.
- #define gpHal_EnableBusyTXCallbackInterrupt(enable) GP_HAL_ENABLE_BUSY_TX_CALLBACK_INTERRU

  Enables the interrupt line of the BusyTX interrupt.
- #define gpHal_WriteDataInPbm(address, pData, length, offset) GP_HAL_WRITE_DATA_IN_PBM(address,pD

  Writes data in the specified pbm address.
- #define **gpHal_CalculateTxPbmDataBufferAddress**(pbmEntry) GP_HAL_CALCULATE_TX_PBM_DATA_BU
- #define **gpHal_CheckPbmValid**(pbmEntry) GP_HAL_CHECK_PBM_VALID(pbmEntry)
- #define gpHal_GetPipMode() GP_HAL_GET_PIP_MODE()

  Returns the number packet in packet mode.
- #define gpHal_GetAddressRecognition() GP_HAL_GET_ADDRESS_RECOGNITION()

  Returns the addressRecognition flag.
- #define gpHal_SetFrameTypeFilterMask(bitmap) GP_HAL_SET_FRAME_TYPE_FILTER_MASK((bitmap))

  Sets the HW FrameType FilterMask.
- #define gpHal_GetFrameTypeFilterMask() GP_HAL_GET_FRAME_TYPE_FILTER_MASK()

  Returns the HW FrameType FilterMask.
- #define gpHal_GetRxOnWhenIdle() GP_HAL_GET_RX_ON_WHEN_IDLE()

  Returns the RxOnWhenIdle flag.

Typedefs

- typedef void(∗ gpHal_DataIndicationCallback_t) (gpPd_Loh_t pdLoh, gpHal_RxInfo_t ∗rxInfo)

  The gpHal_DataIndicationCallback_t callback type definition defines the callback prototype of the DataIndication.
- typedef void(∗ gpHal_SnifferDataIndicationCallback_t) (gpPd_Loh_t pdLoh, gpHal_RxInfo_t ∗rxInfo)

  The gpHal_SnifferDataIndicationCallback_t callback type definition defines the callback prototype of the SnifferDataIndication.
- typedef void(∗ gpHal_DataConfirmCallback_t) (UInt8 status, gpPd_Loh_t pdLoh, UInt8 lastChannelUsed)

  The gpHal_DataConfirmCallback_t callback type definition defines the callback prototype of the DataConfirm.
- typedef void(∗ gpHal_EDConfirmCallback_t) (UInt16 channelMask, UInt8 ∗protoED)

  The gpHal_EDConfirmCallback_t callback typedef defines the callback prototype of the EDConfirm.
- typedef void(∗ gpHal_BusyTXCallback_t) (void)

The gpHal_BusyTXCallback_t callback type definition defines the callback prototype of the BusyTX interrupt.

- typedef void(∗ gpHal_EmptyQueueCallback_t) (void)

  The gpHal_EmptyQueueCallback_t callback type definition defines the callback prototype of the EmptyQueue interrupt.
- typedef void(∗ gpHal_CmdDataReqCallback_t) (void)

  The gpHal_CmdDataReqCallback_t callback type definition defines the callback prototype of the Cmd Data Req interrupt.
- typedef void(∗ **gpHal_MacFrameQueued_t**) (void)
- typedef void(∗ **gpHal_MacFrameUnqueued_t**) (void)

Functions

- gpHal_Result_t gpHal_SetMacRxMode (Bool enableMultiStandard, Bool enableMultiChannel, Bool enableHighSensitivity)

  This function sets the rx mode configuration for the MAC part of the radio.
- void gpHal_GetMacRxMode (Bool ∗enableMultiStandard, Bool ∗enableMultiChannel, Bool ∗enableHighSensitivity)

  This function gets the rx mode configuration for the MAC part of the radio.
- void gpHal_RegisterDataConfirmCallback (gpHal_DataConfirmCallback_t callback)

  Registers the callback for a DataConfirm.
- void gpHal_RegisterDataIndicationCallback (gpHal_DataIndicationCallback_t callback)

  Registers the callback for a DataIndication.
- void gpHal_RegisterEDConfirmCallback (gpHal_EDConfirmCallback_t callback)

  Registers the callback for a EDConfirm.
- void gpHal_RegisterBusyTXCallback (gpHal_BusyTXCallback_t callback)

  Registers the callback for a BusyTX interrupt.
- void gpHal_RegisterEmptyQueueCallback (gpHal_EmptyQueueCallback_t callback)

  Registers the callback for Empty Queue interrupt.
- void gpHal_RegisterCmdDataReqConfirmCallback (gpHal_CmdDataReqCallback_t callback)

  Registers the callback for the reception of a Cmd Data Req interrupt.
- gpHal_Result_t gpHal_DataRequest (gpHal_DataReqOptions_t ∗dataReqOptions, gpPad_Handle_t padHandle, gpPd_Loh_t pdLoh)

  Start a data transmission.
- void **gpHal_FillInTxOptions** (UInt8 pbmHandle, gpPad_Attributes_t ∗pOptions)
- UInt8 gpHal_GetRxChannel (gpHal_SourceIdentifier_t srcId)

  Returns the listening channel currently used.
- void gpHal_SetDefaultTransmitPowers (gpHal_TxPower_t ∗pDefaultTransmitPowerTable)

  Configure the default transmit power for each channel.
- gpHal_TxPower_t gpHal_GetDefaultTransmitPower (gpHal_Channel_t channel)

  Get the current default transmit power for the specified channel.
- void gpHal_SetCCAThreshold (void)

  Set the CCA Threshold setting.
- void gpHal_SetPipMode (Bool pipmode)

  Sets the Rx packet in packet mode.
- gpHal_Result_t **gpHal_GetRadioState** (void)
- gpHal_Result_t gpHal_EDRequest (UInt32 time_us, UInt16 channelMask)

  Performs a Energy Detect request according to the IEEE802.15.4 spec.

- void gpHal_SetPanId (UInt16 panId, gpHal_SourceIdentifier_t srcId)
  Set the PAN ID.
- UInt16 gpHal_GetPanId (gpHal_SourceIdentifier_t srcId)
  Returns the PAN ID stored.
- void gpHal_SetShortAddress (UInt16 shortAddress, gpHal_SourceIdentifier_t srcId)
  Sets the Short Address.
- UInt16 gpHal_GetShortAddress (gpHal_SourceIdentifier_t srcId)
  Returns the ShortAddress.
- void gpHal_SetExtendedAddress (MACAddress_t *pExtendedAddress, gpHal_SourceIdentifier_t srcId)
  Set the Extended Address.
- void gpHal_GetExtendedAddress (MACAddress_t *pExtendedAddress, gpHal_SourceIdentifier_t srcId)
  Returns the ExtendedAddress stored.
- void gpHal_ResetExtendedAddress (gpHal_SourceIdentifier_t srcId)
  Resets the ExtendedAddress to its factory value.
- void gpHal_SetCoordExtendedAddress (MACAddress_t *pCoordExtendedAddress)
  Set the Coordinator Address.
- void **gpHal_SetCoordShortAddress** (UInt16 shortCoordAddress)
- void gpHal_SetPanCoordinator (Bool panCoordinator)
  Set the pan coordinator property.
- Bool gpHal_GetPanCoordinator (void)
  Returns the pan coordinator property of this device.
- void gpHal_SetAddressRecognition (Bool enable, Bool panCoordinator)
  Enables/Disables Address Recognition.
- void gpHal_SetBeaconSrcPanChecking (Bool enable)
  Sets the property for filtering beacons based on src pan.
- Bool gpHal_GetBeaconSrcPanChecking (void)
  Gets the property for filtering beacons based on src pan.
- void gpHal_SetRxOnWhenIdle (gpHal_SourceIdentifier_t srcId, Bool flag, UInt8 channel)
  Sets the RxOnWhenIdle flag.
- void gpHal_SetAutoAcknowledge (Bool flag)
  Sets the Auto Acknowledge flag.
- Bool gpHal_GetAutoAcknowledge (void)
  Returns the AutoAcknowledge flag.
- void gpHal_SetTimedMode (Bool timedMode)
  Sets the chip to timed MAC mode.
- UInt8 gpHal_ConvertProtoEDToProtoRSSI (UInt8 protoED)
  Calculate the protoRSSI from the protoED returned by the data indication handler.
- UInt8 gpHal_CalculateED (UInt8 protoED)
  Calculate the ED value from the protoED returned by the ED scan handler, values are conform the ZIP phy testspec.
- Bool gpHal_CheckNoLock (void)
  Checks if a NO LOCK was triggerd by the radio.
- void gpHal_SetPromiscuousMode (Bool flag)
  Enables the promiscuous mode.
- Bool gpHal_GetPromiscuousMode (void)

Returns promiscuous mode state.

- void gpHal_SetFramePendingAckDefault (Bool enable)

  Set the default ack frame pending bit.
- Bool gpHal_GetFramePendingAckDefault (void)

  Returns the default ack frame pending bit.
- gpHal_TxPower_t gpHal_GetLastUsedTxPower (void)

  Get the last used chip transmit power.
- void gpHal_RegisterMacFrameQueuedCallback (gpHal_MacFrameQueued_t callback)

  Reset the history of the Tx power compensation.
- void **gpHal_RegisterMacFrameUnqueuedCallback** (gpHal_MacFrameUnqueued_t callback)
- Bool **gpHal_IsMacQueueEmpty** (void)
- UInt8 gpHal_GetAvailableSrcIds (void)

  This function gets the number of channels which can be used by different Zigbee/MAC stacks simultaniously.
- void **gpHal_MacSetMaxTransferTime** (UInt32 MacMaxTransferTime)

Variables

- UInt8 **gpHal_MacState**

gpHal_MacScenario_t

- #define **gpHal_MacDefault** 0x0
- #define gpHal_MacPollReq 0x1
- #define gpHal_MacTimedTx 0x2
- #define gpHal_MacManualCrc 0x3
- #define gpHal_MacManualCrc_NoRetries 0x4
- typedef UInt8 gpHal_MacScenario_t

  The gpHal_MacScenario_t type defines the Mac Scenario as defined in the databook.

gpHal_SourceIdentifier_t

- #define gpHal_SourceIdentifier_0 0x0

  Identifier for first Pan (pan 0)
- #define gpHal_SourceIdentifier_1 0x1

  Identifier for second Pan (pan 1)
- #define gpHal_SourceIdentifier_2 0x2

  Identifier for third Pan (pan 2)
- #define gpHal_SourceIdentifier_Inv 0xFF

  Identifier for invalid value.
- typedef UInt8 gpHal_SourceIdentifier_t

  A source identifier refers to a group of settings (address, PAN, channel).

3.12.1 Macro Definition Documentation

gpHal_EnableBusyTXCallbackInterrupt

```
#define gpHal_EnableBusyTXCallbackInterrupt(
                enable ) GP_HAL_ENABLE_BUSY_TX_CALLBACK_INTERRUPT(enable)
```

This function sets the interrupt mask of the BusyTX interrupt.

Parameters

enable	Enables the interrupt source if true.

gpHal_EnableEmptyQueueCallbackInterrupt

```
#define gpHal_EnableEmptyQueueCallbackInterrupt(
                enable ) GP_HAL_ENABLE_EMPTY_QUEUE_CALLBACK_INTERRUPT(enable)
```

This function sets the interrupt mask of the Empty Queue interrupt.

Parameters

enable	Enables the interrupt source if true.

gpHal_EnablePrimitiveCallbackInterrupt

```
#define gpHal_EnablePrimitiveCallbackInterrupt(
                enable ) GP_HAL_ENABLE_PIO_INT(enable)
```

This function sets the interrupt mask of the PIO block.

Parameters

enable	Enables the interrupt source if true.

gpHal_GetFrameTypeFilterMask

```
#define gpHal_GetFrameTypeFilterMask( ) GP_HAL_GET_FRAME_TYPE_FILTER_MASK()
```

This function returns the Frame Type filter mask set in HW. Frametypes which have their mask bit set will be filtered out.

Returns

bitmap The FrameType FilterMask.

gpHal_MacManualCrc

```
#define gpHal_MacManualCrc 0x3
```

gpHal_MacManualCrc Mac scenario to manually set a CRC checksum (a corrupt one if applicable for the test)

gpHal_MacManualCrc_NoRetries

```
#define gpHal_MacManualCrc_NoRetries 0x4
```

gpHal_MacManualCrc Mac scenario to manually set a CRC checksum (a corrupt one if applicable for the test), and force the retries tp 0

gpHal_MacPollReq

```
#define gpHal_MacPollReq 0x1
```

gpHal_MacSPollReq Mac Scenario to send a Poll Req

gpHal_MacTimedTx

```
#define gpHal_MacTimedTx 0x2
```

gpHal_MacTimedTx Put frame on timed TX queue; will be sent at next event gpHal_EventTypeTXPacket

gpHal_SetFrameTypeFilterMask

```
#define gpHal_SetFrameTypeFilterMask(
            bitmap ) GP_HAL_SET_FRAME_TYPE_FILTER_MASK((bitmap))
```

This function sets the Frame Type filter mask used by HW

Packets can be filtered out based on their frametype (BCN, DATA, CMD, RSV). This can be controlled by setting the filter bitmap with this function. Setting a certain type's mask bit to 1 will filter out the packet with that type.

Parameters

bitmap	The FrameType FilterMask.

3.12.2 Typedef Documentation

gpHal_EDConfirmCallback_t

```
gpHal_EDConfirmCallback_t
```

The parameter protoED isn't the real energy level. The real energy level needs to be calculated with the function gpHal_CalculateED().

gpHal_SourceIdentifier_t

```
gpHal_SourceIdentifier_t
```

The number of supported source identifiers depends on the device type and is specified as GP_HAL_NUMBER_OF_RX_SRCIDS.

3.12.3 Function Documentation

gpHal_CalculateED()

```
UInt8 gpHal_CalculateED (
            UInt8 protoED )
```

This function calculates the ED value from the protoED returned by the ED scan handler. The lowest value is 0, which is at -75dBm, the highest value is 0xFF, which is at -35dBm.

Parameters

protoED	Value returned by ED scan handler.

gpHal_CheckNoLock()

```
Bool gpHal_CheckNoLock (
            void  )
```

This function reports if a lock loss was detected by the radio.

Returns

Result of the check.

gpHal_ConvertProtoEDToProtoRSSI()

```
UInt8 gpHal_ConvertProtoEDToProtoRSSI (
            UInt8 protoED )
```

This function calculates the protoRSSI from the protoED value returned by the data indication handler.

Parameters

protoRSSI	Value returned by data indication handler.

gpHal_DataRequest()

```
gpHal_Result_t gpHal_DataRequest (
            gpHal_DataReqOptions_t * dataReqOptions,
            gpPad_Handle_t padHandle,
            gpPd_Loh_t pdLoh )
```

Performs a DataRequest(according to the IEEE802.15.4 specification). The DataConfirm function can be registered as a callback using gpHal_RegisterDataConfirmCallback().

Possible results are:

- gpHal_ResultSuccess
- gpHal_ResultBusy (no packet buffer available)
- gpHal_ResultInvalidParameter

Parameters

dataReqOptions	csma, multiChannelOptions, macScenario
pdLoh	The packet descriptor structure that contains length, offset and unique handle of the packet content.

gpHal_EDRequest()

```
gpHal_Result_t gpHal_EDRequest (
               UInt32 time_us,
               UInt16 channelMask )
```

This function triggers an energy detection. The energy value of this function is given in the EDConfirm callback (to be registered with gpHal_RegisterEDConfirmCallback).

To stop an ongoing ED Request, call this function again with time_us 0 and channelMask 0x0000. The scan will be aborted and the confirm will be generated with the results up to that point. Note this stop request will not generate a confirm.

Possible results are :

- gpHal_ResultSuccess
- gpHal_ResultBusy (no packet buffer available)

Parameters

time_us	Time period to scan on each channel given in the channelMask (in us).
channelMask	Mask of channels to scan. LSB bit = channel 11, MSB bit = channel 26.

gpHal_GetAvailableSrcIds()

```
UInt8 gpHal_GetAvailableSrcIds (
               void  )
```

This function gets the number of channels on which different Zigbee/MAC stacks can listen simultaniously

Returns

UInt8 The number of indexes on which a seperate rx channel can be configured.

gpHal_GetBeaconSrcPanChecking()

```
Bool gpHal_GetBeaconSrcPanChecking (
               void  )
```

This function gets the property for filtering beacons based on src pan.

gpHal_GetExtendedAddress()

```
void gpHal_GetExtendedAddress (
              MACAddress_t * pExtendedAddress,
              gpHal_SourceIdentifier_t srcId )
```

This function returns the extended address stored.

Parameters

pExtendedAddress	pointer where the Extended Address is read back to
srcId	The src id of the extended address.

gpHal_GetMacRxMode()

```
void gpHal_GetMacRxMode (
              Bool * enableMultiStandard,
              Bool * enableMultiChannel,
              Bool * enableHighSensitivity )
```

This function gets the rx mode configuration for the MAC part of the radio.

Parameters

enableMultiStandard	Pointer to Bool indicating: concurrent listening on ZigBee and BLE channels (not compatible with the other two options)
enableMultiChannel	Pointer to Bool indicating: listening to multiple ZigBee channels simultaniously (not compatible with the other two options)
enableHighSensitivity	Pointer to Bool indicating: for higher sensitivity ZigBee reception (not compatible with the other two options)

Returns

void

gpHal_GetPanId()

```
UInt16 gpHal_GetPanId (
              gpHal_SourceIdentifier_t srcId )
```

Parameters

srcId	The src id of the Pan.

gpHal_GetRxChannel()

```
UInt8 gpHal_GetRxChannel (
              gpHal_SourceIdentifier_t srcId )
```

Parameters

srcId	The source identifier.

gpHal_GetShortAddress()

```
UInt16 gpHal_GetShortAddress (
                gpHal_SourceIdentifier_t srcId )
```

Parameters

srcId	The src id of the pan where we want to get the short address.

gpHal_RegisterBusyTXCallback()

```
void gpHal_RegisterBusyTXCallback (
                gpHal_BusyTXCallback_t callback )
```

This function registers the BusyTX callback. The callback will be executed on a BusyTX interrupt, i.e. is triggered when the MAC receives a TX trigger while he is already transmitting another packet. The BusyTX interrupt needs to be enabled.

Parameters

callback	The pointer to the callback function.

gpHal_RegisterCmdDataReqConfirmCallback()

```
void gpHal_RegisterCmdDataReqConfirmCallback (
                gpHal_CmdDataReqCallback_t callback )
```

This function registers the CmdDataReq callback. This function determines whether the Frame Pending bit in an Ack needs to be set.

Parameters

callback	The pointer to the callback function.

gpHal_RegisterDataConfirmCallback()

```
void gpHal_RegisterDataConfirmCallback (
                gpHal_DataConfirmCallback_t callback )
```

This function registers the callback for a DataConfirm. The callback will be executed on a DataConfirm interrupt. This DataConfirm will be given after a DataRequest is finished.

The Primitive interrupt needs to be enabled.

Parameters

callback	The pointer to the callback function.

gpHal_RegisterDataIndicationCallback()

```
void gpHal_RegisterDataIndicationCallback (
                gpHal_DataIndicationCallback_t callback )
```

This function registers the DataIndication callback. The callback will be executed on a DataIndication interrupt This DataIndication will be given if a packet is received.

The Primitive interrupt needs to be enabled.

Parameters

callback	The pointer to the callback function.

gpHal_RegisterEDConfirmCallback()

```
void gpHal_RegisterEDConfirmCallback (
                gpHal_EDConfirmCallback_t callback )
```

This function registers the EDConfirm callback. The callback will be executed on a EDConfirm interrupt This EDConfirm will be given after a EDRequest is finished

The Primitive interrupt needs to be enabled.

Parameters

callback	The pointer to the callback function.

gpHal_RegisterEmptyQueueCallback()

```
void gpHal_RegisterEmptyQueueCallback (
                gpHal_EmptyQueueCallback_t callback )
```

This function registers the EmptyQueue callback. The callback will be executed on an Empty Queue interrupt, i.e. is triggered when a TX trigger is given to the MAC when no packet is pending in the TX queue. The Empty Queue interrupt needs to be enabled.

Parameters

callback	The pointer to the callback function.

gpHal_ResetExtendedAddress()

```
void gpHal_ResetExtendedAddress (
```

```
            gpHal_SourceIdentifier_t srcId )
```

This function resets the extended address to its initial factory value.

Parameters

srcId	The src id of the extended address.

gpHal_SetAddressRecognition()

```
void gpHal_SetAddressRecognition (
            Bool enable,
            Bool panCoordinator )
```

This function sets the address recognition options.

Parameters

enable	Possible values are : • set to true : destination address of a packet will checked against the address (set by gpHal_SetExtendedAddress and gpHal_SetShortAddress) and destination PAN ID of incoming packets. • set to false: address recognition disabled.
panCoordinator	Possible values are : • set to true : The device is a PAN coordinator. He will accept messages without a destination address. • set to false: Normal filtering will be applied according to recognition settings.

gpHal_SetAutoAcknowledge()

```
void gpHal_SetAutoAcknowledge (
            Bool flag )
```

This function sets the Auto Acknowledge flag. All packets addressed to the device (see address recognition) will be automatically acknowledge (if requested by the MAC headerof the packet).

Parameters

flag	• Set to true : Automatic acknowledgement enabled. • Set to false: Automatic acknowledgement disabled.

gpHal_SetBeaconSrcPanChecking()

```
void gpHal_SetBeaconSrcPanChecking (
              Bool enable )
```

This function sets the property for filtering beacons based on src pan.

Parameters

enable	If we want to enable beacon filtering on src pan or not.

gpHal_SetCoordExtendedAddress()

```
void gpHal_SetCoordExtendedAddress (
              MACAddress_t * pCoordExtendedAddress )
```

Setting the Coordinator Address enables the filtering of packets not coming from the coordinator.

Parameters

Address	The pointer to the Address of the coordinator.

gpHal_SetDefaultTransmitPowers()

```
void gpHal_SetDefaultTransmitPowers (
              gpHal_TxPower_t * pDefaultTransmitPowerTable )
```

Parameters

pointer	to 16 byte array with default transmit power for each IEEE channel (11..26).

gpHal_SetExtendedAddress()

```
void gpHal_SetExtendedAddress (
              MACAddress_t * pExtendedAddress,
              gpHal_SourceIdentifier_t srcId )
```

This function sets the Extended Address. Setting the Extended Address of your device enables the automatic filter of packets not intended for your device.

Parameters

pExtendedAddress	The pointer to the Extended Address of the device.
srcId	The src id of the extended address.

gpHal_SetMacRxMode()

```
gpHal_Result_t gpHal_SetMacRxMode (
               Bool enableMultiStandard,
               Bool enableMultiChannel,
               Bool enableHighSensitivity )
```

This function sets the rx mode configuration for the MAC part of the radio.

Parameters

enableMultiStandard	Allows concurrent listening on ZigBee and BLE channels (not compatible with the other two options). This option is also known as ConcurrentConnect™. Note that this is not available on some older products.
enableMultiChannel	Allows listening to multiple ZigBee channels simultaneously (not compatible with the other two options)
enableHighSensitivity	Allows for higher sensitivity ZigBee reception (not compatible with the other two options)

Returns

gpHal_Result_t Possible results are :

- gpHal_ResultSuccess
- gpHal_ResultInvalidParameter (invalid combination was selected)

gpHal_SetPanCoordinator()

```
void gpHal_SetPanCoordinator (
               Bool panCoordinator )
```

This function sets the pan coordinator property of the device.

Parameters

panCoordinator	true if the device is the pan coordinator, false otherwise.

gpHal_SetPanId()

```
void gpHal_SetPanId (
               UInt16 panId,
               gpHal_SourceIdentifier_t srcId )
```

This function sets the PAN ID . Setting the PAN ID of your network enables the automatic filter of packets not intended for your device.

Parameters

panId	The PAN ID of the network.

Parameters

srcId	The PAN src, we want to change the ID from.

gpHal_SetPipMode()

```
void gpHal_SetPipMode (
                Bool pipmode )
```

The function sets the Rx packet in packet mode on or off

gpHal_SetPromiscuousMode()

```
void gpHal_SetPromiscuousMode (
                Bool flag )
```

In promiscuous mode, all packets will be received. In order to enable the receiver the RxOnWhenIdle flag must be set.

Parameters

flag	Possible values are : • set to true : Promiscuous mode is enabled and the filters disabled. • set to false: Normal filtering is applied on incoming packets.

gpHal_SetRxOnWhenIdle()

```
void gpHal_SetRxOnWhenIdle (
                gpHal_SourceIdentifier_t srcId,
                Bool flag,
                UInt8 channel )
```

This function sets the RxOnWhenIdle flag. Turns on the receiver when the device is idle. Switching between TX and RX is done automatically.

Parameters

srcId	Source identifier.
flag	Possible values are : • set to true : RxOnWhenIdle is activated and the radio is turned on. • set to false: RxOnWhenIdle is deactivated.
channel	channel to enable radio on

gpHal_SetShortAddress()

```
void gpHal_SetShortAddress (
                UInt16 shortAddress,
                gpHal_SourceIdentifier_t srcId )
```

This functions sets the Short Address. Setting the Short Address of your device enables the automatic filter of packets not intended for your device.

Parameters

shortAddress	The Short Address of the device.
srcId	The src id of the pan where we want to change the short address.

gpHal_SetTimedMode()

```
void gpHal_SetTimedMode (
                Bool timedMode )
```

This function sets the chip to timed MAC mode. When a timed MAC is used all transmission is done using scheduled triggers from the Event Scheduler (ES). This function may only be called once after the initialization of the stack.

3.13 gpHal_MAC_Ext.h File Reference

This file contains all the extra functionality for MAC.

Functions

- gpHal_Result_t gpHal_PurgeRequest (gpPd_Handle_t pdHandle)

  Purges packet from TX queue.

3.13.1 Function Documentation

gpHal_PurgeRequest()

```
gpHal_Result_t gpHal_PurgeRequest (
                gpPd_Handle_t pdHandle )
```

This function purges a packet currently in the TX queue using the pd handle of the packet. When the handle is non-existent or the packet is already transmitted an error is returned.

Possible results are:

- gpHal_ResultSuccess
- gpHal_ResultInvalidRequest (Invalid pd handle given)
- gpHal_ResultBusy (Packet already sent)

Parameters

pdHandle	Unique pd handle given to packet at request time.

3.14 gpHal_MISC.h File Reference

This file contains miscellaneous functions for GPIO and OTP functionality.

Functions

- UInt8 gpHal_GetRandomSeed (void)
  Returns a 8-bit random value.
- void gpHal_GetQRNGRandomSeed (UInt8 size, UInt8 *buffer)
  fills a buffer with entropy from the QRNG.

3.14.1 Function Documentation

gpHal_GetQRNGRandomSeed()

```
void gpHal_GetQRNGRandomSeed (
              UInt8 size,
              UInt8 * buffer )
```

This function fills a buffer with entropy from the QRNG.

gpHal_GetRandomSeed()

```
UInt8 gpHal_GetRandomSeed (
              void  )
```

This function returns a 8-bit random value using samples of the GP chip radio I and Q signals.

3.15 gpHal_MSI.h File Reference

This file declares functions to access the radio over the MSI protocol.

Macros

- #define GP_HAL_READ_REG(Address) readRegExternal(Address)
  Read register access macro.
- #define GP_HAL_WRITE_REG(Address, Data) writeRegExternal(Address, Data)
  Write register access macro.
- #define GP_HAL_READ_BYTE_STREAM(Address, pBuffer, Length) readByteStreamExternal(Address, (UInt8*)pBuffer, Length)
  Read byte stream access macro.
- #define GP_HAL_WRITE_BYTE_STREAM(Address, pBuffer, Length) writeByteStreamExternal(Address, (const UInt8*)pBuffer, Length)

Write byte stream access macro.

- #define GP_HAL_READMODIFYWRITE_REG(Address, Mask, Data) readModifyWriteRegExternal(Address, Mask, Data)

  Read-moify-write access macro.

Typedefs

- typedef UInt32 **gpHal_Address_t**

3.15.1 Macro Definition Documentation

GP_HAL_READ_BYTE_STREAM

```
#define GP_HAL_READ_BYTE_STREAM(
                Address,
                pBuffer,
                Length ) readByteStreamExternal(Address, (UInt8*)pBuffer, Length)
```

This is the raw macro for reading bytes from consecutive registers.

Depending on the processor the macros are used on, access to the registers is direct (embedded processor) or using SPI/I2C (external processor).

Parameters

Address	The address of the first register to access.
pBuffer	The pointer to the buffer that receives the read data.
Length	The number of bytes to read.

GP_HAL_READ_REG

```
#define GP_HAL_READ_REG(
                Address ) readRegExternal(Address)
```

This is the raw macro for reading registers.

Depending on the processor the macros are used on, access to the registers is direct (embedded processor) or using SPI/I2C (external processor).

Parameters

Address	The address of the register.

GP_HAL_READMODIFYWRITE_REG

```
#define GP_HAL_READMODIFYWRITE_REG(
                Address,
                Mask,
                Data ) readModifyWriteRegExternal(Address, Mask, Data)
```

This is the raw macro for changing register bits with a read-modify-write action.

Depending on the processor the macros are used on, access to the registers is direct (embedded processor) or using SPI/I2C (external processor).

Parameters

Register	The address of the register.
Mask	The read-modify-write mask.
Data	The value to write.

GP_HAL_WRITE_BYTE_STREAM

```
#define GP_HAL_WRITE_BYTE_STREAM(
              Address,
              pBuffer,
              Length ) writeByteStreamExternal(Address, (const UInt8*)pBuffer, Length)
```

This is the raw macro for writing bytes to consecutive registers.

Depending on the processor the macros are used on, access to the registers is direct (embedded processor) or using SPI/I2C (external processor).

Parameters

Address	The address of the first register to access.
pBuffer	The pointer to the buffer that contains the data to write.
Length	The number of bytes to write.

GP_HAL_WRITE_REG

```
#define GP_HAL_WRITE_REG(
              Address,
              Data ) writeRegExternal(Address, Data)
```

This is the raw macro for writing registers.

Depending on the processor the macros are used on, access to the registers is direct (embedded processor) or using SPI/I2C (external processor).

Parameters

Address	The address of the register.
Data	The data to write.

3.16 gpHal_OscillatorBenchmark.h File Reference

Enumerations

- enum **gpHal_OscillatorBenchmark_Status_t** { **gpHal_OscillatorBenchmark_Result_NeedMoreSamples** = 0, **gpHal_OscillatorBenchmark_Result_Stable** = 1, **gpHal_OscillatorBenchmark_Result_Unstable** = 2, **gpHal_OscillatorBenchmark_Result_Broken** = 3 }

Functions

- void **gpHal_OscillatorBenchmark_RunAvg8_Init** (void)
- UInt32 **gpHal_OscillatorBenchmark_RunAvg8_Add** (UInt32 benchmark)
- void **gpHal_OscillatorBenchmark_3Phase_Init** (UInt8 stable_length)
- gpHal_OscillatorBenchmark_Status_t **gpHal_OscillatorBenchmark_3Phase_Add** (UInt32 benchmark)
- UInt32 **gpHal_OscillatorBenchmark_MSE_GetStableValue** (void)
- UInt32 **gpHal_OscillatorBenchmark_3Phase_GetAvg** (void)
- UInt32 **gpHal_OscillatorBenchmark_3Phase_GetMSE** (void)

3.17 gpHal_Pbm.h File Reference

This file contains all the functions needed for PBM functionality.

Macros

- #define GPHAL_NUMBER_OF_PBMS_USED GPHAL_MM_PBM_NR_OF

  Number of PBMs.
- #define **GPHAL_PBM_MAX_SIZE** GPHAL_MM_PBM_MAX_SIZE
- #define **GP_PBM_INVALID_HANDLE** 0xFF
- #define GP_HAL_IS_VALID_PBM_FRAME_OFFSET(offset) GP_HAL_PBM_OFFSET_VALID(offset)
- #define GP_HAL_CALCULATE_TX_PBM_DATA_BUFFER_ADDRESS(entry) (GP_HAL_PBM_ENTRY2ADDR GPHAL_REGISTER_PBM_FORMAT_T_FRAME_0)

  Macro for calculation of PBM buffer address.
- #define GP_HAL_WRITE_DATA_IN_PBM(address, pData, length, offset)

  Macro for writing data to a PBM buffer.
- #define GP_HAL_WRITE_BYTE_IN_PBM(address, byte, offset)

  Macro for writing one byte to a PBM buffer.
- #define GP_HAL_READ_DATA_IN_PBM(address, pData, length, offset)

  Macro for reading data from a PBM buffer.
- #define GP_HAL_READ_BYTE_IN_PBM(address, offset) GP_HAL_READ_REG((address)+ (offset))

  Macro for reading one byte from a PBM buffer.

Functions

- UInt8 gpHal_GetHandle (UInt16 size)

  Returns a free handle.
- void gpHal_FreeHandle (UInt8 handle)

  Releases the given handle.

- Int8 gpHal_GetRSSI (UInt8 PBMhandle)

  Query the calibrated RSSI value from the specified pbm entry.
- UInt8 gpHal_GetLQI (UInt8 PBMhandle)

  Query the calibrated LQI value from the specified pbm entry.
- void gpHal_GetRxTimestamp (UInt8 PBMentry, UInt32 *pTimeStamp)

  Query the timestamp of a received packet from the specified pbm entry.
- UInt16 * gpHal_GetPhaseSamples (UInt8 PBMhandle)

  Get the DF (AOA/AOD) Samples buffer associated with this PBM.
- UInt8 gpHal_GetRxedChannel (UInt8 PBMentry)

  Get the Rx Channel on which the packet was received.
- Int32 gpHal_GetRxedFreqOffset (UInt8 PBMentry)

  Get the frequency offset (in Hz) with which the packet was received - used for the DF (AOA/AOD)feature.
- UInt16 gpHal_GetRxedAntenna (UInt8 PBMentry)

  Get the (internal) antenna with which the packet was received - used for the DF (AOA/AOD)feature.
- void gpHal_GetTxTimestamp (UInt8 PBMentry, UInt32 *pTimeStamp)

  Query the timestamp of a transmitted packet from the specified pbm entry.
- UInt8 gpHal_GetTxAckLQI (UInt8 PBMentry)

  Query the LQI value of the ACK packet related to the specified TX pbm entry.
- UInt8 gpHal_GetTxCCACntr (UInt8 PBMentry)

  Query the CCA counter of a transmitted packet from the specified pbm entry.
- UInt8 gpHal_GetTxRetryCntr (UInt8 PBMhandle)

  Query the retry counter of a transmitted packet from the specified pbm entry.
- UInt8 gpHal_GetFramePendingFromTxPbm (UInt8 PBMentry)

  Query the framepending bit from the ACK of a transmitted packet from the specified pbm entry.
- Int8 gpHal_CalculateRSSI (UInt8 protoRSSI)

  Calculate the RSSI from the protoRSSI returned by the data indication handler.
- UInt8 gpHal_CalculateProtoRSSI (Int8 protoRSSI)

  Calculate the proto RSSI from the RSSI.
- UInt8 gpHal_CalculateLQIfromRSSI (Int8 rssi)

  Calculate the LQI of a received packet based on RSSI.
- void gpHal_WriteDataInPBMCyclic (gpHal_Address_t pbmAddr, UInt8 pbmOffset, UInt8 *pData, UInt8 length)

  function for writing data to a PBM buffer on a cyclic way.
- void gpHal_ReadDataInPBMCyclic (gpHal_Address_t pbmAddr, UInt8 pbmOffset, UInt8 *pData, UInt8 length)

  Macro for reading data from a PBM buffer. The destination is the data segment of the PBM. This segment is handled as a cyclic buffer.
- void gpHal_WriteByteInPBMCyclic (gpHal_Address_t pbmAddr, UInt8 pbmOffset, UInt8 byte)

  Functino for writing one byte to a PBM buffer. The destination is the data segment of the PBM. This segment is handled as a cyclic buffer.
- UInt8 gpHal_ReadByteInPBMCyclic (gpHal_Address_t pbmAddr, UInt8 pbmOffset)

  Function for reading one byte from a PBM buffer. The destination is the data segment of the PBM. This segment is handled as a cyclic buffer.
- void gpHal_WriteDataInPBM (gpHal_Address_t pbmAddr, UInt16 pbmOffset, UInt8 *pData, UInt8 length)

  function for writing data to a PBM buffer.
- void gpHal_ReadDataInPBM (gpHal_Address_t pbmAddr, UInt16 pbmOffset, UInt8 *pData, UInt8 length)

Macro for reading data from a PBM buffer. The destination is the data segment of the PBM.

- void gpHal_WriteByteInPBM (gpHal_Address_t pbmAddr, UInt16 pbmOffset, UInt8 byte)

  Functino for writing one byte to a PBM buffer. The destination is the data segment of the PBM.
- UInt8 gpHal_ReadByteInPBM (gpHal_Address_t pbmAddr, UInt16 pbmOffset)

  Function for reading one byte from a PBM buffer. The destination is the data segment of the PBM.
- void gpHal_MakeBareCopyPBM (UInt8 PBMentryOrig, UInt8 PBMentryDst)

  Function for duplicating a PBM in a new PBM.

3.17.1 Macro Definition Documentation

GP_HAL_CALCULATE_TX_PBM_DATA_BUFFER_ADDRESS

```
#define GP_HAL_CALCULATE_TX_PBM_DATA_BUFFER_ADDRESS(
                entry ) (GP_HAL_PBM_ENTRY2ADDR(entry)+ GPHAL_REGISTER_PBM_FORMAT_T_FRAME_0)
```

range check for offsets into a pbm frameThis macro calculates the base address of the PBM buffer defined by the parameter address. This macro should be called before accessing any byte of the PBM buffer.

Parameters

PBMentry	The index of the PBM buffer to tranmsit

Returns

The base address of the PBM buffer.

GP_HAL_IS_VALID_PBM_FRAME_OFFSET

```
#define GP_HAL_IS_VALID_PBM_FRAME_OFFSET(
                offset ) GP_HAL_PBM_OFFSET_VALID(offset)
```

range check for offsets into a pbm frame

GP_HAL_READ_BYTE_IN_PBM

```
#define GP_HAL_READ_BYTE_IN_PBM(
                address,
                offset ) GP_HAL_READ_REG((address)+ (offset))
```

This macro reads data from a PBM buffer defined by the parameter address. This macro should be used in order to read back one byte of the payload of a PBM buffer.

Parameters

address	The base address of the PBM where the byte should be read from. This value should be obtained from an interrupt callback function
offset	The offset from the base address of the PBM where the byte is read from.

Returns

The read value.

GP_HAL_READ_DATA_IN_PBM

```
#define GP_HAL_READ_DATA_IN_PBM(
                address,
                pData,
                length,
                offset )
```

Value:

```
do {            \
        UInt16 offset_1 = (offset); /*avoid multiple evaluation of argument*/  \
        GP_ASSERT_DEV_EXT(GP_HAL_IS_VALID_PBM_FRAME_OFFSET(offset_1) );
                \
        GP_ASSERT_DEV_EXT(GP_HAL_IS_VALID_PBM_FRAME_OFFSET(offset_1 + (
      length) - 1) ); \
        GP_HAL_READ_BYTE_STREAM(( (address) + offset_1 ) , (pData) , (length) ); \
    } while (false)
```

This macro reads data from a PBM buffer defined by the parameter address. This macro should be used in order to read back the payload of a PBM buffer.

Parameters

address	The base address of the PBM where the data should be read from. This value should be otained from an interrupt callback function
pData	The pointer where the read data will be written to.
length	The number of bytes that will be read.
offset	The offset from the base address of the PBM where the data is read from.

GP_HAL_WRITE_BYTE_IN_PBM

```
#define GP_HAL_WRITE_BYTE_IN_PBM(
                address,
                byte,
                offset )
```

Value:

```
do {                \
        UInt16 offset_1 = (offset); /*avoid multiple evaluation of argument*/  \
        GP_ASSERT_DEV_EXT(GP_HAL_IS_VALID_PBM_FRAME_OFFSET(offset_1));
                \
        GP_HAL_WRITE_REG((address) + offset_1, byte); \
    } while (false)
```

This macro writes one byte to a PBM buffer defined by the parameter address. This macro should be used in order to update the payload of a PBM buffer.

Parameters

address	The base address of the PBM where the data should be written to. This value should be calculated with the macro GP_HAL_CALCULATE_TX_PBM_DATA_BUFFER_ADDRESS().
byte	The value that will be written.
offset	The offset from the base address of the PBM where the byte should be written to.

GP_HAL_WRITE_DATA_IN_PBM

```
#define GP_HAL_WRITE_DATA_IN_PBM(
                address,
                pData,
                length,
                offset )
```

Value:

```
do {      \
        UInt16 offset_l = (offset); /*avoid multiple evaluation of argument*/  \
        GP_ASSERT_DEV_EXT(GP_HAL_IS_VALID_PBM_FRAME_OFFSET(offset_l) );
                  \
        GP_ASSERT_DEV_EXT(GP_HAL_IS_VALID_PBM_FRAME_OFFSET(offset_l + (
      length) - 1) ); \
        GP_HAL_WRITE_BYTE_STREAM(( (address) + offset_l ), (pData), (length)); \
    } while (false)
```

This macro writes data to a PBM buffer defined by the parameter address. This macro should be used in order to update the payload of a PBM buffer.

Parameters

address	The base address of the PBM where the data should be written to. This value should be calculated with the macro GP_HAL_CALCULATE_TX_PBM_DATA_BUFFER_ADDRESS().
pData	The pointer to the data that will be written.
length	The number of bytes that will be written.
offset	The offset from the base address of the PBM where the data should be written to.

3.17.2 Function Documentation

gpHal_CalculateLQIfromRSSI()

```
UInt8 gpHal_CalculateLQIfromRSSI (
              Int8 rssi )
```

This function calculates the LQI of a received packet based on the RSSI. The lowest value is 0 which is at -93dBm. The highest value is 0xFF witch is at -20dBm.

Parameters

RSSI	Value returned by data indication handler.

gpHal_CalculateProtoRSSI()

```
UInt8 gpHal_CalculateProtoRSSI (
            Int8 protoRSSI )
```

This function calculates the proto RSSI from the RSSI value.

Parameters

protoRSSI	Value returned by data indication handler.

gpHal_CalculateRSSI()

```
Int8 gpHal_CalculateRSSI (
            UInt8 protoRSSI )
```

This function calculates the RSSI from the protoRSSI value returned by the data indication handler.

Parameters

protoRSSI	Value returned by data indication handler.

gpHal_FreeHandle()

```
void gpHal_FreeHandle (
            UInt8 handle )
```

This function releases the PBM buffer associated with the handle.

Parameters

handle	A valid handle which will be released.

gpHal_GetFramePendingFromTxPbm()

```
UInt8 gpHal_GetFramePendingFromTxPbm (
            UInt8 PBMentry )
```

This function returns the framepending bit from the ACK of a transmitted packet for a specified pbm entry.

Parameters

PBMentry	the pbm entry containing the requested data.

gpHal_GetHandle()

```
UInt8 gpHal_GetHandle (
              UInt16 size )
```

This function allocate a free PBM buffer and returns its handle. Returns 0xFF if no free buffer is available;

gpHal_GetLQI()

```
UInt8 gpHal_GetLQI (
              UInt8 PBMhandle )
```

This function returns the calibrated LQI from the specified pbm entry. LQI is a value from 0 to 0xFF with 0 the lowest value and 0xFF the highest value. LQI is only based on signal strength not on correlation.

Parameters

PBMentry	the pbm entry containing the requested data.

gpHal_GetPhaseSamples()

```
UInt16* gpHal_GetPhaseSamples (
              UInt8 PBMhandle )
```

This function returns the DF samples from the specified pbm entry.

Parameters

PBMentry	the pbm entry containing the requested data.

gpHal_GetRSSI()

```
Int8 gpHal_GetRSSI (
              UInt8 PBMhandle )
```

This function returns the calibrated RSSI from the specified pbm entry.

Parameters

PBMentry	the pbm entry containing the requested data.

gpHal_GetRxedAntenna()

```
UInt16 gpHal_GetRxedAntenna (
            UInt8 PBMentry )
```

This function returns the Rx antenna (internal antenna id) from the specified pbm entry.

Parameters

PBMentry	the pbm entry containing the requested data.

gpHal_GetRxedChannel()

```
UInt8 gpHal_GetRxedChannel (
            UInt8 PBMentry )
```

This function returns the Rx Channel from the specified pbm entry.

Parameters

PBMentry	the pbm entry containing the requested data.

Returns

rxChannel Channel on which the packet was received.

gpHal_GetRxedFreqOffset()

```
Int32 gpHal_GetRxedFreqOffset (
            UInt8 PBMentry )
```

This function returns the frequency offset from the specified pbm entry.

Parameters

PBMentry	the pbm entry containing the requested data.

gpHal_GetRxTimestamp()

```
void gpHal_GetRxTimestamp (
            UInt8 PBMentry,
            UInt32 * pTimeStamp )
```

This function returns the timestamp of a received packet for a specified pbm entry. The timestamp is taken at the beginning of the frame (preamble).

Parameters

PBMentry	the pbm entry containing the requested data.
timeStamp	the pointer to which the timestamp will be returned

gpHal_GetTxAckLQI()

```
UInt8 gpHal_GetTxAckLQI (
              UInt8 PBMentry )
```

This function returns the LQI of the received ACK packet, the passed pbm entry needs to be for a TX packet.

Parameters

PBMentry	the TX pbm entry containing the requested data.

Note

if transmission is not in ACKED MODE or the ACK is not received, 0 is returned

gpHal_GetTxCCACntr()

```
UInt8 gpHal_GetTxCCACntr (
              UInt8 PBMentry )
```

This function returns the number of CCA backoffs that were done for transmitting the packet. If csma-cca is disabled, 0 is returned.

Parameters

PBMentry	the pbm entry containing the requested data.

gpHal_GetTxRetryCntr()

```
UInt8 gpHal_GetTxRetryCntr (
              UInt8 PBMhandle )
```

This function returns the retry counter of a transmitted packet for a specified pbm entry.

Parameters

PBMentry	the pbm entry containing the requested data.

Note

retry counter 0 means - first time succesfully transmitted

gpHal_GetTxTimestamp()

```
void gpHal_GetTxTimestamp (
              UInt8 PBMentry,
              UInt32 * pTimeStamp )
```

This function returns the timestamp of a transmitted packet for a specified pbm entry. The timestamp is taken at the beginning of the frame (preamble).

Parameters

PBMentry	the pbm entry containing the requested data.
timeStamp	the pointer to which the timestamp will be returned

gpHal_MakeBareCopyPBM()

```
void gpHal_MakeBareCopyPBM (
              UInt8 PBMentryOrig,
              UInt8 PBMentryDst )
```

Parameters

PBMentryOrig	The pbm.which will be copied
PBMentryDst	The destination pbm. This pbm will have the same data and options as the original pbm.

gpHal_ReadByteInPBM()

```
UInt8 gpHal_ReadByteInPBM (
              gpHal_Address_t pbmAddr,
              UInt16 pbmOffset )
```

This function reads data from a PBM buffer defined by the parameter address. This function should be used in order to read back one byte of the payload of a PBM buffer.

Parameters

address	The base address of the PBM where the byte should be read from. This value should be calculated with the macro GP_HAL_CALCULATE_TX_PBM_DATA_BUFFER_ADDRESS().
offset	The offset from the base address of the PBM where the byte is read from.

Returns

The read value.

gpHal_ReadByteInPBMCyclic()

```
UInt8 gpHal_ReadByteInPBMCyclic (
              gpHal_Address_t pbmAddr,
              UInt8 pbmOffset )
```

This function reads data from a PBM buffer defined by the parameter address. This function should be used in order to read back one byte of the payload of a PBM buffer.

Parameters

address	The base address of the PBM where the byte should be read from. This value should be calculated with the macro GP_HAL_CALCULATE_TX_PBM_DATA_BUFFER_ADDRESS().
offset	The offset from the base address of the PBM where the byte is read from.

Returns

The read value.

gpHal_ReadDataInPBM()

```
void gpHal_ReadDataInPBM (
              gpHal_Address_t pbmAddr,
              UInt16 pbmOffset,
              UInt8 * pData,
              UInt8 length )
```

This function reads data from a PBM buffer defined by the parameter address. This function should be used in order to read back the payload of a PBM buffer.

Parameters

address	The base address of the PBM where the data should be read from. This value should be calculated with the macro GP_HAL_CALCULATE_TX_PBM_DATA_BUFFER_ADDRESS().
pData	The pointer where the read data will be written to.
length	The number of bytes that will be read.
offset	The offset from the base address of the PBM where the data is read from.

gpHal_ReadDataInPBMCyclic()

```
void gpHal_ReadDataInPBMCyclic (
              gpHal_Address_t pbmAddr,
              UInt8 pbmOffset,
              UInt8 * pData,
              UInt8 length )
```

This function reads data from a PBM buffer defined by the parameter address. This function should be used in order to read back the payload of a PBM buffer.

Parameters

address	The base address of the PBM where the data should be read from. This value should be calculated with the macro GP_HAL_CALCULATE_TX_PBM_DATA_BUFFER_ADDRESS().
pData	The pointer where the read data will be written to.
length	The number of bytes that will be read.

Parameters

offset	The offset from the base address of the PBM where the data is read from.

gpHal_WriteByteInPBM()

```
void gpHal_WriteByteInPBM (
              gpHal_Address_t pbmAddr,
              UInt16 pbmOffset,
              UInt8 byte )
```

This function writes one byte to a PBM buffer defined by the parameter address. This function should be used in order to update the payload of a PBM buffer.

Parameters

address	The base address of the PBM where the data should be written to. This value should be calculated with the macro GP_HAL_CALCULATE_TX_PBM_DATA_BUFFER_ADDRESS().
byte	The value that will be written.
offset	The offset from the base address of the PBM where the byte should be written to.

gpHal_WriteByteInPBMCyclic()

```
void gpHal_WriteByteInPBMCyclic (
              gpHal_Address_t pbmAddr,
              UInt8 pbmOffset,
              UInt8 byte )
```

This function writes one byte to a PBM buffer defined by the parameter address. This function should be used in order to update the payload of a PBM buffer.

Parameters

address	The base address of the PBM where the data should be written to. This value should be calculated with the macro GP_HAL_CALCULATE_TX_PBM_DATA_BUFFER_ADDRESS().
byte	The value that will be written.
offset	The offset from the base address of the PBM where the byte should be written to.

gpHal_WriteDataInPBM()

```
void gpHal_WriteDataInPBM (
              gpHal_Address_t pbmAddr,
              UInt16 pbmOffset,
              UInt8 * pData,
              UInt8 length )
```

This function writes data to a PBM buffer defined by the parameter address. The destination is the data segment of the PBM. This function should be used in order to update the payload of a PBM buffer.

Parameters

address	The base address of the PBM where the data should be written to. This value should be calculated with the macro GP_HAL_CALCULATE_TX_PBM_DATA_BUFFER_ADDRESS().
pData	The pointer to the data that will be written.
length	The number of bytes that will be written.
offset	The offset from the base address of the PBM where the data should be written to.

gpHal_WriteDataInPBMCyclic()

```
void gpHal_WriteDataInPBMCyclic (
              gpHal_Address_t pbmAddr,
              UInt8 pbmOffset,
              UInt8 * pData,
              UInt8 length )
```

This function writes data to a PBM buffer defined by the parameter address. The destination is the data segment of the PBM. This segment is handled as a cyclic buffer. This function should be used in order to update the payload of a PBM buffer.

Parameters

address	The base address of the PBM where the data should be written to. This value should be calculated with the macro GP_HAL_CALCULATE_TX_PBM_DATA_BUFFER_ADDRESS().
pData	The pointer to the data that will be written.
length	The number of bytes that will be written.
offset	The offset from the base address of the PBM where the data should be written to.

3.18 gpHal_reg.h File Reference

3.18.1 Detailed Description

Wrapper around the register definitions.

3.19 gpHal_SEC.h File Reference

Contains all security functionality of the HAL.

Functions

- gpHal_Result_t gpHal_AESEncrypt (UInt8 *pInplaceBuffer, UInt8 *pAESKey, gpEncryption_AESOptions_t AESOptions)

  Performs a synchronous AES Encryption.
- gpHal_Result_t gpHal_CCMEncrypt_RAM (UInt16 dataLength, UInt16 auxLength, UInt8 micLength, UInt8 *dataPtr, UInt8 *auxPtr, UInt8 *micPtr, UInt8 *pKey, UInt8 *pNonce, UInt8 *dataOutPtr)

  Performs a synchronous CCM Encryption with support for larger buffers.
- gpHal_Result_t gpHal_CCMDecrypt_RAM (UInt16 dataLength, UInt16 auxLength, UInt8 micLength, UInt8 *dataPtr, UInt8 *auxPtr, UInt8 *micPtr, UInt8 *pKey, UInt8 *pNonce, UInt8 *dataOutPtr)

  Performs a synchronous CCM Decryption with support for larger buffers.
- gpHal_Result_t gpHal_CCMEncrypt (gpEncryption_CCMOptions_t *pCCMOptions)

  Performs a synchronous CCM Encryption.
- gpHal_Result_t gpHal_CCMDecrypt (gpEncryption_CCMOptions_t *pCCMOptions)

  Performs a synchronous CCM Decryption.
- gpHal_Result_t gpHal_HMAC (UInt8 hashFct, UInt16 keyLength, UInt16 msgLength, UInt8 resultLength, UInt8 *pKey, UInt8 *pMsg, UInt8 *pResult)

  Hash-based message authentication code.
- void gpHalSec_SspAesMMO (UInt32 compressedDataPtr, UInt32 compressedKeyPtr, gpEncryption_AESKeyLen_t keylen, UInt8 msgLengthBytes)

  Performs an AES MMO hash generation.

3.19.1 Detailed Description

This file contains all security functionality of the HAL. Standalone AES encryption can be performed as well as CCM encryption and decryption.

3.19.2 Function Documentation

gpHal_AESEncrypt()

```
gpHal_Result_t gpHal_AESEncrypt (
              UInt8 * pInplaceBuffer,
              UInt8 * pAESKey,
              gpEncryption_AESOptions_t AESOptions )
```

The function will encrypt the number of bytes specified in keylen with the AES algorithm and return the result in place.

Possible results are:

- gpHal_ResultSuccess
- gpHal_ResultBusy
- gpHal_ResultInvalidParameter

Parameters

pInplaceBuffer	Pointer to the buffer of the 16 to be encrypted bytes. Encrypted result will be returned in same buffer
pAESKey	Pointer to the byte key, this key is only uses if specified by the options parameter. When used but specified as NULL, 0 will be used as key value.
AESOptions	This parameter specifies the keylen and an 8bit bitmask specifying the options: bits[6:0] specify the keyid to be used (see gpEncryption_API_Manual); bit[7] indicates additional hardening

gpHal_CCMDecrypt()

```
gpHal_Result_t gpHal_CCMDecrypt (
               gpEncryption_CCMOptions_t * pCCMOptions )
```

The function will decrypt the bytes with the CCM algorithm according to the specified options in the gpHal_CCMOptions structure.

Possible results are:

- gpHal_ResultSuccess
- gpHal_ResultBusy
- gpHal_ResultInvalidParameter

Parameters

pCCMOptions	Pointer to the gpHal_CCMOptions structure.

gpHal_CCMDecrypt_RAM()

```
gpHal_Result_t gpHal_CCMDecrypt_RAM (
               UInt16 dataLength,
               UInt16 auxLength,
               UInt8 micLength,
               UInt8 * dataPtr,
               UInt8 * auxPtr,
               UInt8 * micPtr,
               UInt8 * pKey,
               UInt8 * pNonce,
               UInt8 * dataOutPtr )
```

The function will decrypt the bytes with the CCM algorithm according to the specified options Possible results are:

- gpHal_ResultSuccess
- gpHal_ResultInvalidRequest

Parameters

pCCMOptions	Pointer to the gpHal_CCMOptions structure.

gpHal_CCMEncrypt()

```
gpHal_Result_t gpHal_CCMEncrypt (
               gpEncryption_CCMOptions_t * pCCMOptions )
```

The function will encrypt the bytes with the CCM algorithm according to the specified options in the gpHal_CCMOptions structure.

Possible results are:

- gpHal_ResultSuccess
- gpHal_ResultBusy

Parameters

pCCMOptions	Pointer to the gpHal_CCMOptions structure.

gpHal_CCMEncrypt_RAM()

```
gpHal_Result_t gpHal_CCMEncrypt_RAM (
               UInt16 dataLength,
               UInt16 auxLength,
               UInt8 micLength,
               UInt8 * dataPtr,
               UInt8 * auxPtr,
               UInt8 * micPtr,
               UInt8 * pKey,
               UInt8 * pNonce,
               UInt8 * dataOutPtr )
```

The function will encrypt the bytes with the CCM algorithm according to the specified options

Possible results are:

- gpHal_ResultSuccess
- gpHal_ResultInvalidRequest

Parameters

pCCMOptions	Pointer to the gpHal_CCMOptions structure.

gpHal_HMAC()

```
gpHal_Result_t gpHal_HMAC (
```

```
                UInt8 hashFct,
                UInt16 keyLength,
                UInt16 msgLength,
                UInt8 resultLength,
                UInt8 * pKey,
                UInt8 * pMsg,
                UInt8 * pResult )
```

This function performs HMAC, a mechanism for message authentication using cryptographic hash functions.

Possible results are:

- gpHal_ResultSuccess
- gpHal_ResultBusy
- gpHal_ResultUnsupported

Parameters

hashFct	Type of hash funtion.
pKey	Secret cryptographic Key.
pMsg	Message for authentication.
pResult	Message Authentication Codes Result, The size of the output of HMAC is the same as that of the underlying hash function.

gpHalSec_SspAesMMO()

```
void gpHalSec_SspAesMMO (
                UInt32 compressedDataPtr,
                UInt32 compressedKeyPtr,
                gpEncryption_AESKeyLen_t keylen,
                UInt8 msgLengthBytes )
```

The function will generate a fixed length digest value for a given input message

Parameters

compressedDataPtr	Pointer to the message buffer in compressed address map for which digest has to be generated
compressedKeyPtr	Pointer to the key buffer in compressed address map (generated digest value is stored in this buffer)
keylen	Key length
msgLengthBytes	Input message length in bytes

3.20 gpHal_Statistics.h File Reference

Getters and Setters of gpHal.

Data Structures

- struct gpHal_StatisticsCntPrio_t
- struct gpHal_StatisticsCoexCounter_t
- struct gpHal_StatisticsMacCounter_t

Functions

- void **gpHal_StatisticsCountersClear** (void)
- void **gpHal_StatisticsCountersGet** (gpHal_StatisticsMacCounter_t *pStatisticsMacCounters, gpHal_StatisticsCoexCounter_t *pStatisticsCoexCounters)

3.21 gpHal_Zgp.h File Reference

This file contains all the functions needed for ZigBee GreenPower functionality.

Functions

- void gpHal_GetZgpSourceId (UInt32 *pSourceId, UInt8 *pNumberOfSourceId)

  Returns the Zigbee Greenpower source ID stored in the chip.

3.21.1 Function Documentation

gpHal_GetZgpSourceId()

```
void gpHal_GetZgpSourceId (
              UInt32 * pSourceId,
              UInt8 * pNumberOfSourceId )
```

This function returns the ZigBee GreenPower source Id information stored in chip read-only memory. If no pre-configured sourceId are available pNumberOfSourceId will return zero.

Parameters

pSourceId	Pointer where the Source Id is read back to. In case of multiple sourceIds, this is the start id.
pNumberOfSourceId	Pointer where the amount of sequentially allocated Source Id is returned.