Sensor Hub Framework

1.32

Generated by Doxygen 1.8.8

Tue Aug 4 2015 17:25:35

Contents

Chapter 1

Module Index

1.1 Modules

Here is a list of all modules:

Chapter 2

Data Structure Index

2.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 3

Module Documentation

3.1 Sensor Hub: Breathing LED Driver

Data Structures

- struct LPCSH_BLED_PWM_Slope_t
- struct LPCSH_BLED_LED_t

  Breathing LED Structure.

Functions

- void BLED_Enable (uint8_t channelMask, uint8_t brightness, uint32_t t0, uint32_t t1, uint32_t t2, uint32_t t3)

  Enable breathing LED.
- void BLED_Disable (void)

  Disable breathing LED.
- uint32_t BLED_SleepOk (void)

  Indicates whether breathing LED is active or not.
- uint32_t BLED_UpdateClock (void)

  Updates the timer's prescaler according to the timer's base clock frequency.

3.1.1 Detailed Description

3.1.2 Function Documentation

3.1.2.1 void BLED_Disable (void)

Disable breathing LED.

Returns

none

3.1.2.2 void BLED_Enable (uint8_t *channelMask,* uint8_t *brightness,* uint32_t *t0,* uint32_t *t1,* uint32_t *t2,* uint32_t *t3*)

Enable breathing LED.

Parameters

channelMask	1-7, select on which PWM channels to output the specified PWM profile
brightness	0-255 maximum brightness of LED
t0	Intensity ramp-up time in ms
t1	ON time in ms
t2	Intensity ramp-down time in ms
t3	OFF time in ms

Returns

none

3.1.2.3 uint32_t BLED_SleepOk (void)

Indicates whether breathing LED is active or not.

Returns

true when BLED is enabled, false when disabled

3.1.2.4 uint32_t BLED_UpdateClock (void)

Updates the timer's prescaler according to the timer's base clock frequency.

Returns

0 if OK, 1 if BLED_TIMER_TARGET_CLK_FREQ is too high (i.e. timer's base clock is lower than BLED_TI↩ MER_TARGET_CLK_FREQ)

3.2 Sensor Hub: Host interface

Functions

- void Hostif_Init (void)

  Initialize host interface subsystem.
- void Hostif_StartTx (uint8_t *pBuf, uint16_t size, uint8_t cmd, uint8_t lastPacket)

  Start Host interface transfer.
- uint32_t Hostif_SleepOk (void)

  Tells caller if it is ok to sleep.

Variables

- unsigned char **version_major**
- unsigned char **version_minor**

3.2.1 Detailed Description

3.2.2 Function Documentation

3.2.2.1 void Hostif_Init (void)

Initialize host interface subsystem.

Returns

none

Initialize the Sensor Hub host/AP interface

3.2.2.2 uint32_t Hostif_SleepOk (void)

Tells caller if it is ok to sleep.

Returns

0 - Ok to Sleep. 1 - Not ok to sleep (Hostif Tx might be pending);

3.2.2.3 void Hostif_StartTx (uint8_t * *pBuf*, uint16_t *size*, uint8_t *cmd*, uint8_t *lastPacket*)

Start Host interface transfer.

Parameters

pBuf	: Pointer to buffer to be transmitted
size	: Size of the buffer
cmd	: Last command received.
lastPacket	: Last packet for this batch transfer

Returns

none

Note

The "cmd" param specifies the command Id for which the this response is sent.

3.3 Sensor Hub: Host Command Handler

Functions

- uint64_t Host_Cmd_getWakeupConfiguration (void)

  Returns a sensor mask which indicates the sensors that can wakeup the host.
- uint8_t Host_Cmd_queueBuffer (const uint8_t *pBuffer, uint16_t length, sensorFIFO_t *f, uint32_t timestamp)

  Add the given buffer to transmit queue.
- void Host_Cmd_cmdEngineInit (void)

  Initialize the buffer and command engine.
- void Host_Cmd_cmdProcess (uint8_t *rx_buf, uint16_t length)

  Process the host command.
- void Host_Cmd_checkForPendingTxData (void)

  Callback for Transmit data completion.
- void Host_Cmd_flush_onchange_samples (void)

  Check whether there is any "dirty" On change data which needs to be added to FIFO that we're sending. If so, copy it.
- uint32_t Host_Cmd_processSensorUpdate (void)

  Check if there is any pending sensor timing updates. If so, call Algorithm to process the update.
- void Host_Cmd_setSensorUpdate (uint64_t update_mask)

  Set Sensor timing update flags for processing.
- void Host_Cmd_flush_flush_records (void)

  Check whether there is any "dirty" FLUSH data which needs to be added to FIFO that we're sending. If so, copy it.

3.3.1 Detailed Description

3.3.2 Function Documentation

3.3.2.1 void Host_Cmd_checkForPendingTxData (void)

Callback for Transmit data completion.

Returns

none

Note

Callback routine called by the low-level hostIF driver when it completes the transfer of given data buffer.

3.3.2.2 void Host_Cmd_cmdEngineInit (void)

Initialize the buffer and command engine.

Returns

none

3.3.2.3 void Host_Cmd_cmdProcess (uint8_t * *rx_buf*, uint16_t *length*)

Process the host command.

Parameters

rx_buf	: Pointer to buffer received from host
length	: Length of the buffer

Returns

None.

3.3.2.4 void Host_Cmd_flush_flush_records (void)

Check whether there is any "dirty" FLUSH data which needs to be added to FIFO that we're sending. If so, copy it.

Returns

none

3.3.2.5 void Host_Cmd_flush_onchange_samples (void)

Check whether there is any "dirty" On change data which needs to be added to FIFO that we're sending. If so, copy it.

Returns

none

3.3.2.6 uint64_t Host_Cmd_getWakeupConfiguration (void)

Returns a sensor mask which indicates the sensors that can wakeup the host.

Returns

WakeUp Mask, each bit corresponds to a sensor and if it is 1 then the sensor wakes up the host

3.3.2.7 uint32_t Host_Cmd_processSensorUpdate (void)

Check if there is any pending sensor timing updates. If so, call Algorithm to process the update.

Returns

Ok to Sleep indicator 0 - Ok to Sleep, != 0 Not ok to sleep

3.3.2.8 uint8_t Host_Cmd_queueBuffer (const uint8_t * *pBuffer,* uint16_t *length,* sensorFIFO_t * *f,* uint32_t *timestamp*)

Add the given buffer to transmit queue.

Parameters

pBuffer	: Pointer to buffer.

length	: Size of the buffer.
f	: Pointer to the FIFO buffer
timestamp	: 32 bit timestamp of current sample

Returns

Returns error code. 0 - successful; non-zero - Tx queue is full

3.3.2.9 void Host_Cmd_setSensorUpdate (uint64_t *update_mask*)

Set Sensor timing update flags for processing.

Parameters

update_mask	: Bit field mask of sensors whose timing needs to be update

Returns

None

3.4 Sensor Hub: Host Fifo interface

Data Structures

- struct sensorFIFO_s

Typedefs

- typedef struct sensorFIFO_s sensorFIFO_t

Enumerations

- enum LPCSH_FIFO_IDS_T {
  HOSTIF_CIRC_FIFO_CNT_FIRST = 0, WAKEUP = 0, NON_WAKEUP = 1, LOGGING = 2,
  HOSTIF_CIRC_FIFO_CNT_MAX }

Functions

- uint16_t **Host_Fifo_getAvailableSpace** (sensorFIFO_t *fifo)
- void **Host_Fifo_writeRecordRingBuf** (const uint8_t *pBuffer, uint16_t length, sensorFIFO_t *fifo)
- bool **Host_Fifo_readRecordRingBuf** (const uint8_t *pBuffer, uint16_t length, sensorFIFO_t *fifo)
- bool **Host_Fifo_peekRecordRingBuf** (const uint8_t *pBuffer, uint16_t length, sensorFIFO_t *fifo)
- uint8_t **Host_Fifo_writeRingBuf** (const uint8_t *pBuffer, uint16_t length, sensorFIFO_t *fifo, uint64_t timestamp)
- void **Host_Fifo_determineBatchFifo** (void)
- void **DEBUG_verify_fifo_integrity** (void)
- void **hostif_fifo_init** (void)

Variables

- sensorFIFO_t **sensFifo** [HOSTIF_CIRC_FIFO_CNT_MAX]
- sensorFIFO_t * **pendingFIFO**
- sensorFIFO_t * **lastPendingFIFO**

3.4.1 Detailed Description

3.4.2 Typedef Documentation

3.4.2.1 typedef struct sensorFIFO_s sensorFIFO_t

Defines the generic FIFO data structure for all circular FIFO in framework. the specific handling of FIFO data is customized via function pointers.

3.4.3 Enumeration Type Documentation

3.4.3.1 enum LPCSH_FIFO_IDS_T

Defines the enumeration of FIFO instance The priority order of FIFO is from FIRST (high) to last (low)

Enumerator

WAKEUP WAKEUP FIFO id

NON_WAKEUP NON_WAKEUP FIFO id

LOGGING LOGGING FIFO id

HOSTIF_CIRC_FIFO_CNT_MAX num of FIFO id

3.5 Sensor Hub: Algorithm interface

Functions

- void Algorithm_GetPhysicalSensorList (struct lpcsh_sensor_info_t list[], uint8_t *size)

  returns the list of physical sensors that were deected
- void Algorithm_GetSensorWakeupSettings (struct lpcsh_configuration_data_t *pConfiguration)

  returns sensor hub wakeup sensor configuration bit map per enum LPCSH_SENSOR_ID
- void Algorithm_Init (void)

  Initialize algorithm module.
- uint32_t Algorithm_SensorTimingUpdated (uint64_t sensorEnableMask)

  Update the new sensor timings to algo and physical sensors.
- uint32_t Algorithm_EnableSensor (enum LPCSH_SENSOR_ID hif_SensorId, uint8_t enable)

  Enable/disable given virtual sensor.
- void Algorithm_SetSensorSampleRateUsec (enum LPCSH_SENSOR_ID hif_SensorId, uint32_t sensor↩ SampleRateUsec)

  Sets given sensor's sample rate in micro-seconds.
- void Algorithm_SetSensorBatch (enum LPCSH_SENSOR_ID hif_SensorId, int flags, uint32_t periodUsec, uint32_t timeoutUsec)

  Sets given sensor's batch period in microseconds, batch timeout in microseconds, and batch flags.
- void Algorithm_i2c_completed (uint16_t status)

  indicates sensor I2C transaction state
- uint32_t Algorithm_GetSensorSampleRateUsec (enum LPCSH_SENSOR_ID hif_SensorId)

  Gets given sensor's sample rate in micro-seconds.
- uint32_t Algorithm_Process (PhysicalSensor_t *pSens)

  Process the received sensor data.
- uint32_t Algorithm_bgProcess (uint32_t *pEstSleepTime)

  Run background process if needed by fusion library for example processing callback routines, calibration routines etc.
- void Algorithm_setCalibration (const unsigned char *pData, size_t size)

  Set a flag to apply for the calibration process.
- int Algorithm_getCalibrationData (void *dcd, uint16_t *size)

  Reads DCD data from memory, if available.

3.5.1 Detailed Description

3.5.2 Function Documentation

3.5.2.1 uint32_t Algorithm_bgProcess (uint32_t * *pEstSleepTime*)

Run background process if needed by fusion library for example processing callback routines, calibration routines etc.

Parameters

pEstSleepTime	: Updates estimated sleep time with the time remaining

Returns

returns sleep indicator 0 means it is Ok to sleep else sleep is not Ok

3.5.2.2 uint32_t Algorithm_EnableSensor (enum LPCSH_SENSOR_ID *hif_SensorId,* uint8_t *enable*)

Enable/disable given virtual sensor.

Parameters

hif_SensorId	: sensor ID defined in host interface
enable	: 0 disable, 1 enable

Returns

Operation completion status 0 - Success, else error conditions

3.5.2.3 int Algorithm_getCalibrationData (void * *dcd,* uint16_t * *size*)

Reads DCD data from memory, if available.

Returns

dcd : address of DCD data or NULL if not present

3.5.2.4 void Algorithm_GetPhysicalSensorList (struct lpcsh_sensor_info_t *list[],* uint8_t * *size*)

returns the list of physical sensors that were deected

Parameters

list	pointer for storing the list
size	in/out param, in: allocated space for storing list, out: actual number of sensors returned in list

Returns

none

3.5.2.5 uint32_t Algorithm_GetSensorSampleRateUsec (enum LPCSH_SENSOR_ID *hif_SensorId*)

Gets given sensor's sample rate in micro-seconds.

Parameters

hif_SensorId	: sensor ID defined in host interface

Returns

sensorSampleRateUsec : 0 disable , >0 sample

3.5.2.6 void Algorithm_GetSensorWakeupSettings (struct lpcsh_configuration_data_t * *pConfiguration*)

returns sensor hub wakeup sensor configuration bit map per enum LPCSH_SENSOR_ID

Parameters

pConfiguration	: storage for returned configuration

Returns

none

3.5.2.7 void Algorithm_i2c_completed (uint16_t *status*)

indicates sensor I2C transaction state

Parameters

status	: status of I2C transaction

Returns

none

3.5.2.8 void Algorithm_Init (void)

Initialize algorithm module.

Returns

none

3.5.2.9 uint32_t Algorithm_Process (PhysicalSensor_t * *pSens*)

Process the received sensor data.

Parameters

pSens	: Pointer to physical sensor which received data

Returns

Ok to sleep indicator

3.5.2.10 uint32_t Algorithm_SensorTimingUpdated (uint64_t *sensorEnableMask*)

Update the new sensor timings to algo and physical sensors.

Parameters

sensorEnableMask	: bit mask, bit set per enabled sensor id LPCSH_SENSOR_ID position

Returns

Operation completion status 0 - Success, else error conditions

3.5.2.11 void Algorithm_setCalibration (const unsigned char * *pData,* size_t *size*)

Set a flag to apply for the calibration process.

Returns

none

3.5.2.12 void Algorithm_SetSensorBatch (enum LPCSH_SENSOR_ID *hif_SensorId,* int *flags,* uint32_t *periodUsec,* uint32_t *timeoutUsec*)

Sets given sensor's batch period in microseconds, batch timeout in microseconds, and batch flags.

Parameters

hif_SensorId	: sensor ID defined in host interface
flags	: batch flags
periodUsec	: batch period in microseconds (0 == sensor off)
timeoutUsec	: batch timeout in microseconds (0 == no batching)

Returns

none

3.5.2.13 void AIgorithm_SetSensorSampleRateUsec (enum LPCSH_SENSOR_ID *hif_SensorId,* uint32_t *sensorSampleRateUsec*)

Sets given sensor's sample rate in micro-seconds.

Parameters

hif_SensorId	: sensor ID defined in host interface
sensorSample↩ RateUsec	: 0 disable , >0 sample rate

Returns

none

3.6 Sensor Hub: Breathing LED HW Driver

Functions

- void BLED_HW_ClearPending (void)

  Clear pending timer IRQ match.
- void BLED_HW_UpdateDC (uint32_t DC)

  Update PWM duty cycle.
- void BLED_HW_TimerInit (void)

  Enable and initialize timer.
- void BLED_HW_ConfigPins (uint8_t channelMask)

  Enable PWM pins defined by channelMask.
- void BLED_HW_StartTimer (void)

  Start timer.
- void BLED_HW_TimerDeInit (void)

  De-initialize and disable timer.
- uint32_t BLED_HW_TimerGetBaseClock (void)

  Get base clock frequency of timer.
- void BLED_HW_UpdatePrescaler (uint8_t prescaler)

  Update prescaler of timer.

3.6.1 Detailed Description

3.6.2 Function Documentation

3.6.2.1 void BLED_HW_ClearPending (void)

Clear pending timer IRQ match.

Returns

none

3.6.2.2 void BLED_HW_ConfigPins (uint8_t *channelMask*)

Enable PWM pins defined by channelMask.

Parameters

channelMask	mask indicating channels to be enabled, 1 is enabled

Returns

none

3.6.2.3 void BLED_HW_StartTimer (void)

Start timer.

Returns

none

3.6.2.4 void BLED_HW_TimerDeInit (void)

De-initialize and disable timer.

Returns

none

3.6.2.5 uint32_t BLED_HW_TimerGetBaseClock (void)

Get base clock frequency of timer.

Returns

Base clock of timer in Hz

3.6.2.6 void BLED_HW_TimerInit (void)

Enable and initialize timer.

Returns

none

3.6.2.7 void BLED_HW_UpdateDC (uint32_t *DC*)

Update PWM duty cycle.

Parameters

DC	duty cycle for PWM

Returns

none

3.6.2.8 void BLED_HW_UpdatePrescaler (uint8_t *prescaler*)

Update prescaler of timer.

Parameters

prescaler	Timer prescaler

Returns

none

3.7 Sensor Hub: Resource manager interface

Data Structures

- struct _RESMGR_CTRL

  Resource Manager Structure.

Typedefs

- typedef enum _RESMGR_MODES **ResMgr_Mode_T**
- typedef struct _RESMGR_CTRL ResMgr_Ctrl_T

  Resource Manager Structure.

Enumerations

- enum _RESMGR_MODES { RESMGR_STARTUP = 0, RESMGR_NORMAL_MODE, RESMGR_PLL_MODE, RESMGR_PWRDOWN_MODE }

Functions

- void ResMgr_Init (void)

  Initialize resource manager module.
- void ResMgr_EnterPowerDownMode (uint32_t estSleepTime)

  Puts the system in power down mode.
- void ResMgr_EnterNormalMode (void)

  Sets the system to run off of IRC.
- void ResMgr_EnterPLLMode (void)

  Sets the system to run off of PLL.
- int32_t ResMgr_EstimateSleepTime (void)

  Estimate how long we can sleep.

Variables

- ResMgr_Ctrl_T g_ResMgr

3.7.1 Detailed Description

3.7.2 Enumeration Type Documentation

3.7.2.1 enum _RESMGR_MODES

Enumerator

RESMGR_STARTUP Startup mode

RESMGR_NORMAL_MODE Normal mode (IRC 12Mhz)

RESMGR_PLL_MODE PLL mode (84Mhz from PLL)

RESMGR_PWRDOWN_MODE Power down mode

3.7.3 Function Documentation

3.7.3.1 void ResMgr_EnterNormalMode (void)

Sets the system to run off of IRC.

Returns

none

3.7.3.2 void ResMgr_EnterPLLMode (void)

Sets the system to run off of PLL.

Returns

none

3.7.3.3 void ResMgr_EnterPowerDownMode (uint32_t *estSleepTime*)

Puts the system in power down mode.

Returns

none

3.7.3.4 int32_t ResMgr_EstimateSleepTime (void)

Estimate how long we can sleep.

Returns

none

3.7.3.5 void ResMgr_Init (void)

Initialize resource manager module.

Returns

none

3.7.4 Variable Documentation

3.7.4.1 ResMgr_Ctrl_T g_ResMgr

Data structure which hold resource manager control data

3.8 Sensor Hub: Kernel Watchdog timer interface

Data Structures

- struct _SH_TIMER_CTRL

  Kernel Timer Structure.

Typedefs

- typedef struct _SH_TIMER_CTRL KernelTimer_Ctrl_t

  Kernel Timer Structure.

Enumerations

- enum LPCSH_TIMER_ID { **LPCSH_TIMER_ID_FIRST** = 0, **LPCSH_TIMER_ID_FUSION** = LPCSH_TIMER_ID_FIRST, **LPCSH_TIMER_ID_SENSOR**, **LPCSH_TIMER_ID_COUNT** }

  Enumeration for timer id's.

Functions

- void Kernel_Init (void)

  Initialize kernel sub-system.

Variables

- KernelTimer_Ctrl_t **g_Timer**

3.8.1 Detailed Description

3.8.2 Typedef Documentation

3.8.2.1 typedef struct _SH_TIMER_CTRL KernelTimer_Ctrl_t

Kernel Timer Structure.

Data structure which holds kernel timer function pointers structure

3.8.3 Function Documentation

3.8.3.1 void Kernel_Init (void)

Initialize kernel sub-system.

Returns

none

3.9 Sensor Hub: Sensor I2C Drivers

Functions

- bool dev_i2c_init (void)

  Initialize I2C bus connected sensors.
- void dev_i2c_delay (unsigned int msec)

  API for I2C delay.
- signed char dev_i2c_write (unsigned char dev_addr, unsigned char reg_addr, unsigned char *reg_data, unsigned char cnt)

  I2C API to write data to a specific register of a I2C device.
- signed char dev_i2c_read (unsigned char dev_addr, unsigned char reg_addr, unsigned char *reg_data, unsigned char cnt)

  I2C API to read data from specific register of a I2C device.
- char dev_i2c_readOnly (unsigned char dev_addr, unsigned char *reg_data, unsigned char cnt)

  I2C API to read data at address (assumes auto address increment)
- uint32_t dev_i2c_sleepOk (void)

  I2C API to check if transmission is active and decide whether to sleep or not.

3.9.1 Detailed Description

3.9.2 Function Documentation

3.9.2.1 void dev_i2c_delay (unsigned int *msec*)

API for I2C delay.

Parameters

msec	: Delay for specified number of miliseconds

Returns

Nothing

API for I2C delay.

Returns

Nothing

3.9.2.2 bool dev_i2c_init (void)

Initialize I2C bus connected sensors.

Returns

True if initialized
True if initalized

3.9.2.3 signed char dev_i2c_read (unsigned char *dev_addr,* unsigned char *reg_addr,* unsigned char * *reg_data,* unsigned char *cnt*)

I2C API to read data from specific register of a I2C device.

Parameters

dev_addr	: I2C device address
reg_addr	: Register in device to read
reg_data	: Pointer to buffer where data will be stored
cnt	: Number of bytes to read

Returns

Status : LPC_OK on success.

I2C API to read data from specific register of a I2C device.

Returns

Nothing

3.9.2.4 char dev_i2c_readOnly (unsigned char *dev_addr,* unsigned char * *reg_data,* unsigned char *cnt*)

I2C API to read data at address (assumes auto address increment)

Parameters

dev_addr	: I2C device address
reg_data	: Pointer to buffer where data will be stored
cnt	: Number of bytes to read

Returns

Status : LPC_OK on success.

I2C API to read data at address (assumes auto address increment)

Returns

Nothing

3.9.2.5 uint32_t dev_i2c_sleepOk (void)

I2C API to check if transmission is active and decide whether to sleep or not.

Returns

I2C busy status : 0 - not busy, Ok to sleep, 1 - busy, not Ok to sleep

3.9.2.6 signed char dev_i2c_write (unsigned char *dev_addr,* unsigned char *reg_addr,* unsigned char * *reg_data,* unsigned char *cnt*)

I2C API to write data to a specific register of a I2C device.

Parameters

dev_addr	: I2C device address
reg_addr	: Register in device to write to
reg_data	: Pointer to data to write
cnt	: Number of bytes to write

Returns

Status : LPC_OK on success.

I2C API to write data to a specific register of a I2C device.

Returns

Nothing

3.10 Sensor Hub: Sensor API and Defines

Data Structures

- struct _PhysSensor_CTRL
  Public Sensor data / control interface.
- struct PhysSensorCtrl
  Sensor interface.

Typedefs

- typedef enum _PhysSensorId PhysSensorId_t
  Physical sensor ID's.
- typedef enum _PhysSensorMode PhysSensorMode_t
  Sensor data acquisition modes.
- typedef struct _PhysSensor_CTRL PhysicalSensor_t
  Public Sensor data / control interface.
- typedef struct PhysSensorCtrl PhysSensorCtrl_t
  Sensor interface.

Enumerations

- enum _PhysSensorId {
  PHYS_ACCEL_ID = 0, PHYS_GYRO_ID, PHYS_MAG_ID, PHYS_PRESSURE_ID,
  PHYS_PROXI_ID, PHYS_AMBIENT_ID, **PHYS_MAX_ID** }
  Physical sensor ID's.
- enum _PhysSensorMode { PHYS_MODE_IRQ = 0, PHYS_MODE_POLL }
  Sensor data acquisition modes.

Functions

- int32_t PhysSensorCtrl_init (PhysicalSensor_t *pSens)
  Initialize sensor.
- int32_t PhysSensorCtrl_read (PhysicalSensor_t *pSens)
  Read sensor data and store in pSens->data.
- int32_t PhysSensorCtrl_activate (PhysicalSensor_t *pSens, bool enable)
  Activate / de-activate sensor.
- int32_t PhysSensorCtrl_setDelay (PhysicalSensor_t *pSens, uint32_t usec)
  Set data acquisition rate.
- void PhysSensors_Init (void)
  Initialize all physical sensors.
- void PhysSensors_Enable (PhysSensorId_t sensorId, uint32_t enable)
  Enable/disable physical sensor.
- uint32_t Sensor_process (void)
  Process sensor tasks.
- void set_sensors_used_by_library (bool busy)
  Method for library to mark sensor bus busy while being used by library.
- bool get_sensors_used_by_library (void)
  Method for library to mark sensor bus busy while being used by library.
- void execute_sensor_transaction (const struct ShSensor_RegisterDump_t *transaction_request)
  request for sensor bus transaction.

Variables

- PhysicalSensor_t * **g_phySensors** [PHYS_MAX_ID]

3.10.1 Detailed Description

3.10.2 Typedef Documentation

3.10.2.1 typedef struct PhysSensorCtrl PhysSensorCtrl_t

Sensor interface.

Global array of sensors

3.10.3 Enumeration Type Documentation

3.10.3.1 enum _PhysSensorId

Physical sensor ID's.

Enumerator

PHYS_ACCEL_ID Accelerometer sensor ID

PHYS_GYRO_ID Gyro sensor ID

PHYS_MAG_ID Magnetometer sensor ID

PHYS_PRESSURE_ID Pressure sensor ID

PHYS_PROXI_ID Proximity sensor ID

PHYS_AMBIENT_ID Ambient light sensor ID

3.10.3.2 enum _PhysSensorMode

Sensor data acquisition modes.

Enumerator

PHYS_MODE_IRQ Data read triggered by IRQ

PHYS_MODE_POLL Data read triggered by timed interval

3.10.4 Function Documentation

3.10.4.1 void execute_sensor_transaction (const struct ShSensor_RegisterDump_t * *transaction_request*)

request for sensor bus transaction.

Parameters

transaction_request	- details of transaction

Returns

none.

3.10.4.2 bool get_sensors_used_by_library (void)

Method for library to mark sensor bus busy while being used by library.

Returns

boolean to mark bus busy (true) or not busy (false).

3.10.4.3 int32_t PhysSensorCtrl_activate (PhysicalSensor_t * *pSens*, bool *enable*)

Activate / de-activate sensor.

Parameters

pSens	: Pointer to sensor control structure
enable	1 to enable, 0 to disable

Returns

Activation status (0 on success)

Note

Interface detailing (* activate)() api in PhysSensorCtrl structure. Do NOT implement directly!

3.10.4.4 int32_t PhysSensorCtrl_init (PhysicalSensor_t * *pSens*)

Initialize sensor.

Parameters

pSens	: Pointer to sensor control structure

Returns

Init status (0 on success)

Note

Interface detailing (* int)() api in PhysSensorCtrl structure. Do NOT implement directly!

3.10.4.5 int32_t PhysSensorCtrl_read (PhysicalSensor_t * *pSens*)

Read sensor data and store in pSens->data.

Parameters

pSens	: Pointer to sensor control structure

Returns

Read status (0 on success)

Note

Interface detailing (* read)() api in PhysSensorCtrl structure. Do NOT implement directly!

3.10.4.6 int32_t PhysSensorCtrl_setDelay (PhysicalSensor_t * *pSens*, uint32_t *usec*)

Set data acquisition rate.

Parameters

pSens	: Pointer to sensor control structure
usec	: Data acquisition rate in uSec

Returns

Status (0 on success)

Note

Interface detailing (∗ setDelay)() api in PhysSensorCtrl structure. Do NOT implement directly!

3.10.4.7 void PhysSensors_Enable (PhysSensorId_t *sensorId,* uint32_t *enable*)

Enable/disable physical sensor.

Parameters

sensorId	: Physical sensor id
enable	: If 0 disable else enable

Returns

none

3.10.4.8 void PhysSensors_Init (void)

Initialize all physical sensors.

Returns

none

3.10.4.9 uint32_t Sensor_process (void)

Process sensor tasks.

Returns

Return 0 when it is ok to sleep.

3.10.4.10 void set_sensors_used_by_library (bool *busy*)

Method for library to mark sensor bus busy while being used by library.

Parameters

busy	- boolean to mark bus busy (true) or not busy (false)

Returns

none.

3.11 Sensor Hub: Framework configuration

Chapter 4

Data Structure Documentation

4.1 __CFG_FW_HOSTIF_I2CCtrl_t Struct Reference

Data Fields

- Hostif_I2CState_t **hostI2CState**
- uint8_t * **txBuff**
- uint8_t **lenBuff** [4]
- uint8_t **rxBuff** [RX_LENGTH]
- uint8_t **i2cBusy**
- uint8_t **txLength**

The documentation for this struct was generated from the following file:

- sh_framework/src/hostif_i2c.c

4.2 __FILE Struct Reference

Data Fields

- int **handle**

The documentation for this struct was generated from the following file:

- sh_framework/src/sysinit.c

4.3 _PhysSensor_CTRL Struct Reference

Public Sensor data / control interface.

`#include <sensors.h>`

Data Fields

- const struct PhysSensorCtrl * hw
- int16_t data16 [4]
- uint32_t ts_lastSampleUs

- uint32_t ts_nextSampleUs
- uint8_t id
- uint8_t enabled
- uint8_t irq_pending
- uint8_t mode
- uint32_t period
- void * libHandle

4.3.1 Detailed Description

Public Sensor data / control interface.

4.3.2 Field Documentation

4.3.2.1 int16_t _PhysSensor_CTRL::data16[4]

Sample data

4.3.2.2 uint8_t _PhysSensor_CTRL::enabled

flag indicating if the sensor is enabled

4.3.2.3 const struct PhysSensorCtrl* _PhysSensor_CTRL::hw

Sensor control interface structure

4.3.2.4 uint8_t _PhysSensor_CTRL::id

PhysSensorId_t type

4.3.2.5 uint8_t _PhysSensor_CTRL::irq_pending

irq is pending

4.3.2.6 void* _PhysSensor_CTRL::libHandle

Fusion library's handle for the physical sensor

4.3.2.7 uint8_t _PhysSensor_CTRL::mode

Sensor mode, 0 - IRQ, 1 - polling

4.3.2.8 uint32_t _PhysSensor_CTRL::period

sample period in microseconds

4.3.2.9 uint32_t _PhysSensor_CTRL::ts_lastSampleUs

Timestamp when last sample is captured in uSec

4.3.2.10 uint32_t _PhysSensor_CTRL::ts_nextSampleUs

Next sample timestamp in uSec

The documentation for this struct was generated from the following file:

- sh_framework/inc/sensors.h

4.4 _RESMGR_CTRL Struct Reference

Resource Manager Structure.

`#include <kernel_res_mgr.h>`

Data Fields

- ResMgr_Mode_T currMode
- ResMgr_Mode_T prevMode
- uint32_t fSkipSleep
- uint32_t fVoiceTrigSleep

4.4.1 Detailed Description

Resource Manager Structure.

4.4.2 Field Documentation

4.4.2.1 ResMgr_Mode_T _RESMGR_CTRL::currMode

Current mode of resource manager

4.4.2.2 uint32_t _RESMGR_CTRL::fSkipSleep

Flag to enable sleep when processing is finished

4.4.2.3 uint32_t _RESMGR_CTRL::fVoiceTrigSleep

Flag for voice trigger to enter low power state

4.4.2.4 ResMgr_Mode_T _RESMGR_CTRL::prevMode

Previous mode of resource manager

The documentation for this struct was generated from the following file:

- sh_framework/inc/kernel_res_mgr.h

4.5 _SH_TIMER_CTRL Struct Reference

Kernel Timer Structure.

`#include <kernel_timer.h>`

Data Fields

- uint32_t sleep_threshold_us
- uint32_t pwrDown_threshold_us
- uint32_t bgProc_threshold_us
- void(∗const init)(void)
- uint64_t(∗const GetCurrent)(void)
- uint64_t(∗const GetCurrentUsec)(void)
- uint32_t(∗const GetCurrentMsec)(void)
- int32_t(∗const diff)(uint32_t operand1, uint32_t operand2)
- uint32_t(∗const add)(uint32_t operand1, uint32_t operand2)
- uint32_t(∗const GetResolution)(void)
- void(∗const delayTicks)(uint32_t ticks, void(∗callback)(void), enum LPCSH_TIMER_ID timerId)
- void(∗const delayMs)(uint32_t milliseconds, void(∗callback)(void), enum LPCSH_TIMER_ID timerId)
- void(∗const delayUs)(uint32_t microseconds, void(∗callback)(void), enum LPCSH_TIMER_ID timerId)
- uint32_t(∗const getTicksFromUs)(uint32_t microseconds)
- uint32_t(∗const getTicksFromMs)(uint32_t milliseconds)
- uint32_t(∗const getMsFromTicks)(uint64_t ticks)
- uint32_t(∗const getUsFromTicks)(uint64_t ticks)

4.5.1 Detailed Description

Kernel Timer Structure.

4.5.2 Field Documentation

4.5.2.1 uint32_t(∗const _SH_TIMER_CTRL::add)(uint32_t operand1, uint32_t operand2)

Function pointer for calculating sum of timestamps

4.5.2.2 uint32_t _SH_TIMER_CTRL::bgProc_threshold_us

Background Process Threshold in microseconds

4.5.2.3 void(∗const _SH_TIMER_CTRL::delayMs)(uint32_t milliseconds, void(∗callback)(void), enum LPCSH_TIMER_ID timerId)

Function pointer for delay in milliseconds

4.5.2.4 void(∗const _SH_TIMER_CTRL::delayTicks)(uint32_t ticks, void(∗callback)(void), enum LPCSH_TIMER_ID timerId)

Function pointer for delay in timer ticks

4.5.2.5 void(∗const _SH_TIMER_CTRL::delayUs)(uint32_t microseconds, void(∗callback)(void), enum LPCSH_TIMER_ID timerId)

Function pointer for delay in microseconds

4.5.2.6 int32_t(∗const _SH_TIMER_CTRL::diff)(uint32_t operand1, uint32_t operand2)

Function pointer for calculating time difference

4.5.2.7 uint64_t(∗const _SH_TIMER_CTRL::GetCurrent)(void)

Function pointer for reading current time in ticks

4.5.2.8 uint32_t(∗const _SH_TIMER_CTRL::GetCurrentMsec)(void)

Function pointer for reading current time in milli seconds

4.5.2.9 uint64_t(∗const _SH_TIMER_CTRL::GetCurrentUsec)(void)

Function pointer for reading current time in micro seconds

4.5.2.10 uint32_t(∗const _SH_TIMER_CTRL::getMsFromTicks)(uint64_t ticks)

Converts a kernel tick time to a millisecond count

4.5.2.11 uint32_t(∗const _SH_TIMER_CTRL::GetResolution)(void)

Function pointer to read timer resolution

4.5.2.12 uint32_t(∗const _SH_TIMER_CTRL::getTicksFromMs)(uint32_t milliseconds)

Converts a passed milli second count to a kernel tick time

4.5.2.13 uint32_t(∗const _SH_TIMER_CTRL::getTicksFromUs)(uint32_t microseconds)

Converts a passed micro second count to a kernel tick time

4.5.2.14 uint32_t(∗const _SH_TIMER_CTRL::getUsFromTicks)(uint64_t ticks)

Converts a kernel tick time to a microsecond count

4.5.2.15 void(∗const _SH_TIMER_CTRL::init)(void)

Initialization function pointer

4.5.2.16 uint32_t _SH_TIMER_CTRL::pwrDown_threshold_us

Power Down Threshold in microseconds

4.5.2.17 uint32_t _SH_TIMER_CTRL::sleep_threshold_us

Sleep Threshold in microseconds

The documentation for this struct was generated from the following file:

- sh_framework/inc/kernel_timer.h

4.6 Hostif_SPICtrl_t Struct Reference

Data Fields

- Hostif_SPIState_t **hostSPIstate**
- uint8_t * **txBuff**
- uint8_t **lenBuff** [4]
- uint8_t **rxBuff** [RX_LENGTH]
- uint8_t **spiBusy**
- uint8_t **txLength**

The documentation for this struct was generated from the following file:

- sh_framework/src/hostif_spi.c

4.7 LPCSH_BLED_LED_t Struct Reference

Breathing LED Structure.

`#include <bled_driver.h>`

Data Fields

- uint8_t matchNum
- uint8_t SCTOutNum
- uint8_t port
- uint8_t pin
- uint8_t IOCON_Func

4.7.1 Detailed Description

Breathing LED Structure.

4.7.2 Field Documentation

4.7.2.1 uint8_t LPCSH_BLED_LED_t::IOCON_Func

IOCON FUNC value for selected match channel

4.7.2.2 uint8_t LPCSH_BLED_LED_t::matchNum

Match channel to control PWM of LED

4.7.2.3 uint8_t LPCSH_BLED_LED_t::pin

Pin LED is connected to

4.7.2.4 uint8_t LPCSH_BLED_LED_t::port

Port LED is connected to

4.7.2.5 uint8_t LPCSH_BLED_LED_t::SCTOutNum

SCTOut number to control PWM of LED

The documentation for this struct was generated from the following file:

- lib_lpcsh/inc/bled_driver.h

4.8 LPCSH_BLED_PWM_Slope_t Struct Reference

Data Fields

- uint32_t targetDC
- uint16_t cnt

4.8.1 Field Documentation

4.8.1.1 uint16_t LPCSH_BLED_PWM_Slope_t::cnt

...in cnt∗10 ms

4.8.1.2 uint32_t LPCSH_BLED_PWM_Slope_t::targetDC

Duty cycle to reach

The documentation for this struct was generated from the following file:

- lib_lpcsh/inc/bled_driver.h

4.9 lpcsh_sensor_oc_node_s Struct Reference

Data Fields

- uint64_t **timestamp64**
- struct lpcsh_sensor_node **lpcsh_sensor_node**

The documentation for this struct was generated from the following file:

- lib_lpcsh/inc/hostif_sensors_fifo.h

4.10 PhysSensorCtrl Struct Reference

Sensor interface.

`#include <sensors.h>`

Data Fields

- int32_t(∗const init)(PhysicalSensor_t ∗pSens)
- int32_t(∗const read)(PhysicalSensor_t ∗pSens)
- int32_t(∗const activate)(PhysicalSensor_t ∗pSens, bool enable)
- int32_t(∗const setDelay)(PhysicalSensor_t ∗pSens, uint32_t usec)

4.10.1 Detailed Description

Sensor interface.

4.10.2 Field Documentation

4.10.2.1 int32_t(∗const PhysSensorCtrl::activate)(PhysicalSensor_t ∗pSens, bool enable)

See also

PhysSensorCtrl_activate()

4.10.2.2 int32_t(∗const PhysSensorCtrl::init)(PhysicalSensor_t ∗pSens)

See also

PhysSensorCtrl_init()

4.10.2.3 int32_t(∗const PhysSensorCtrl::read)(PhysicalSensor_t ∗pSens)

See also

PhysSensorCtrl_read()

4.10.2.4 int32_t(∗const PhysSensorCtrl::setDelay)(PhysicalSensor_t ∗pSens, uint32_t usec)

See also

PhysSensorCtrl_setDelay()

The documentation for this struct was generated from the following file:

- sh_framework/inc/sensors.h

4.11 sensorFIFO_s Struct Reference

`#include <hostif_fifo.h>`

Data Fields

- uint16_t head_wr
- uint16_t tail_rd
- uint8_t ∗ data
- uint16_t fifosize
- uint8_t reportNow
- uint8_t reserved0
- uint64_t sensorsMask
- uint64_t lastTimestamp
- struct lpcsh_sensor_node oldestTimestampNode
- uint8_t(∗ preProcessRecord)(const void ∗record, uint16_t ∗length, uint64_t timestamp, uint8_t ∗sensorId, struct sensorFIFO_s ∗∗fifo)
- void(∗ **process_Timestamp**)(struct sensorFIFO_s ∗fifo, uint16_t ∗length, uint64_t timestamp)

- uint8_t(∗ **postProcessRecord**)(uint8_t sensorId, struct sensorFIFO_s ∗fifo, uint64_t timestamp)
- void(∗ **startReportToHost**)(struct sensorFIFO_s ∗fifo, uint64_t batchTime)
- uint8_t(∗ **makeRoomForRecord**)(struct sensorFIFO_s ∗fifo, int32_t space_needed)
- uint16_t(∗ **copyDataToOutputBuffer**)(struct sensorFIFO_s ∗fifo, uint8_t ∗pBuffer)

4.11.1 Detailed Description

Defines the generic FIFO data structure for all circular FIFO in framework. the specific handling of FIFO data is customized via function pointers.

4.11.2 Field Documentation

4.11.2.1 uint8_t∗ sensorFIFO_s::data

Sensor data FIFO's (wakeup and non-wakeup fifo's)

4.11.2.2 uint16_t sensorFIFO_s::fifosize

Fifo size

4.11.2.3 uint16_t sensorFIFO_s::head_wr

write data index (where to put data arriving from sensor)

4.11.2.4 uint64_t sensorFIFO_s::lastTimestamp

full timestamp of last inserted sample

4.11.2.5 struct lpcsh_sensor_node sensorFIFO_s::oldestTimestampNode

keeps track of last read/removed timestamp

4.11.2.6 uint8_t(∗ sensorFIFO_s::preProcessRecord)(const void ∗record,uint16_t ∗length,uint64_t timestamp,uint8_t ∗sensorId,struct sensorFIFO_s ∗∗fifo)

pointer to store the FIFO pointer where record is to be stored

4.11.2.7 uint8_t sensorFIFO_s::reportNow

set when Host interrupt is set for reporting this FIFO

4.11.2.8 uint8_t sensorFIFO_s::reserved0

Reserved

4.11.2.9 uint64_t sensorFIFO_s::sensorsMask

Mask for all sensors directed to this FIFO cleared when FLUSH done based on time stamp.

4.11.2.10 uint16_t sensorFIFO_s::tail_rd

read data index (where to retrieve when sending to Host) note: if rd==wr, then buffer is empty

The documentation for this struct was generated from the following file:

- lib_lpcsh/inc/hostif_fifo.h

4.12 sensorProperties_s Struct Reference

Data Fields

- lpcsh_sensor_oc_node_t * **onChangeData**
- SensorReportingMode_t **reporting_mode**
- uint16_t **sensDataSize**

The documentation for this struct was generated from the following file:

- lib_lpcsh/inc/hostif_sensors_fifo.h

4.13 SensorReportingMode_t Struct Reference

Data Fields

- uint8_t **trigger_mode**: 3
- uint8_t **wake_up_mode**: 1
- uint8_t **supported**: 1

The documentation for this struct was generated from the following file:

- lib_lpcsh/inc/hostif_sensors_fifo.h

4.14 sleepEvent_t Struct Reference

Data Fields

- void(* **callback**)(void)
- uint64_t **wakeupTimeInTicks**
- uint8_t **active**

The documentation for this struct was generated from the following file:

- sh_framework/src/kernel_wdt_rtc_timer.c