SENSEnuts

WSN sensation

API REFERENCE GUIDE

Revision 6.1.2

First Release

Contents

About This Manual

This manual lists all the APIs needed for programming SENSEnuts devices. The document should be studied after user is comfortable with Quick Start Guide.

The APIs are divided according to the categories, for example, all the APIs related to the peripheral hardware of the Radio Module falls into “Peripheral Hardware Specific Calls”. There is no specific order to study the APIs. This document must be used as a reference manual while writing your own applications. It is advised to go through the demo examples to get familiar with the programming methodology used in SENSEnuts WSN platform.

1. Peripheral Hardware Specific Calls

1.1 DIO Functions (dio.h)

i) `void setPin(uint8 dioVal)`

Description

Function to set a digital input/output pin specified by dioVal. The DIOs are used for some special functions. The list is as under:

DIO1: power to SNHTP module
DIO2: LED on Radio module
DIO4: critical interrupt from temperature sensor
DIO5: critical interrupt from light sensor
DIO6: PC communication interface
DIO7: PC communication interface
DIO8: flag pin to update availability of accelerometer data
DIO9: communication with GPS
DIO10: flag pin to update availability of accelerometer data
DIO11: communication with GPS/pressure threshold interrupt
DIO13: switch to power GPS
DIO14: clock to sensors connected on i2c interface
DIO15: data to/from sensors connected on i2c interface
DIO16: interrupt from humidity higher threshold
DIO17: interrupt from humidity lower threshold

Parameters

`dioVal`: the DIO pin to be set (from 0 to 19)

Implementation

setPin(1) shall set (logic 1/Vcc) DIO 1 on the processor.

ii) `void clearPin(uint8 dioVal)`

Description

Clears (unset/ logic 0) the DIO pin specified by `dioval`.

Parameters

`dioVal`: the DIO pin to be cleared (from 0 to 19)

Implementation

clearPin(1) shall clear (logic 0/Ground) DIO 1 on the processor.

iii) `initDioInterrupt(uint8 dioVal)`

Description

Enables the registry of interrupts from the DIO specified by dioVal. The interrupt is registered from the falling edge of the signal. This call also enables the wake Interrupts from the specified DIO to wake the Radio Module from Sleep

Parameters

`dioVal:` the DIO pin from which the interrupt has to be registered (from 0 to 17)

iv) `void ledOn()`

Description

Switches on the LED in Radio Module

Parameters

`NONE`

v) `void ledOff()`

Description

Switches off the LED in Radio Module

Parameters

`NONE`

vi) `void setInput(uint8 dioVal)`

Description

set the DIO specified by dioVal as the input pin

Parameters

`dioVal: value of DIO to be set as input`

vi) `void setoutput(uint8 dioVal)`

Description

sets the DIO specified by dioVal as the output pin

Parameters

`dioVal: value of DIO to be set as output`

1.2 I^{2}C Functions (i2c.h)

I) void i2cInit( )

Description

Initializes the i2c port on the Radio Module. This function must be called whenever user connect an i2c sensor on the Radio Module.

Parameters

NONE

ii) void i2cDisable( )

Description

Disables the i2c port on the Radio Module. Disabling the port when not in use will save energy.

Parameters

NONE

iii) `void i2cWriteCommand(uint8 address)`

Description

This is the write command to the i2c slave connected to the Radio Module. Some of the i2c slave needs a write command where no data has to be sent but the address to refer to the slave.

Parameters

`address:` address of the i2c slave to be addressed

iv) `void i2cWrite(uint8 address, uint8 data)`

Description

This API writes the data specified by the arguments to the i2c slave addressed by its address. This operation does not end with a stop condition. If a user needs to stop the i2c operation after sending one byte to the slave, he/she must call i2cStop() command to achieve the same.

Parameters

`address:` address of the i2c slave to be addressed (7 bit address)

`data:` 8 bit data to be sent to the i2c slave

v) `void i2cWriteAndStop(uint8 data)`

Description

This function sends the data to i2c slave (addressed using i2cWrite function) and stops the i2c operation to release the i2c bus.

Parameters

data: 8 bit data to be sent to addressed i2c slave

Implementation

i2cWriteAndStop(0x33);

2. MAC functions

2.1 General Functions (mac.h)

i) `void macInit()`

Description

Initializes the MAC layer for wireless communication setup

Parameters

None

ii)
```
bool sendDataToMac(uint8 *data, uint8 len, uint16 destAddr,
                   uint8 macHandle, bool isAckReq)
```

Description

Sends data pointed by "data" of length "len" to the MAC layer with destination address specified by "destAddr" and specifying if acknowledgement is required or not Sends data pointed by "data" of length "len" to the MAC layer with destination address specified by "destAddr" and specifying if acknowledgement is required or not. If acknowledgement is required, macHandle acts as current packet identifier. If the packet gets dropped, value sent as macHandle is returned by mac layer based on which user can track which packet got dropped.

Parameters

`*data:`	Pointer to the starting address of data to be sent to MAC
`len:`	The length of data to be sent to MAC

`destAddr`: Destination address where data to be sent

`macHandle`: Current packet identifier, can take value between 0-255

`isAckReq`: TRUE if acknowledgment if requied

FALSE if acknowledgment not requied

2.2 Coordinator Functions (coordinator.h)

```
i) void panCoordinatorInit()
```

Description

This function initlizes pancoordinator of network.

Parameters

```
NONE
```

```
ii) void startCoordinator(void)
```

Description

This function starts the coordinator once energy scan/active scan is done

Parameters

```
NONE
```

```
iii) void handleEnergyScanResponse(MAC_MlmeDcfmInd_s *psMlmeInd)
```

Description

This function directs the coordinator to go through each channel, select the best 1 and inform mac that scanning is complete

```
iv) void handleNodeAssociation(MAC_MlmeDcfmInd_s *psMlmeInd)
```

Description

handle association request sent by nearby node

2.3 genMac Functions (genMac.h)

i) `void genMacInit()`

Description

This function initializes the mac layer of nodes which are not Pan Coordinator

Parameters

`None`

ii) `void handleActiveScanResponse(MAC_MlmeDcfmInd_s *psMlmeInd)`

Description

This function handles indicator from mac about active scan

iii) `void handleAssociateResponse(MAC_MlmeDcfmInd_s *psMlmeInd)`

Description

This function handles response from pan coordinator regarding association

iv) bool `isAssociated()`

Description

This function checks whether the node is able to associate with a coordinator or not

Parameters

`NONE`

3. Phy Layer Functions (phy.h)

3.1 `void setTransmitPower(int32 powerLevel)`

Description

This function sets the transmission power of the transceiver. There are some predefined levels that can be chosen from to set the power.

Parameters

`powerLevel`: -32, -20, -9, 2.5 (in dbm)

3.2 `void setCcaMode(uint8 ccaMode)`

Description

This function sets the Clear Channel Access mode.

Parameters

`ccaMode:`	0x01 : ED only
	0x10 : Carrier Sense only
	0x11 : Carrier Sense with ED

4. Routing Functions (routing.h)

`4.1 void routingInit()`

Description

This function initializes the routing protocol which is specified inside config.h file inside the source folder

Parameters

`NONE`

`4.2 int8_routingSendData(uint8 *data, uint8 len, uint16 destAddr)`

Description

Send data packet pointed by “data” of length “len” to destination address “destAddr” via route found by the routing protocol

Parameters

`data:`	`data packet to be sent`
`len:`	`length of data packet to be sent`
`destAddr:`	`address of the destination to which the data has to be sent`

4.3 void routingReceivePacket(uint8* payload,uint8 length, uint16 prevAddr, uint16 curAddr,uint8 linkQuality)

Description

This function saves the data packet (which has to be relayed) received by the routing protocol at the locaiton pointed by “payload”. The function also displays the address from which the data is being received and the link quality.

Parameters

payload:	location at which the packet has to be saved
length:	length of the packet received from the network
prevAddr:	address of the previous node from which packet is received
curAddr:	address sent by previous hop as the next hop address
linkQuality:	link quality of the signal

4.4 void routingHandleError()

Description

This fuction resets the node in case there is any error in routing process

Parameters

NONE

4.5 void userReceiveDataPacket(uint8* payload,uint8 length,uint16 prevAddr,uint8 linkQuality)

Description

This function handles the received packet task whenever a packet is received by the mote. payload is the pointer to the location where the incoming data of length “length” from node ID prevAddr” and link quality “linkQuality” is received.

Parameters

*payload:	location at which the packet has to be saved
length:	length of the packet received from the network
prevAddr:	address of the previous node from which packet is received
linkQuality:	link quality of the signal

5. Sensor Functions

5.1 Light Sensor Functions (lightSensor.h)

```
i) void lightSensorInit()
```

Description

This function initializes the communication interface of the light sensor and the Radio Module.

Parameters

NONE

```
ii) void setLightThreshold(uint16 high, uint16 low)
```

Description

This function sets the upper and lower threshold of the light sensor. This enables the sensor to generate an interrupt on one of the DIOs on Radio Module and inform about the crossing of the upper or lower threshold.

Parameters

`high`	sets the higher (upper) threshold in “lux”
`low`	sets the lower threshold in “lux”

```
iii) uint16 readLux(void)
```

Description

This function reads the light intesity in lux and saves it in a variable. The function returns a 16 bit value.

Parameters

`NONE`

Implementation

```
uint16 lightIntensity;

lightIntensity=readLux();
```

```
iv) void clearLightINterrupt()
```

Description

This function clears the interrupt generated by the light sensor whenever a threshold is crossed

Parameters

```
NONE
```

5.2 Temperature Sensor Functions (tmpSensor.h)

```
i) void tmpInit()
```

Description

This function initializes the communication interface between temperature sensor and Radio Module

Parameters

NONE

```
ii) uint8 readTmp()
```

Description

This function reads the temperature data from the temperature sensor and returns it in 8-bit format

Parameters

NONE

```
iii) void setTmpThreshold (uint8 high, uint8 low)
```

Description

This function sets the threshold value of temperature on the temperature sensor. It generates an interrupt (logic 0) on one of the DIOs and the interrupt remains active until the temperature falls below the lower threshold specified by “low”

Parameters

high:	temperature in degree Celsius at which alarm has to be set
low:	temperature to turn off the alarm which is set when temperature jumps above value specified by high

iv) `float readTmpFloat( )`

Description

This function reads the temperature value in float. It is not recommended to use this function as this increases the flash memory requirement

Parameters

NONE

5.3 Humidity Sensor Functions (humiditySensor.h)

i) `void humInit()`

Description

This function enables the power to SNHTP module

Parameters

`NONE`

ii) `void setUpperHumidThreshold(uint8 thresholdOn,uint8 thresholdOff)`

Description

This function sets the upper humidity threshold level. Whenever the humidity crosses the threshold value specified by "thresholdOn", it asserts an interrupt which remains active until the humidity level falls below "thresholdOff"

Parameters

`thresholdOn:`	humidity value in %RH to assert an interrupt
`thresholdOff:`	humidity value in %RH to disable the interrupt

iii) `void setLowerHumidThreshold(uint8 thresholdOn,uint8 thresholdOff)`

Description

This function sets the lower humidity threshold level. Whenever the humidity falls below the threshold value specified by "thresholdOn", it asserts an interrupt which remains active until the humidity level rises above "thresholdOff"

Parameters

`thresholdOn:`	humidity value in %RH to assert an interrupt

`thresholdOff:` humidity value in %RH to disable the interrupt

5.4 Pressure and Temperature sensor Functions (tempPressure.h)

i) `void pressureInit()`

Description

This function enables the power to the SNHTP module

Parameters

NONE

ii) `void readTempPressure(uint32* pressure, uint16* temperature)`

Description

This function reads the temperature and pressure values from the sensors on SNHTP module and save the results in the locations pointed by *pressure and *temperature

Parameters

`*pressure:` location where the pressure results (in hPa) has to be stored

`*temperature:` location where the temperature results (in °C) has to be stored

Implementation

```
uint32 pressure;

uint16 temperature;

readTempPressure(&pressure, &temperature);
```

iii) `void tempPressureDisable()`

Description

This function disables the temperature and pressure sensor on SNHTP module. It does not disconnect the power supply from the sensors but sends the same in low power mode

Parameters

NONE

5.5 GPS Functions (gps.h)

i) `void gpsInit()`

Description

This function initializes the gps device on SNGAP module by enabling power to it and the communication interface between GPS device and the Radio Module

Parameters

NONE

ii) `void gpsConfigure()`

Description

This function configures the GPS module to stop reporting any data to the Radio Module.

Parameters

NONE

iii) `void transferToGps(uint8 *data, uint8 len)`

Description

This function tranfers the data pointed by "data" of length "len" to GPS.

Parameters

`*data`: memory location where data to be sent to GPS is stored

`len`: length of data (in bytes) to be sent to GPS module implementation

```
uint8 stopGps[]= { 0xB5, 0x62, 0x06, 0x04, 0x04, 0x00, 0x00, 0x00,
                   0x08, 0x00, 0x16, 0x74};
```

```
transferToGps(stopGps,12);
```

The above implementation sends a command to GPS and shuts down the GPS activity on it.

iv) `void gpsRead()`

Description

This function collects the GPS data, saves it in a buffer and thereafter displays it on GUI

Parameters

NONE

`v) void disableGps()`

Description

This function stops the GPS activities on GPS module without turning off the GPS device. This is important funtion which helps to save power by turning off the power hungry operations performed by GPS device.

Parameters

`NONE`

`vi) void enableGps()`

Description

This function enables the GPS activities on the GPS module. This is a hot start function which helps the module to attain a fix quickly after it is disabled by disableGps() function.

Parameters

`NONE`

5.6 Passive Infrared Functions (dio.h)

`i)void enablePir()`

`Description`

This function enables the reporting of a PIR event

Parameters

`NONE`

6. Task Management Functions (task.h)

`i) uint8 addTask(uint8 taskAdder,uint8 taskType,uint32 time)`

Description

This function adds a user defined time bound task which is executed at the end of the time specified by “time” in the parameters

Parameters

`taskAdder` specifies the task (user or routing)
USER for user defined tasks
ROUTING for routing tasks

`taskType` specifies the type of the task (can take 8-bit unsigned integer values) `time` specifies the time after which the task has to be executed

usage: 100*MILLI_SECONDS	to specify time in milli seconds
100*SECONDS	to specify time in seconds
100*HOURS	to specify time in hours
10*DAYS	to specify time in days

Implementaiton

```
addTask(USER,40,5*SECONDS);
```

```
ii) void userTaskHandler(uint8 taskType)
```

Description

This function handles the task added by the user at the time specified using addTask function

Parameters

`taskType` not to be specified by the user. This is passed automatically from addTask function

```
iii) void userCriticalTaskHandler(uint8 critTaskType)
Description
```

This function gets called automatically whenever there is a critical event. The critical event may be from the predefined critical settings in SENSEnuts which are enabled by a user, or a user may define additional settings as well

Parameters

`critTaskType` This is passed automatically and is not to be specified by the user

```
iv) void routingTaskHandler(uint8 taskType)
```

Description

This function gets called automatically to handle the routing tasks added using addTask funtion.

Parameters

`taskType` this is added automatically from addTask function and should not be defined

```
v) void userReceiveDataPacket(uint8* payload,uint8 length,uint16
                     prevAddr,uint8 linkQuality)
```

Description

This function gets called when a packet is received by the mote from network. What has to be done when the packet is received is to be defined in this function

Parameters

`*payload:`	location in the memory where the packet has to be saved
`length:`	length of the packet received from the neetwork
`prevAddr:`	address of the previous node (one hop behind) from which packet is received
`linkQuality:`	quality of signal

7. Miscellaneous Functions

7.1 Gateway Communication Functions (pcInterface.h)

```
i) void sendToPcInit()
```

Description

This function Initializes the communication between a computer and the node connected to the computer via USB cable with the help of Gateway Module

Parameters

NONE

```
ii) void updateSTLdb(uint16 nodeId,uint16 light,uint8 temperature)
```

Description

Updates the database of temperature (uint8) and light (uint16) sensor on SNSTL module with the readings received from the node specified by “nodeId”.

Parameters

`nodeId`	ID of the node from which the packet has been sent
`light`	light intensity in “lux” received from the light sensor module
`temperature`	temperature (8-bit) received from the temperature sensor on STL module

```
iii) void updateSTLdbf(uint16 nodeId,uint16 light,uint8 temperature)
```

Description

Updates the database of temperature (uint16) and light (uint16) sensor on SNSTL module with the readings received from the node specified by “nodeId”.

Parameters

`nodeId`	ID of the node from which the packet has been sent
`light`	light intensity in “lux” received from the light sensor module

`temperature`	temperature (16-bit) received from the temperature sensor on STL module

`iv)void debug(uint8* string, uint8 length)`

Description

This function prints the message on “Print Window”, pointed by “string” and length “length”. Note that the maximum allowed size of the string is 224 bytes.

Parameters

`*string`	address of the location in memory from where data has to be sent
`length`	length of message to be printed on Print Window

`v)uint8 readByteFromPc()`

Description

This function reads the byte coming from PC. This function should be called is called in interrupt context and should be used in pcInterruptHandler(), in order to read the byte coming from PC

Parameters

NONE

```
vi)void updateMetdb(uint16 nodeId, uint16 humidity, uint32 pressure,
                    uint16 temperature)
```

Descriptions

This function updates the database from humidity, pressure and temperature sensor on SNHTP module from the node specified by “nodeId” with “humidity” %RH, “pressure” hPa and “temperature” °C.

Parameters

`nodeId:`	ID of the node from which the data is received
`humidity:`	humidity string received from humidity sensor
`pressure:`	pressure string received from pressure sensor
`temperature:`	temperature string received from temperature sensor

`vi)void updateGapData(uint16 nodeId)`

Description

This function updates the GPS data (GGA format of NMEA protocol) to PC

Parameters

```
NONE
```

7.2 Node Functions (node.h)

```
i)uint16 getNodeId()
```

Description

This function Returns the ID of the node on which the call is being made. It is extraced from the MAC address of the node (basically the two least significant bytes

Parameters

```
NONE
```

```
ii)float remainingBattery()
```

Description

This function returns the voltage level supplied by battery to the Radio Module

Parameters

```
NONE
```

```
iii) void startNode()
```

Description

It is the first function, just like "void main()" in C which is called on boot. All the initializations of the sensors and other hardware features that has to be used should be perfromed here

Parameters

```
NONE
```

```
iv) void msdelay(uint32 milliSeconds)
```

Description

This function generates a delay in milliseconds specified.

Parameters

`milliSeconds:` number of milliseconds for which the delay has to be generated

Implementaiton

```
msdelay(5000);
```

The above call will result into a delay of 5000 milliseconds, i.e., 5 seconds

8. Application Functions

```
i) void startNode()
```

Description

It is the first function, just like “void main()” in C which is called on boot. All the initializations of the sensors and other hardware features that has to be used should be perfromed here

Parameters

`NONE`

```
ii) void userTaskHandler(uint8 taskType)
```

Description

This function handles the task added by the user at the time specified using addTask function

Parameters

`taskType` not to be specified by the user. This is passed automatically from addTask function

```
iii) void userReceiveDataPacket(uint8* payload,uint8 length,uint16
                              prevAddr,uint8 linkQuality)
```

Description

This function gets called when a packet is received by the mote from network. What has to be done when the packet is received is to be defined in this function

Parameters

`*payload:` location in the memory where the packet has to be saved

`length:` length of the packet received from the network

`prevAddr:` address of the previous node (one hop behind) from which packet is received

`linkQuality:` quality of signal

```
iv) void userCriticalTaskHandler(uint8 critTaskType)
Description
```

This function gets called automatically whenever there is a critical event. The critical event may be from the predefined critical settings (configurable) in SENSEnuts which are enabled by a user.

Parameters

`critTaskType` This is passed automatically and is not to be specified by the user

v) `pcInterruptHandler()`

Description

This function gets called if receive from pc is enabled and a byte is received from pc.

Parameters

NONE

Revision History

Version	Date	Comments
6.1.2	09/19/2015	First Release

Important Notice

Limited warranty and liability - Information in this document is believed to be accurate and reliable. However, Eigen Technologies does not give any representations or warranties, expressed or implied, as to the accuracy or completeness of such information and shall have no liability for the consequences of use of such information. Eigen Technologies takes no responsibility for the content in this document if provided by an information source outside of Eigen Technologies.

In no event shall Eigen Technologies be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or ework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory. Notwithstanding any damages that customer might incur for any reason whatsoever, Eigen Technologies' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the *Terms and conditions of commercial sale* of Eigen Technologies.

Right to make changes – Eigen Technologies reserves the right to make changes to information published in this document, including without limitation specifications and product descriptions, at any time and without notice. This document supersedes and replaces all information supplied prior to the publication hereof.

Suitability for use - NXP Semiconductors products are not designed, authorized or warranted to be suitable for use in life support, life-critical or safety-critical systems or equipment, nor in applications where failure or malfunction of an NXP Semiconductors product can reasonably be expected to result in personal injury, death or severe property or environmental damage. NXP Semiconductors and its suppliers accept no liability for inclusion and/or use of NXP Semiconductors products in such equipment or applications and therefore such inclusion and/or use is at the customer's own risk.

Applications - Applications that are described herein for any of these products are for illustrative purposes only. Eigen Technologies makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification. Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products. NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.