SSN Firmware

Generated by Doxygen 1.8.18

Chapter 1

Smart Sense Node (SSN) Firmware Documentation

1.1 Introduction

The following document summarises the key functions, variables and APIs for the Smart Sense Node Firmware developed for PIC32MX170F256B

1.2 Installation

1.2.1 Step 1: Opening the box

etc...

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

File Index

3.1 File List

Here is a list of all files with brief descriptions:

Chapter 4

Class Documentation

4.1 pseudo_clock Struct Reference

`#include <pseudo_rtcc.h>`

Public Attributes

- uint8_t hours
- uint8_t minutes
- uint8_t seconds
- uint8_t day
- uint8_t month
- uint8_t year

4.1.1 Detailed Description

A structure for maintaining the global clock of Smart Sense Node

4.1.2 Member Data Documentation

4.1.2.1 day

`uint8_t pseudo_clock::day`

4.1.2.2 hours

`uint8_t pseudo_clock::hours`

4.1.2.3 minutes

```
uint8_t pseudo_clock::minutes
```

4.1.2.4 month

```
uint8_t pseudo_clock::month
```

4.1.2.5 seconds

```
uint8_t pseudo_clock::seconds
```

4.1.2.6 year

```
uint8_t pseudo_clock::year
```

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

- src/SSN_API/Drivers/PSEUDO_RTCC/pseudo_rtcc.h

Chapter 5

File Documentation

5.1 main.c File Reference

```
#include "src/SSN_API/SSN_API.h"
```

Macros

- #define _SUPPRESS_PLIB_WARNING
- #define _DISABLE_OPENADC10_CONFIGPORT_WARNING

Functions

- void __ISR (_TIMER_1_VECTOR, IPL4SOFT)
- int main ()

Variables

- SOCKET SSN_UDP_SOCKET
- uint8_t SSN_SERVER_IP [ ] = {192, 168, 1, 100}
- uint16_t SSN_SERVER_PORT = 9999
- uint32_t SSN_SENT_MESSAGES_COUNTER = 0
- uint8_t interrupts_per_second = 2
- uint8_t half_second_counter = 0
- uint8_t report_counter = 0
- uint8_t SSN_CURRENT_STATE
- uint8_t SSN_REPORT_INTERVAL = 1
- uint8_t SSN_CONFIG [EEPROM_CONFIG_SIZE]
- uint8_t SSN_CURRENT_SENSOR_RATINGS [4]
- uint8_t SSN_CURRENT_SENSOR_THRESHOLDS [4]
- uint8_t SSN_CURRENT_SENSOR_MAXLOADS [4]
- uint8_t Machine_load_currents [NO_OF_MACHINES] = {0}
- uint8_t Machine_load_percentages [NO_OF_MACHINES] = {0}
- uint8_t Machine_status [NO_OF_MACHINES] = {MACHINE_RESET_SENTINEL_STATE, MACHINE_RESET_SENTINEL_ST
  MACHINE_RESET_SENTINEL_STATE, MACHINE_RESET_SENTINEL_STATE}

- pseudo_clock Machine_status_timestamp [NO_OF_MACHINES]
- uint32_t Machine_status_duration [NO_OF_MACHINES] = {0}
- uint8_t SSN_UDP_SOCKET_NUM = 1
- uint8_t SSN_DEFAULT_MAC [ ] = {0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF}
- uint8_t SSN_MAC_ADDRESS [6] = {0}
- uint8_t temperature_bytes [2]
- uint8_t relative_humidity_bytes [2]
- uint8_t temp_humidity_recv_status
- uint8_t abnormal_activity
- uint8_t i

5.1.1 Macro Definition Documentation

5.1.1.1 _DISABLE_OPENADC10_CONFIGPORT_WARNING

```
#define _DISABLE_OPENADC10_CONFIGPORT_WARNING
```

5.1.1.2 _SUPPRESS_PLIB_WARNING

```
#define _SUPPRESS_PLIB_WARNING
```

5.1.2 Function Documentation

5.1.2.1 __ISR()

```
void __ISR (
            _TIMER_1_VECTOR ,
            IPL4SOFT  )
```

Half-Second interrupt that controls our send message routine of the SSN. Half-second and not one second is because we can not set an interrupt of up to 1 second with the current clock of the SSN. We only start this interrupt service once we have Ethernet configured and all self-tests are successful. The message to be sent is constructed every half a second in the main function and only reported to the server after every "SSN_REPORT_INTERVAL" seconds.

5.1.2.2 main()

```
int main ( )
```

The main loop of SSN operation. It calls the following functions in order:

- Sets up all required peripherals
- Runs system tests for checking:
  1. EEPROM Read/Write
  2. Temperature and Humidity Sensor
  3. Ethernet Physical Connection
- Finds MAC address in EEPROM; if available, assigns it to SSN; if not available, assigns default MAC address to SSN
- Sets up Ethernet connection using which ever MAC address was selected
- If using default MAC address, SSN sends periodic GET_MAC requests to SSN Server until it successfully retrieves one and resets self
- Waits for five seconds for new Current Sensor Configurations from SSN Server. These cannot be reprogrammed after this five seconds window
- If new configurations received, assigns them to SSN, writes them to EEPROM and proceeds
- If new configurations not received, finds Current Sensor Configurations in EEPROM; if available, assigns them to SSN; if not available SSN sends periodic GET_CONFIG requests to SSN Server until it successfully retrieves one and writes them to EEPROM and proceeds
- SSN sends periodic GET_TimeOfDay requests to SSN Server until it successfully retrieves it and proceeds
- SSN starts global clock and half-second periodic interrupt
- SSN calculates machine status update and ambient conditions every 100 milliseconds. The ISR sends the status update after every ${SSN_REPORT_INTERVAL} seconds

5.1.3 Variable Documentation

5.1.3.1 abnormal_activity

```
uint8_t abnormal_activity
```

SSN abnormal activity bit

5.1.3.2 half_second_counter

```
uint8_t half_second_counter = 0
```

Counter variable for counting half seconds per second

5.1.3.3 i

```
uint8_t i
```

SSN loop variable

5.1.3.4 interrupts_per_second

```
uint8_t interrupts_per_second = 2
```

Counter variable for interrupts per second

5.1.3.5 Machine_load_currents

```
uint8_t Machine_load_currents[NO_OF_MACHINES] = {0}
```

SSN machine load currents array

5.1.3.6 Machine_load_percentages

```
uint8_t Machine_load_percentages[NO_OF_MACHINES] = {0}
```

SSN machine load percentages array

5.1.3.7 Machine_status

```
uint8_t Machine_status[NO_OF_MACHINES] = {MACHINE_RESET_SENTINEL_STATE, MACHINE_RESET_SENTINEL_STATE,
MACHINE_RESET_SENTINEL_STATE, MACHINE_RESET_SENTINEL_STATE}
```

SSN machine status array initialized to a Sentinel or Reset state

5.1.3.8 Machine_status_duration

```
uint32_t Machine_status_duration[NO_OF_MACHINES] = {0}
```

SSN machine status duration array for holding the number of seconds for which the machines have been in the current state

5.1.3.9 Machine_status_timestamp

```
pseudo_clock Machine_status_timestamp[NO_OF_MACHINES]
```

SSN machine timestamps for recording since when the machines have been in the current states

5.1.3.10 relative_humidity_bytes

```
uint8_t relative_humidity_bytes[2]
```

SSN relative humidity reading bytes

5.1.3.11 report_counter

```
uint8_t report_counter = 0
```

Counter variable for counting after how many intervals to send the status update

5.1.3.12 SSN_CONFIG

```
uint8_t SSN_CONFIG[EEPROM_CONFIG_SIZE]
```

SSN current sensor configurations

5.1.3.13 SSN_CURRENT_SENSOR_MAXLOADS

```
uint8_t SSN_CURRENT_SENSOR_MAXLOADS[4]
```

SSN machine maximum loads for calculating percentage loads on machines

5.1.3.14 SSN_CURRENT_SENSOR_RATINGS

```
uint8_t SSN_CURRENT_SENSOR_RATINGS[4]
```

SSN current sensor ratings

5.1.3.15 SSN_CURRENT_SENSOR_THRESHOLDS

```
uint8_t SSN_CURRENT_SENSOR_THRESHOLDS[4]
```

SSN machine thresholds for deciding IDLE state

5.1.3.16 SSN_CURRENT_STATE

```
uint8_t SSN_CURRENT_STATE
```

Current State of the SSN. There is no state machine of the SSN but we still use this variable to keep track at some instances

5.1.3.17 SSN_DEFAULT_MAC

```
uint8_t SSN_DEFAULT_MAC[] = {0xAA, 0xBB, 0xCC, 0xDD, 0xEE, 0xFF}
```

SSN default MAC address. This is the same for all SSNs

5.1.3.18 SSN_MAC_ADDRESS

```
uint8_t SSN_MAC_ADDRESS[6] = {0}
```

SSN current MAC address. May hold the default MAC or the one received from SSN Server. The last two bytes are the SSN Identity

5.1.3.19 SSN_REPORT_INTERVAL

```
uint8_t SSN_REPORT_INTERVAL = 1
```

Report Interval of SSN set according to the configurations passed to the SSN

5.1.3.20 SSN_SENT_MESSAGES_COUNTER

```
uint32_t SSN_SENT_MESSAGES_COUNTER = 0
```

A counter to maintain how many messages have been sent from SSN to Server since wakeup

5.1.3.21 SSN_SERVER_IP

```
uint8_t SSN_SERVER_IP[] = {192, 168, 1, 100}
```

SSN Server Address

5.1.3.22 SSN_SERVER_PORT

```
uint16_t SSN_SERVER_PORT = 9999
```

SSN Server PORT

5.1.3.23 SSN_UDP_SOCKET

```
SOCKET SSN_UDP_SOCKET
```

Our SSN UDP communication socket

5.1.3.24 SSN_UDP_SOCKET_NUM

```
uint8_t SSN_UDP_SOCKET_NUM = 1
```

SSN UDP socket number

5.1.3.25 temp_humidity_recv_status

```
uint8_t temp_humidity_recv_status
```

SSN temperature and humidity reading successful/unsuccessful status bit

5.1.3.26 temperature_bytes

```
uint8_t temperature_bytes[2]
```

SSN temperature sensor reading bytes

5.2 src/SSN_API/Communication/Communication.c File Reference

```
#include "Communication.h"
```

Functions

- void SendMessage (uint8_t SSN_Socket, uint8_t *SSN_SERVER_IP, uint16_t SSN_SERVER_PORT, uint8_t *message_to_send, uint8_t ssn_message_to_send_size)
- void Send_GETMAC_Message (uint8_t *NodeID, uint8_t SSN_Socket, uint8_t *SSN_SERVER_IP, uint16_t SSN_SERVER_PORT)
- void Send_GETCONFIG_Message (uint8_t *NodeID, uint8_t SSN_Socket, uint8_t *SSN_SERVER_IP, uint16_t SSN_SERVER_PORT)
- void Send_ACKCONFIG_Message (uint8_t *NodeID, uint8_t SSN_Socket, uint8_t *SSN_SERVER_IP, uint16_t SSN_SERVER_PORT, uint8_t *SSN_CONFIG)
- void Send_GETTimeOfDay_Message (uint8_t *NodeID, uint8_t SSN_Socket, uint8_t *SSN_SERVER_IP, uint16_t SSN_SERVER_PORT)
- void Send_STATUSUPDATE_Message (uint8_t *NodeID, uint8_t SSN_Socket, uint8_t *SSN_SERVER_IP, uint16_t SSN_SERVER_PORT, uint8_t *temperature_bytes, uint8_t *relative_humidity_bytes, uint8_t *Machine_load_currents, uint8_t *Machine_load_percentages, uint8_t *Machine_status, uint32_t *Machine_status_duration, pseudo_clock *Machine_status_timestamp, uint32_t ssn_uptime_in_seconds, uint8_t abnormal_activity)
- void Receive_MAC (uint8_t SSN_Socket, uint8_t *SSN_SERVER_IP, uint16_t SSN_SERVER_PORT)
- uint8_t Receive_CONFIG (uint8_t SSN_Socket, uint8_t *SSN_SERVER_IP, uint16_t SSN_SERVER_PORT, uint8_t *SSN_CONFIG, uint8_t *SSN_REPORT_INTERVAL, uint8_t *SSN_CURRENT_SENSOR_RATINGS, uint8_t *SSN_CURRENT_SENSOR_THRESHOLDS, uint8_t *SSN_CURRENT_SENSOR_MAXLOADS, uint8_t *Machine_status)
- uint8_t Receive_TimeOfDay (uint8_t SSN_Socket, uint8_t *SSN_SERVER_IP, uint16_t SSN_SERVER_PORT)

5.2.1 Function Documentation

5.2.1.1 Receive_CONFIG()

```
uint8_t Receive_CONFIG (
            uint8_t SSN_Socket,
            uint8_t * SSN_SERVER_IP,
            uint16_t SSN_SERVER_PORT,
            uint8_t * SSN_CONFIG,
            uint8_t * SSN_REPORT_INTERVAL,
            uint8_t * SSN_CURRENT_SENSOR_RATINGS,
            uint8_t * SSN_CURRENT_SENSOR_THRESHOLDS,
            uint8_t * SSN_CURRENT_SENSOR_MAXLOADS,
            uint8_t * Machine_status )
```

Receives a response for Sensor Configurations requested from SSN Server

Parameters

SSN_Socket	UDP socket used by SSN
SSN_SERVER_IP	IP of the destination Server
SSN_SERVER_PORT	Port of the destination Server
SSN_CONFIG	Byte array to save received configurations
SSN_REPORT_INTERVAL	Pointer to a single byte that saves the report interval for SSN status update message
SSN_CURRENT_SENSOR_RATINGS	Byte array for saving current sensor ratings
SSN_CURRENT_SENSOR_THRESHOLDS	Byte array for saving machine thresholds for deciding IDLE state of machines
SSN_CURRENT_SENSOR_MAXLOADS	Byte array for saving machine maximum loads for calculating percentage loads
Machine_status	Byte array of current machine status (ON/OFF/IDLE)

Returns

1 if received, else 0

5.2.1.2 Receive_MAC()

```
void Receive_MAC (
            uint8_t SSN_Socket,
            uint8_t * SSN_SERVER_IP,
            uint16_t SSN_SERVER_PORT )
```

Receives a response for MAC requested from SSN Server

Parameters

SSN_Socket	UDP socket used by SSN
SSN_SERVER_IP	IP of the destination Server
SSN_SERVER_PORT	Port of the destination Server

5.2.1.3 Receive_TimeOfDay()

```
uint8_t Receive_TimeOfDay (
            uint8_t SSN_Socket,
            uint8_t * SSN_SERVER_IP,
            uint16_t SSN_SERVER_PORT )
```

Receives a response for Time of Day requested from SSN Server

Parameters

SSN_Socket	UDP socket used by SSN
SSN_SERVER_IP	IP of the destination Server
SSN_SERVER_PORT	Port of the destination Server

Returns

1 if received, else 0

5.2.1.4 Send_ACKCONFIG_Message()

```
void Send_ACKCONFIG_Message (
            uint8_t * NodeID,
            uint8_t SSN_Socket,
            uint8_t * SSN_SERVER_IP,
            uint16_t SSN_SERVER_PORT,
            uint8_t * SSN_CONFIG )
```

Sends an Acknowledge for Sensor Configuration message received from SSN Server

Parameters

NodeID	Two byte identity of SSN which are the last two bytes of the MAC address
SSN_Socket	UDP socket used by SSN
SSN_SERVER_IP	IP of the destination Server
SSN_SERVER_PORT	Port of the destination Server
SSN_CONFIG	Configurations array previously received from SSN Server

5.2.1.5 Send_GETCONFIG_Message()

```
void Send_GETCONFIG_Message (
            uint8_t * NodeID,
            uint8_t SSN_Socket,
            uint8_t * SSN_SERVER_IP,
            uint16_t SSN_SERVER_PORT )
```

Sends a Sensor Configuration Request message to receive a configuration for SSN to compute statistics of the connected machines

Parameters

NodeID	Two byte identity of SSN which are the last two bytes of the MAC address
SSN_Socket	UDP socket used by SSN
SSN_SERVER_IP	IP of the destination Server
SSN_SERVER_PORT	Port of the destination Server

5.2.1.6 Send_GETMAC_Message()

```
void Send_GETMAC_Message (
            uint8_t * NodeID,
            uint8_t SSN_Socket,
            uint8_t * SSN_SERVER_IP,
            uint16_t SSN_SERVER_PORT )
```

Sends a MAC Request message to receive a custom MAC address for host SSN

Parameters

NodeID	Two byte identity of SSN which are the last two bytes of the MAC address
SSN_Socket	UDP socket used by SSN
SSN_SERVER_IP	IP of the destination Server
SSN_SERVER_PORT	Port of the destination Server

5.2.1.7 Send_GETTimeOfDay_Message()

```
void Send_GETTimeOfDay_Message (
            uint8_t * NodeID,
            uint8_t SSN_Socket,
            uint8_t * SSN_SERVER_IP,
            uint16_t SSN_SERVER_PORT )
```

Sends a Time of Day Request message to SSN Server to receive the current time of day

Parameters

NodeID	Two byte identity of SSN which are the last two bytes of the MAC address
SSN_Socket	UDP socket used by SSN
SSN_SERVER_IP	IP of the destination Server
SSN_SERVER_PORT	Port of the destination Server

5.2.1.8 Send_STATUSUPDATE_Message()

```
void Send_STATUSUPDATE_Message (
             uint8_t * NodeID,
             uint8_t SSN_Socket,
             uint8_t * SSN_SERVER_IP,
             uint16_t SSN_SERVER_PORT,
             uint8_t * temperature_bytes,
             uint8_t * relative_humidity_bytes,
             uint8_t * Machine_load_currents,
             uint8_t * Machine_load_percentages,
             uint8_t * Machine_status,
             uint32_t * Machine_status_duration,
             pseudo_clock * Machine_status_timestamp,
             uint32_t ssn_uptime_in_seconds,
             uint8_t abnormal_activity )
```

Sends a Status Update message to SSN Server containing the machine status and ambient conditions

Parameters

NodeID	Two byte identity of SSN which are the last two bytes of the MAC address
SSN_Socket	UDP socket used by SSN
SSN_SERVER_IP	IP of the destination Server
temperature_bytes	Two byte temperature reading containing the high and low byte of temperature in that order
relative_humidity_bytes	Two byte relative humidity reading containing the high and low byte of humidity in that order
Machine_load_currents	A byte array of machine load currents calculated by SSN
Machine_load_percentages	A byte array of machine load percentages calculated by SSN
Machine_status	A byte array of machine status (ON/OFF/IDLE) calculated by SSN
Machine_status_duration	An array of machine status duration indicating for how long the machines have been in current state
Machine_status_timestamp	An array of machine status timestamp indicating since when the machines have been in current state
ssn_uptime_in_seconds	A four byte integer containing the number of seconds indicating for how long the SSN has been awake
abnormal_activity	A single byte indicating NORMAL or ABNORMAL ambient condition based on temperature and humidity readings

5.2.1.9 SendMessage()

```
void SendMessage (
            uint8_t SSN_Socket,
            uint8_t * SSN_SERVER_IP,
            uint16_t SSN_SERVER_PORT,
            uint8_t * message_to_send,
            uint8_t ssn_message_to_send_size )
```

Sends a message to over UDP

Parameters

SSN_Socket	UDP socket used by SSN
SSN_SERVER_IP	IP of the destination Server
SSN_SERVER_PORT	Port of the destination Server
message_to_send	The byte or char array of message
ssn_message_to_send_size	Message size in bytes

5.3 src/SSN_API/Communication/Communication.h File Reference

```
#include "../global.h"
#include "../Drivers/UART/uart.h"
#include "../Drivers/EEPROM/eeprom.h"
#include "../Drivers/NETWORK/network.h"
#include "../Drivers/CURRENT_SENSOR/current_sensor.h"
#include "../Drivers/Messages/messages.h"
```

Functions

- void SendMessage (uint8_t SSN_Socket, uint8_t *SSN_SERVER_IP, uint16_t SSN_SERVER_PORT, uint8_t *message_to_send, uint8_t ssn_message_to_send_size)
- void Send_GETMAC_Message (uint8_t *NodeID, uint8_t SSN_Socket, uint8_t *SSN_SERVER_IP, uint16_t SSN_SERVER_PORT)
- void Send_GETCONFIG_Message (uint8_t *NodeID, uint8_t SSN_Socket, uint8_t *SSN_SERVER_IP, uint16_t SSN_SERVER_PORT)
- void Send_ACKCONFIG_Message (uint8_t *NodeID, uint8_t SSN_Socket, uint8_t *SSN_SERVER_IP, uint16_t SSN_SERVER_PORT, uint8_t *SSN_CONFIG)
- void Send_GETTimeOfDay_Message (uint8_t *NodeID, uint8_t SSN_Socket, uint8_t *SSN_SERVER_IP, uint16_t SSN_SERVER_PORT)
- void Send_STATUSUPDATE_Message (uint8_t *NodeID, uint8_t SSN_Socket, uint8_t *SSN_SERVER_IP, uint16_t SSN_SERVER_PORT, uint8_t *temperature_bytes, uint8_t *relative_humidity_bytes, uint8_t *Machine_load_currents, uint8_t *Machine_load_percentages, uint8_t *Machine_status, uint32_t *Machine_status_duration, pseudo_clock *Machine_status_timestamp, uint32_t ssn_uptime_in_seconds, uint8_t abnormal_activity)
- void Receive_MAC (uint8_t SSN_Socket, uint8_t *SSN_SERVER_IP, uint16_t SSN_SERVER_PORT)
- uint8_t Receive_CONFIG (uint8_t SSN_Socket, uint8_t *SSN_SERVER_IP, uint16_t SSN_SERVER_PORT, uint8_t *SSN_CONFIG, uint8_t *SSN_REPORT_INTERVAL, uint8_t *SSN_CURRENT_SENSOR_RATINGS, uint8_t *SSN_CURRENT_SENSOR_THRESHOLDS, uint8_t *SSN_CURRENT_SENSOR_MAXLOADS, uint8_t *Machine_status)
- uint8_t Receive_TimeOfDay (uint8_t SSN_Socket, uint8_t *SSN_SERVER_IP, uint16_t SSN_SERVER_PORT)

5.3.1 Function Documentation

5.3.1.1 Receive_CONFIG()

```
uint8_t Receive_CONFIG (
            uint8_t SSN_Socket,
            uint8_t * SSN_SERVER_IP,
            uint16_t SSN_SERVER_PORT,
            uint8_t * SSN_CONFIG,
            uint8_t * SSN_REPORT_INTERVAL,
            uint8_t * SSN_CURRENT_SENSOR_RATINGS,
            uint8_t * SSN_CURRENT_SENSOR_THRESHOLDS,
            uint8_t * SSN_CURRENT_SENSOR_MAXLOADS,
            uint8_t * Machine_status )
```

Receives a response for Sensor Configurations requested from SSN Server

Parameters

SSN_Socket	UDP socket used by SSN
SSN_SERVER_IP	IP of the destination Server
SSN_SERVER_PORT	Port of the destination Server
SSN_CONFIG	Byte array to save received configurations
SSN_REPORT_INTERVAL	Pointer to a single byte that saves the report interval for SSN status update message
SSN_CURRENT_SENSOR_RATINGS	Byte array for saving current sensor ratings
SSN_CURRENT_SENSOR_THRESHOLDS	Byte array for saving machine thresholds for deciding IDLE state of machines
SSN_CURRENT_SENSOR_MAXLOADS	Byte array for saving machine maximum loads for calculating percentage loads
Machine_status	Byte array of current machine status (ON/OFF/IDLE)

Returns

1 if received, else 0

5.3.1.2 Receive_MAC()

```
void Receive_MAC (
            uint8_t SSN_Socket,
            uint8_t * SSN_SERVER_IP,
            uint16_t SSN_SERVER_PORT )
```

Receives a response for MAC requested from SSN Server

Parameters

SSN_Socket	UDP socket used by SSN
SSN_SERVER_IP	IP of the destination Server
SSN_SERVER_PORT	Port of the destination Server

5.3.1.3 Receive_TimeOfDay()

```
uint8_t Receive_TimeOfDay (
            uint8_t SSN_Socket,
            uint8_t * SSN_SERVER_IP,
            uint16_t SSN_SERVER_PORT )
```

Receives a response for Time of Day requested from SSN Server

Parameters

SSN_Socket	UDP socket used by SSN
SSN_SERVER_IP	IP of the destination Server
SSN_SERVER_PORT	Port of the destination Server

Returns

1 if received, else 0

5.3.1.4 Send_ACKCONFIG_Message()

```
void Send_ACKCONFIG_Message (
            uint8_t * NodeID,
            uint8_t SSN_Socket,
            uint8_t * SSN_SERVER_IP,
            uint16_t SSN_SERVER_PORT,
            uint8_t * SSN_CONFIG )
```

Sends an Acknowledge for Sensor Configuration message received from SSN Server

Parameters

NodeID	Two byte identity of SSN which are the last two bytes of the MAC address
SSN_Socket	UDP socket used by SSN
SSN_SERVER_IP	IP of the destination Server
SSN_SERVER_PORT	Port of the destination Server
SSN_CONFIG	Configurations array previously received from SSN Server

5.3.1.5 Send_GETCONFIG_Message()

```
void Send_GETCONFIG_Message (
            uint8_t * NodeID,
            uint8_t SSN_Socket,
            uint8_t * SSN_SERVER_IP,
            uint16_t SSN_SERVER_PORT )
```

Sends a Sensor Configuration Request message to receive a configuration for SSN to compute statistics of the connected machines

Parameters

NodeID	Two byte identity of SSN which are the last two bytes of the MAC address
SSN_Socket	UDP socket used by SSN
SSN_SERVER_IP	IP of the destination Server
SSN_SERVER_PORT	Port of the destination Server

5.3.1.6 Send_GETMAC_Message()

```
void Send_GETMAC_Message (
            uint8_t * NodeID,
            uint8_t SSN_Socket,
            uint8_t * SSN_SERVER_IP,
            uint16_t SSN_SERVER_PORT )
```

Sends a MAC Request message to receive a custom MAC address for host SSN

Parameters

NodeID	Two byte identity of SSN which are the last two bytes of the MAC address
SSN_Socket	UDP socket used by SSN
SSN_SERVER_IP	IP of the destination Server
SSN_SERVER_PORT	Port of the destination Server

5.3.1.7 Send_GETTimeOfDay_Message()

```
void Send_GETTimeOfDay_Message (
            uint8_t * NodeID,
            uint8_t SSN_Socket,
            uint8_t * SSN_SERVER_IP,
            uint16_t SSN_SERVER_PORT )
```

Sends a Time of Day Request message to SSN Server to receive the current time of day

Parameters

NodeID	Two byte identity of SSN which are the last two bytes of the MAC address
SSN_Socket	UDP socket used by SSN
SSN_SERVER_IP	IP of the destination Server
SSN_SERVER_PORT	Port of the destination Server

5.3.1.8 Send_STATUSUPDATE_Message()

```
void Send_STATUSUPDATE_Message (
            uint8_t * NodeID,
            uint8_t SSN_Socket,
            uint8_t * SSN_SERVER_IP,
            uint16_t SSN_SERVER_PORT,
            uint8_t * temperature_bytes,
            uint8_t * relative_humidity_bytes,
            uint8_t * Machine_load_currents,
            uint8_t * Machine_load_percentages,
            uint8_t * Machine_status,
            uint32_t * Machine_status_duration,
            pseudo_clock * Machine_status_timestamp,
            uint32_t ssn_uptime_in_seconds,
            uint8_t abnormal_activity )
```

Sends a Status Update message to SSN Server containing the machine status and ambient conditions

Parameters

NodeID	Two byte identity of SSN which are the last two bytes of the MAC address
SSN_Socket	UDP socket used by SSN
SSN_SERVER_IP	IP of the destination Server
temperature_bytes	Two byte temperature reading containing the high and low byte of temperature in that order
relative_humidity_bytes	Two byte relative humidity reading containing the high and low byte of humidity in that order
Machine_load_currents	A byte array of machine load currents calculated by SSN
Machine_load_percentages	A byte array of machine load percentages calculated by SSN
Machine_status	A byte array of machine status (ON/OFF/IDLE) calculated by SSN
Machine_status_duration	An array of machine status duration indicating for how long the machines have been in current state
Machine_status_timestamp	An array of machine status timestamp indicating since when the machines have been in current state
ssn_uptime_in_seconds	A four byte integer containing the number of seconds indicating for how long the SSN has been awake
abnormal_activity	A single byte indicating NORMAL or ABNORMAL ambient condition based on temperature and humidity readings

5.3.1.9 SendMessage()

```
void SendMessage (
            uint8_t SSN_Socket,
            uint8_t * SSN_SERVER_IP,
            uint16_t SSN_SERVER_PORT,
            uint8_t * message_to_send,
            uint8_t ssn_message_to_send_size )
```

Sends a message to over UDP

Parameters

SSN_Socket	UDP socket used by SSN
SSN_SERVER_IP	IP of the destination Server
SSN_SERVER_PORT	Port of the destination Server
message_to_send	The byte or char array of message
ssn_message_to_send_size	Message size in bytes

5.4 src/SSN_API/Connection/Connection.c File Reference

```
#include "Connection.h"
```

Functions

- uint8_t SetupConnection (uint8_t *SSN_MAC_ADDRESS, uint8_t UDP_SOCKET)

5.4.1 Function Documentation

5.4.1.1 SetupConnection()

```
uint8_t SetupConnection (
            uint8_t * SSN_MAC_ADDRESS,
            uint8_t UDP_SOCKET )
```

Sets up Ethernet Connection for SSN

Parameters

SSN_MAC_ADDRESS	Six byte array containing the six bytes of SSN MAC address
UDP_SOCKET	SSN UDP socket number

Returns

Socket Number, should be the same as UDP_SOCKET if successfully created

5.5 src/SSN_API/Connection/Connection.h File Reference

```
#include "../global.h"
#include "../Drivers/UART/uart.h"
#include "../Drivers/NETWORK/network.h"
```

Functions

- uint8_t SetupConnection (uint8_t *SSN_MAC_ADDRESS, uint8_t UDP_SOCKET)

5.5.1 Function Documentation

5.5.1.1 SetupConnection()

```
uint8_t SetupConnection (
            uint8_t * SSN_MAC_ADDRESS,
            uint8_t UDP_SOCKET )
```

Sets up Ethernet Connection for SSN

Parameters

SSN_MAC_ADDRESS	Six byte array containing the six bytes of SSN MAC address
UDP_SOCKET	SSN UDP socket number

Returns

Socket Number, should be the same as UDP_SOCKET if successfully created

5.6 src/SSN_API/Drivers/CURRENT_SENSOR/current_sensor.c File Reference

```
#include "current_sensor.h"
```

Macros

- #define PARAM1 ADC_MODULE_ON | ADC_FORMAT_INTG | ADC_CLK_AUTO | ADC_AUTO_SAMPL↩ ING_ON
- #define PARAM2 ADC_VREF_AVDD_AVSS | ADC_OFFSET_CAL_DISABLE | ADC_SCAN_ON | ADC_↩ SAMPLES_PER_INT_4 | ADC_ALT_BUF_OFF | ADC_ALT_INPUT_OFF
- #define PARAM3 ADC_CONV_CLK_INTERNAL_RC | ADC_SAMPLE_TIME_15
- #define PARAM4 ENABLE_AN0_ANA | ENABLE_AN1_ANA | ENABLE_AN2_ANA | ENABLE_AN3_ANA
- #define PARAM5 SKIP_SCAN_AN4 | SKIP_SCAN_AN5 | SKIP_SCAN_AN6 | SKIP_SCAN_AN7 | SKIP_↩ SCAN_AN8 | SKIP_SCAN_AN9 | SKIP_SCAN_AN10 | SKIP_SCAN_AN11 | SKIP_SCAN_AN12 | SKIP_↩ SCAN_AN13 | SKIP_SCAN_AN14 | SKIP_SCAN_AN15

Functions

- void open_ADC ()
- void setup_Current_Sensors ()
- uint16_t sample_Current_Sensor_channel (uint8_t channel)
- unsigned char CurrentSensor_Read_RMS (uint8_t channel, uint16_t num_samples, uint8_t SENSOR_RA↩ TING, float SENSOR_TYPE_SCALAR)
- void Calculate_RMS_Current_On_All_Channels (uint8_t *SENSOR_RATINGS, uint16_t num_samples, unsigned char *single_byte_RMS_CURRENTS)
- float Current_VSensor_Read_RMS (uint8_t channel, uint16_t *adc_samples_array, uint16_t num_samples, uint8_t sensor_max_value)
- float Current_CSensor_Read_RMS (uint8_t channel, uint16_t *adc_samples_array, uint16_t num_samples, uint8_t sensor_max_value)
- void Get_Machines_Status_Update (uint8_t *SSN_CURRENT_SENSOR_RATINGS, uint8_t *SSN_CURRENT_SENSOR_TH uint8_t *SSN_CURRENT_SENSOR_MAXLOADS, uint8_t *Machine_load_currents, uint8_t *Machine_load_percentages, uint8_t *Machine_status, uint32_t *Machine_status_duration, pseudo_clock *Machine_status_timestamp)

5.6.1 Macro Definition Documentation

5.6.1.1 PARAM1

```
#define PARAM1 ADC_MODULE_ON | ADC_FORMAT_INTG | ADC_CLK_AUTO | ADC_AUTO_SAMPLING_ON
```

5.6.1.2 PARAM2

```
#define PARAM2 ADC_VREF_AVDD_AVSS | ADC_OFFSET_CAL_DISABLE | ADC_SCAN_ON | ADC_SAMPLES_PER_INT↩
_4 | ADC_ALT_BUF_OFF | ADC_ALT_INPUT_OFF
```

5.6.1.3 PARAM3

```
#define PARAM3 ADC_CONV_CLK_INTERNAL_RC | ADC_SAMPLE_TIME_15
```

5.6.1.4 PARAM4

```
#define PARAM4 ENABLE_AN0_ANA | ENABLE_AN1_ANA | ENABLE_AN2_ANA | ENABLE_AN3_ANA
```

5.6.1.5 PARAM5

```
#define PARAM5 SKIP_SCAN_AN4 | SKIP_SCAN_AN5 | SKIP_SCAN_AN6 | SKIP_SCAN_AN7 | SKIP_SCAN_AN8 |
SKIP_SCAN_AN9 | SKIP_SCAN_AN10 | SKIP_SCAN_AN11 | SKIP_SCAN_AN12 | SKIP_SCAN_AN13 | SKIP_SCAN↩
_AN14 | SKIP_SCAN_AN15
```

5.6.2 Function Documentation

5.6.2.1 Calculate_RMS_Current_On_All_Channels()

```
void Calculate_RMS_Current_On_All_Channels (
            uint8_t * SENSOR_RATINGS,
            uint16_t num_samples,
            unsigned char * single_byte_RMS_CURRENTS )
```

Calculates RMS value of current on all ADC channels

Parameters

SENSOR_RATINGS	Array of sensor ratings to consider for calculating RMS values
num_samples	Number of samples to read on each channel for calculating RMS values
single_byte_RMS_CURRENTS	Byte array to hold single byte integer RMS currents for each channel

5.6.2.2 Current_CSensor_Read_RMS()

```
float Current_CSensor_Read_RMS (
            uint8_t channel,
            uint16_t * current_samples_array,
            uint16_t num_samples,
            uint8_t sensor_max_value )
```

Calculates RMS value of current for a current output current sensor. Expects the ADC channel has already been sampled

Parameters

channel	Which channel to sample
current_samples_array	Array holding the ADC samples
num_samples	Number of samples in the ADC samples array
sensor_max_value	Rating of the current sensor for which RMS is to be computed

Returns

RMS value of current

5.6.2.3 Current_VSensor_Read_RMS()

```
float Current_VSensor_Read_RMS (
            uint8_t channel,
            uint16_t * current_samples_array,
            uint16_t num_samples,
            uint8_t sensor_max_value )
```

Calculates RMS value of current for a voltage output current sensor. Expects the ADC channel has already been sampled

Parameters

channel	Which channel to sample
current_samples_array	Array holding the ADC samples
num_samples	Number of samples in the ADC samples array
sensor_max_value	Rating of the current sensor for which RMS is to be computed

Returns

RMS value of current

5.6.2.4 CurrentSensor_Read_RMS()

```
unsigned char CurrentSensor_Read_RMS (
            uint8_t channel,
            uint16_t num_samples,
            uint8_t SENSOR_RATING,
            float SENSOR_TYPE_SCALAR )
```

Calculates RMS value of current on one ADC channel

Parameters

channel	Channel number to sample; expected 1-4
num_samples	Number of samples to read for calculating RMS value
SENSOR_RATING	Sensor rating to consider for calculating RMS value
SENSOR_TYPE_SCALAR	Sensor type scalar varies with type of sensor, either voltage output or current output

Returns

Single byte integer RMS value of current at this channel

5.6.2.5 Get_Machines_Status_Update()

```
void Get_Machines_Status_Update (
            uint8_t * SSN_CURRENT_SENSOR_RATINGS,
            uint8_t * SSN_CURRENT_SENSOR_THRESHOLDS,
            uint8_t * SSN_CURRENT_SENSOR_MAXLOADS,
            uint8_t * Machine_load_currents,
            uint8_t * Machine_load_percentages,
            uint8_t * Machine_status,
            uint32_t * Machine_status_duration,
            pseudo_clock * Machine_status_timestamp )
```

Calculates RMS values of current for each ADC channel along with the load percentages, machine status, timestamps and duration in state

Parameters

SSN_CURRENT_SENSOR_RATINGS	Byte array for saving current sensor ratings
SSN_CURRENT_SENSOR_THRESHOLDS	Byte array for saving machine thresholds for deciding IDLE state of machines
SSN_CURRENT_SENSOR_MAXLOADS	Byte array for saving machine maximum loads for calculating percentage loads
Machine_load_currents	A byte array of machine load currents calculated by SSN
Machine_load_percentages	A byte array of machine load percentages calculated by SSN
Machine_status	A byte array of machine status (ON/OFF/IDLE) calculated by SSN
Machine_status_duration	An array of machine status duration indicating for how long the machines have been in current state
Machine_status_timestamp	An array of machine status timestamp indicating since when the machines have been in current state

5.6.2.6 open_ADC()

```
void open_ADC ( )
```

Setups up ADC peripheral

5.6.2.7 sample_Current_Sensor_channel()

```
uint16_t sample_Current_Sensor_channel (
            uint8_t channel )
```

Samples specific ADC channel

Parameters

channel	Channel number to sample; expected 1-4

Returns

16-bit ADC sample value

5.6.2.8 setup_Current_Sensors()

```
void setup_Current_Sensors ( )
```

Sets up Current Sensors for SSN

5.7 src/SSN_API/Drivers/CURRENT_SENSOR/current_sensor.h File Reference

```
#include "../../global.h"
#include <stdint.h>
#include <plib.h>
```

Macros

- #define _SUPPRESS_PLIB_WARNING
- #define _DISABLE_OPENADC10_CONFIGPORT_WARNING
- #define NUM_OF_ADC_SAMPLES_FOR_IRMS 400
- #define VOLTAGE_OUTPUT_CURRENT_SENSOR_SCALAR 1
- #define CURRENT_OUTPUT_CURRENT_SENSOR_SCALAR 2

Enumerations

- enum Machine_Status { MACHINE_OFF =0, MACHINE_IDLE, MACHINE_ON, MACHINE_RESET_SENTINEL_STATE }

Functions

- void open_ADC ()
- void setup_Current_Sensors ()
- uint16_t sample_Current_Sensor_channel (uint8_t channel)
- unsigned char CurrentSensor_Read_RMS (uint8_t channel, uint16_t num_samples, uint8_t SENSOR_RA↩ TING, float SENSOR_TYPE_SCALAR)
- void Calculate_RMS_Current_On_All_Channels (uint8_t ∗SENSOR_RATINGS, uint16_t num_samples, unsigned char ∗single_byte_RMS_CURRENTS)
- float Current_VSensor_Read_RMS (uint8_t channel, uint16_t ∗current_samples_array, uint16_t num_↩ samples, uint8_t sensor_max_value)
- float Current_CSensor_Read_RMS (uint8_t channel, uint16_t ∗current_samples_array, uint16_t num_↩ samples, uint8_t sensor_max_value)
- void Get_Machines_Status_Update (uint8_t ∗SSN_CURRENT_SENSOR_RATINGS, uint8_t ∗SSN_CURRENT_SENSOR_TH uint8_t ∗SSN_CURRENT_SENSOR_MAXLOADS, uint8_t ∗Machine_load_currents, uint8_t ∗Machine_load_percentages, uint8_t ∗Machine_status, uint32_t ∗Machine_status_duration, pseudo_clock ∗Machine_status_timestamp)

Variables

- uint32_t MACHINES_STATE_TIME_MARKERS [NO_OF_MACHINES]

5.7.1 Macro Definition Documentation

5.7.1.1 _DISABLE_OPENADC10_CONFIGPORT_WARNING

```
#define _DISABLE_OPENADC10_CONFIGPORT_WARNING
```

5.7.1.2 _SUPPRESS_PLIB_WARNING

```
#define _SUPPRESS_PLIB_WARNING
```

5.7.1.3 CURRENT_OUTPUT_CURRENT_SENSOR_SCALAR

```
#define CURRENT_OUTPUT_CURRENT_SENSOR_SCALAR 2
```

An implementation specific scalar value for current output current sensors

5.7.1.4 NUM_OF_ADC_SAMPLES_FOR_IRMS

```
#define NUM_OF_ADC_SAMPLES_FOR_IRMS 400
```

Number of samples from ADC channel to calculate the Root Mean Square Current

5.7.1.5 VOLTAGE_OUTPUT_CURRENT_SENSOR_SCALAR

```
#define VOLTAGE_OUTPUT_CURRENT_SENSOR_SCALAR 1
```

An implementation specific scalar value for voltage output current sensors

5.7.2 Enumeration Type Documentation

5.7.2.1 Machine_Status

```
enum Machine_Status
```

Enumeration for listing the possible machine states

Enumerator

MACHINE_OFF	
MACHINE_IDLE	
MACHINE_ON	
MACHINE_RESET_SENTINEL_STATE	

5.7.3 Function Documentation

5.7.3.1 Calculate_RMS_Current_On_All_Channels()

```
void Calculate_RMS_Current_On_All_Channels (
            uint8_t * SENSOR_RATINGS,
            uint16_t num_samples,
            unsigned char * single_byte_RMS_CURRENTS )
```

Calculates RMS value of current on all ADC channels

Parameters

SENSOR_RATINGS	Array of sensor ratings to consider for calculating RMS values
num_samples	Number of samples to read on each channel for calculating RMS values
single_byte_RMS_CURRENTS	Byte array to hold single byte integer RMS currents for each channel

5.7.3.2 Current_CSensor_Read_RMS()

```
float Current_CSensor_Read_RMS (
            uint8_t channel,
            uint16_t * current_samples_array,
            uint16_t num_samples,
            uint8_t sensor_max_value )
```

Calculates RMS value of current for a current output current sensor. Expects the ADC channel has already been sampled

Parameters

channel	Which channel to sample
current_samples_array	Array holding the ADC samples
num_samples	Number of samples in the ADC samples array
sensor_max_value	Rating of the current sensor for which RMS is to be computed

Returns

RMS value of current

5.7.3.3 Current_VSensor_Read_RMS()

```
float Current_VSensor_Read_RMS (
            uint8_t channel,
            uint16_t * current_samples_array,
            uint16_t num_samples,
            uint8_t sensor_max_value )
```

Calculates RMS value of current for a voltage output current sensor. Expects the ADC channel has already been sampled

Parameters

channel	Which channel to sample
current_samples_array	Array holding the ADC samples
num_samples	Number of samples in the ADC samples array
sensor_max_value	Rating of the current sensor for which RMS is to be computed

Returns

RMS value of current

5.7.3.4 CurrentSensor_Read_RMS()

```
unsigned char CurrentSensor_Read_RMS (
            uint8_t channel,
            uint16_t num_samples,
            uint8_t SENSOR_RATING,
            float SENSOR_TYPE_SCALAR )
```

Calculates RMS value of current on one ADC channel

Parameters

channel	Channel number to sample; expected 1-4
num_samples	Number of samples to read for calculating RMS value
SENSOR_RATING	Sensor rating to consider for calculating RMS value
SENSOR_TYPE_SCALAR	Sensor type scalar varies with type of sensor, either voltage output or current output

Returns

Single byte integer RMS value of current at this channel

5.7.3.5 Get_Machines_Status_Update()

```
void Get_Machines_Status_Update (
            uint8_t * SSN_CURRENT_SENSOR_RATINGS,
            uint8_t * SSN_CURRENT_SENSOR_THRESHOLDS,
            uint8_t * SSN_CURRENT_SENSOR_MAXLOADS,
            uint8_t * Machine_load_currents,
            uint8_t * Machine_load_percentages,
            uint8_t * Machine_status,
            uint32_t * Machine_status_duration,
            pseudo_clock * Machine_status_timestamp )
```

Calculates RMS values of current for each ADC channel along with the load percentages, machine status, timestamps and duration in state

Parameters

SSN_CURRENT_SENSOR_RATINGS	Byte array for saving current sensor ratings
SSN_CURRENT_SENSOR_THRESHOLDS	Byte array for saving machine thresholds for deciding IDLE state of machines
SSN_CURRENT_SENSOR_MAXLOADS	Byte array for saving machine maximum loads for calculating percentage loads
Machine_load_currents	A byte array of machine load currents calculated by SSN
Machine_load_percentages	A byte array of machine load percentages calculated by SSN
Machine_status	A byte array of machine status (ON/OFF/IDLE) calculated by SSN
Machine_status_duration	An array of machine status duration indicating for how long the machines have been in current state
Machine_status_timestamp	An array of machine status timestamp indicating since when the machines have been in current state

5.7.3.6 open_ADC()

```
void open_ADC ( )
```

Setups up ADC peripheral

5.7.3.7 sample_Current_Sensor_channel()

```
uint16_t sample_Current_Sensor_channel (
            uint8_t channel )
```

Samples specific ADC channel

Parameters

channel	Channel number to sample; expected 1-4

Returns

16-bit ADC sample value

5.7.3.8 setup_Current_Sensors()

```
void setup_Current_Sensors ( )
```

Sets up Current Sensors for SSN

5.7.4 Variable Documentation

5.7.4.1 MACHINES_STATE_TIME_MARKERS

```
uint32_t MACHINES_STATE_TIME_MARKERS[NO_OF_MACHINES]
```

Implementation specific machine status time markers for keeping state timestamps

5.8 src/SSN_API/Drivers/EEPROM/eeprom.c File Reference

```
#include "eeprom.h"
```

Functions

- void open_I2C1 ()
- void setup_EEPROM ()
- void I2C1_wait_while_busy ()
- void I2C1_transmit_start_bit ()
- void I2C1_transmit_stop_bit ()
- void I2C1_transmit_restart_bit ()
- void I2C1_transmit_byte (uint8_t byte)
- uint8_t I2C1_receive_byte ()
- uint8_t EEPROM_Write_BYTE (uint8_t block, uint8_t address, uint8_t data)
- uint8_t EEPROM_Read_BYTE (uint8_t block, uint8_t address)
- uint8_t EEPROM_Write_Array (uint8_t block, uint8_t address, uint8_t *arr, uint8_t size)
- uint8_t EEPROM_Read_Array (uint8_t block, uint8_t address, uint8_t *arr, uint8_t size)
- uint8_t EEPROM_Clear ()
- uint8_t EEPROM_Check ()

5.8.1 Function Documentation

5.8.1.1 EEPROM_Check()

```
uint8_t EEPROM_Check ( )
```

Checks EEPROM read/write by writing and reading a single byte at a pre-defined location

Returns

EEPROM_TEST_PASSED for successful test; **EEPROM_TEST_FAILED** for failure

5.8.1.2 EEPROM_Clear()

```
uint8_t EEPROM_Clear ( )
```

Clears EEPROM

Returns

1 for successfully cleared

5.8.1.3 EEPROM_Read_Array()

```
uint8_t EEPROM_Read_Array (
            uint8_t block,
            uint8_t address,
            uint8_t * arr,
            uint8_t size )
```

Reads a byte array from the EEPROM. Bytes are read from contiguous locations

Parameters

block	Block number to read from (1-4)
address	Address inside the block to start reading from (0-255)
arr	The byte array to write the read values into
size	The number of bytes to read

Returns

1 for read successful

5.8.1.4 EEPROM_Read_BYTE()

```
uint8_t EEPROM_Read_BYTE (
            uint8_t block,
            uint8_t address )
```

Reads a single byte from the EEPROM

Parameters

block	Block number to read from (1-4)
address	Address inside the block to read from (0-255)

Returns

The single byte read from this location

5.8.1.5 EEPROM_Write_Array()

```
uint8_t EEPROM_Write_Array (
            uint8_t block,
            uint8_t address,
            uint8_t * arr,
            uint8_t size )
```

Writes byte array into the EEPROM. Bytes are written at contiguous locations

Parameters

block	Block number to write into (1-4)
address	Address inside the block to start writing from (0-255)
arr	The byte array to write into the EEPROM
size	The number of bytes to write

5.8.1.6 EEPROM_Write_BYTE()

```
uint8_t EEPROM_Write_BYTE (
            uint8_t block,
            uint8_t address,
            uint8_t data )
```

Writes single byte into the EEPROM

Parameters

block	Block number to write into (1-4)
address	Address inside the block to write into (0-255)
data	The single byte to write at this location

5.8.1.7 I2C1_receive_byte()

```
uint8_t I2C1_receive_byte ( )
```

Recieves byte over I2C1

Returns

Single byte read over I2C1

5.8.1.8 I2C1_transmit_byte()

```
void I2C1_transmit_byte (
            uint8_t byte )
```

Transmit single byte over I2C

Parameters

byte	Single byte to transmit

5.8.1.9 I2C1_transmit_restart_bit()

```
void I2C1_transmit_restart_bit ( )
```

Transmit single bit for restarting I2C communication

5.8.1.10 I2C1_transmit_start_bit()

```
void I2C1_transmit_start_bit ( )
```

Transmit single bit for starting I2C communication

5.8.1.11 I2C1_transmit_stop_bit()

```
void I2C1_transmit_stop_bit ( )
```

Transmit single bit for stoping I2C communication

5.8.1.12 I2C1_wait_while_busy()

```
void I2C1_wait_while_busy ( )
```

Waits while I2C1 is busy reading or writing

5.8.1.13 open_I2C1()

```
void open_I2C1 ( )
```

Opens I2C1 peripheral

5.8.1.14 setup_EEPROM()

```
void setup_EEPROM ( )
```

Sets up EEPROM using I2C1

5.9 src/SSN_API/Drivers/EEPROM/eeprom.h File Reference

```
#include "../../global.h"
#include <stdint.h>
#include <stdio.h>
#include <plib.h>
```

Macros

- #define _SUPPRESS_PLIB_WARNING
- #define _DISABLE_OPENADC10_CONFIGPORT_WARNING
- #define EEPROM_24LC08_BASE_ADDRESS 0xA0
- #define EEPROM_24LC08_READ_BIT 0x00
- #define EEPROM_24LC08_WRITE_BIT 0x01
- #define EEPROM_BYTE_READ 0xA1
- #define EEPROM_BYTE_WRITE 0xA0
- #define EEPROM_BLOCK_SIZE 256
- #define EEPROM_TEST_LOCATION 0x50
- #define EEPROM_TEST_VALUE 0x91
- #define EEPROM_CLEAR_VALUE 0xFF
- #define EEPROM_TEST_PASSED 1
- #define EEPROM_TEST_FAILED 0

Enumerations

- enum EEPROM_24LC08_BLOCKS {
  EEPROM_BLOCK_0 =0, EEPROM_BLOCK_1, EEPROM_BLOCK_2, EEPROM_BLOCK_3,
  EEPROM_BLOCK_COUNT }

Functions

- void open_I2C1 ()
- void I2C1_wait_while_busy ()
- void I2C1_transmit_start_bit ()
- void I2C1_transmit_stop_bit ()
- void I2C1_transmit_restart_bit ()
- void I2C1_transmit_byte (uint8_t byte)
- uint8_t I2C1_receive_byte ()
- void setup_EEPROM ()
- uint8_t EEPROM_Write_BYTE (uint8_t block, uint8_t address, uint8_t data)
- uint8_t EEPROM_Read_BYTE (uint8_t block, uint8_t address)
- uint8_t EEPROM_Write_Array (uint8_t block, uint8_t address, uint8_t *arr, uint8_t size)
- uint8_t EEPROM_Read_Array (uint8_t block, uint8_t address, uint8_t *arr, uint8_t size)
- uint8_t EEPROM_Clear ()
- uint8_t EEPROM_Check ()

5.9.1 Macro Definition Documentation

5.9.1.1 _DISABLE_OPENADC10_CONFIGPORT_WARNING

```
#define _DISABLE_OPENADC10_CONFIGPORT_WARNING
```

5.9.1.2 _SUPPRESS_PLIB_WARNING

```
#define _SUPPRESS_PLIB_WARNING
```

5.9.1.3 EEPROM_24LC08_BASE_ADDRESS

```
#define EEPROM_24LC08_BASE_ADDRESS 0xA0
```

EEPROM 24LC08 Address

5.9.1.4 EEPROM_24LC08_READ_BIT

```
#define EEPROM_24LC08_READ_BIT 0x00
```

EEPROM 24LC08 Read bit

5.9.1.5 EEPROM_24LC08_WRITE_BIT

```
#define EEPROM_24LC08_WRITE_BIT 0x01
```

EEPROM 24LC08 Write bit

5.9.1.6 EEPROM_BLOCK_SIZE

```
#define EEPROM_BLOCK_SIZE 256
```

EEPROM 24LC08 block size of 256 bytes

5.9.1.7 EEPROM_BYTE_READ

```
#define EEPROM_BYTE_READ 0xA1
```

EEPROM 24LC08 Read bit combined with address

5.9.1.8 EEPROM_BYTE_WRITE

```
#define EEPROM_BYTE_WRITE 0xA0
```

EEPROM 24LC08 Write bit combined with address

5.9.1.9 EEPROM_CLEAR_VALUE

```
#define EEPROM_CLEAR_VALUE 0xFF
```

EEPROM 24LC08 clear value. All bytes must be set to 0xFF

5.9.1.10 EEPROM_TEST_FAILED

```
#define EEPROM_TEST_FAILED 0
```

EEPROM 24LC08 test failed

5.9.1.11 EEPROM_TEST_LOCATION

```
#define EEPROM_TEST_LOCATION 0x50
```

EEPROM 24LC08 test location for testing operation

5.9.1.12 EEPROM_TEST_PASSED

```
#define EEPROM_TEST_PASSED 1
```

EEPROM 24LC08 test passed

5.9.1.13 EEPROM_TEST_VALUE

```
#define EEPROM_TEST_VALUE 0x91
```

EEPROM 24LC08 test value

5.9.2 Enumeration Type Documentation

5.9.2.1 EEPROM_24LC08_BLOCKS

```
enum EEPROM_24LC08_BLOCKS
```

Our EEPROM chip has four blocks of memory, 256 bytes in each block

Enumerator

EEPROM_BLOCK_0	
EEPROM_BLOCK_1	
EEPROM_BLOCK_2	
EEPROM_BLOCK_3	
EEPROM_BLOCK_COUNT	

5.9.3 Function Documentation

5.9.3.1 EEPROM_Check()

```
uint8_t EEPROM_Check ( )
```

Checks EEPROM read/write by writing and reading a single byte at a pre-defined location

Returns

EEPROM_TEST_PASSED for successful test; **EEPROM_TEST_FAILED** for failure

5.9.3.2 EEPROM_Clear()

```
uint8_t EEPROM_Clear ( )
```

Clears EEPROM

Returns

1 for successfully cleared

5.9.3.3 EEPROM_Read_Array()

```
uint8_t EEPROM_Read_Array (
            uint8_t block,
            uint8_t address,
            uint8_t * arr,
            uint8_t size )
```

Reads a byte array from the EEPROM. Bytes are read from contiguous locations

Parameters

block	Block number to read from (1-4)
address	Address inside the block to start reading from (0-255)
arr	The byte array to write the read values into
size	The number of bytes to read

Returns

1 for read successful

5.9.3.4 EEPROM_Read_BYTE()

```
uint8_t EEPROM_Read_BYTE (
            uint8_t block,
            uint8_t address )
```

Reads a single byte from the EEPROM

Parameters

block	Block number to read from (1-4)
address	Address inside the block to read from (0-255)

Returns

The single byte read from this location

5.9.3.5 EEPROM_Write_Array()

```
uint8_t EEPROM_Write_Array (
            uint8_t block,
            uint8_t address,
            uint8_t * arr,
            uint8_t size )
```

Writes byte array into the EEPROM. Bytes are written at contiguous locations

Parameters

block	Block number to write into (1-4)
address	Address inside the block to start writing from (0-255)
arr	The byte array to write into the EEPROM
size	The number of bytes to write

5.9.3.6 EEPROM_Write_BYTE()

```
uint8_t EEPROM_Write_BYTE (
            uint8_t block,
            uint8_t address,
            uint8_t data )
```

Writes single byte into the EEPROM

Parameters

block	Block number to write into (1-4)
address	Address inside the block to write into (0-255)
data	The single byte to write at this location

5.9.3.7 I2C1_receive_byte()

```
uint8_t I2C1_receive_byte ( )
```

Recieves byte over I2C1

Returns

Single byte read over I2C1

5.9.3.8 I2C1_transmit_byte()

```
void I2C1_transmit_byte (
            uint8_t byte )
```

Transmit single byte over I2C

Parameters

byte	Single byte to transmit

5.9.3.9 I2C1_transmit_restart_bit()

```
void I2C1_transmit_restart_bit ( )
```

Transmit single bit for restarting I2C communication

5.9.3.10 I2C1_transmit_start_bit()

```
void I2C1_transmit_start_bit ( )
```

Transmit single bit for starting I2C communication

5.9.3.11 I2C1_transmit_stop_bit()

```
void I2C1_transmit_stop_bit ( )
```

Transmit single bit for stoping I2C communication

5.9.3.12 I2C1_wait_while_busy()

```
void I2C1_wait_while_busy ( )
```

Waits while I2C1 is busy reading or writing

5.9.3.13 open_I2C1()

```
void open_I2C1 ( )
```

Opens I2C1 peripheral

5.9.3.14 setup_EEPROM()

```
void setup_EEPROM ( )
```

Sets up EEPROM using I2C1

5.10 src/SSN_API/Drivers/MESSAGES/messages.c File Reference

```
#include "messages.h"
```

Functions

- void get_bytes_from_uint16 (uint16_t word, char *bytes)
- void get_bytes_from_uint32 (uint32_t long_word, char *bytes)
- uint8_t is_Valid_MAC (uint8_t *mac_address)
- uint8_t is_Valid_CONFIG (uint8_t *config_array)
- uint8_t construct_get_mac_message (uint8_t *message_array, uint8_t *node_id)
- uint8_t construct_get_configuration_message (uint8_t *message_array, uint8_t *node_id)
- uint8_t construct_ack_configuration_message (uint8_t *message_array, uint8_t *node_id, uint8_t *received_configs)
- uint8_t construct_get_timeofday_message (uint8_t *message_array, uint8_t *node_id)
- uint8_t construct_status_update_message (uint8_t *message_array, uint8_t *node_id, uint8_t *temperature_bytes, uint8_t *relative_humidity_bytes, uint8_t *Machine_load_currents, uint8_t *Machine_load_percentages, uint8_t *Machine_status, uint32_t *Machine_status_duration, pseudo_clock *Machine_status_timestamp, uint32_t node_uptime_in_seconds, uint8_t abnormal_activity)
- uint8_t decipher_received_message (uint8_t *message, uint8_t *params)

5.10.1 Function Documentation

5.10.1.1 construct_ack_configuration_message()

```
uint8_t construct_ack_configuration_message (
            uint8_t * message_array,
            uint8_t * node_id,
            uint8_t * received_configs )
```

Constructs Acknowledge Configuration message

Parameters

message_array	Byte array to keep
node_id	Identity of SSN, last two bytes of SSN MAC address
received_configs	Byte array of current sensor configurations received from Server

Returns

Message size in bytes

5.10.1.2 construct_get_configuration_message()

```
uint8_t construct_get_configuration_message (
            uint8_t * message_array,
            uint8_t * node_id )
```

Constructs GET_CONFIG request message to retrieve current sensor configurations

Parameters

message_array	Byte array to keep
node_id	Identity of SSN, last two bytes of SSN MAC address

Returns

Message size in bytes

5.10.1.3 construct_get_mac_message()

```
uint8_t construct_get_mac_message (
            uint8_t * message_array,
            uint8_t * node_id )
```

Constructs GET_MAC request message

Parameters

message_array	Byte array to keep
node_id	Identity of SSN, last two bytes of SSN MAC address

Returns

Message size in bytes

5.10.1.4 construct_get_timeofday_message()

```
uint8_t construct_get_timeofday_message (
            uint8_t * message_array,
            uint8_t * node_id )
```

Constructs GET_TimeOfDay request message

Parameters

message_array	Byte array to keep
node_id	Identity of SSN, last two bytes of SSN MAC address

Returns

Message size in bytes

5.10.1.5 construct_status_update_message()

```
uint8_t construct_status_update_message (
            uint8_t * message_array,
            uint8_t * node_id,
            uint8_t * temperature_bytes,
            uint8_t * relative_humidity_bytes,
            uint8_t * Machine_load_currents,
            uint8_t * Machine_load_percentages,
            uint8_t * Machine_status,
            uint32_t * Machine_status_duration,
            pseudo_clock * Machine_status_timestamp,
            uint32_t node_uptime_in_seconds,
            uint8_t abnormal_activity )
```

Constructs periodic status update message for Server

Parameters

message_array	Byte array for keeping the constructed message
node_id	Node identity in two bytes, last two bytes of current MAC address
temperature_bytes	Byte array saving the temperature reading high and low bytes
relative_humidity_bytes	Byte array saving the relative humidity high and low bytes
Machine_load_currents	A byte array of machine load currents calculated by SSN
Machine_load_percentages	A byte array of machine load percentages calculated by SSN
Machine_status	A byte array of machine status (ON/OFF/IDLE) calculated by SSN
Machine_status_duration	An array of machine status duration indicating for how long the machines have been in current state
Machine_status_timestamp	An array of machine status timestamp indicating since when the machines have been in current state
node_uptime_in_seconds	A four byte integer containing the number of seconds indicating for how long the SSN has been awake
abnormal_activity	A single byte indicating NORMAL or ABNORMAL ambient condition based on temperature and humidity readings

Returns

Message size in bytes

5.10.1.6 decipher_received_message()

```
uint8_t decipher_received_message (
            uint8_t * message,
            uint8_t * params )
```

Deciphers the received message and returns whatever data was received with it

Parameters

message	Message byte array received from Server
params	The parameters array to save the received data into in specific order

Returns

Message ID of received message

5.10.1.7 get_bytes_from_uint16()

```
void get_bytes_from_uint16 (
            uint16_t word,
            char * bytes )
```

5.10.1.8 get_bytes_from_uint32()

```
void get_bytes_from_uint32 (
            uint32_t long_word,
            char * bytes )
```

5.10.1.9 is_Valid_CONFIG()

```
uint8_t is_Valid_CONFIG (
            uint8_t * config_array )
```

Checks the validity of current sensor configurations

Parameters

config_array	Byte array of current configurations

Returns

1 if valid; 0 otherwise

5.10.1.10 is_Valid_MAC()

```
uint8_t is_Valid_MAC (
            uint8_t * mac_address )
```

Checks the validity of MAC address whether it is custom MAC or default MAC address of SSN

Parameters

mac_address	Six bytes of current MAC address of SSN

Returns

1 if valid; 0 otherwise

5.11 src/SSN_API/Drivers/MESSAGES/messages.h File Reference

```
#include <xc.h>
#include <p32xxxx.h>
#include <plib.h>
#include <stdint.h>
#include <stdbool.h>
#include "../../global.h"
```

Macros

- #define _SUPPRESS_PLIB_WARNING
- #define _DISABLE_OPENADC10_CONFIGPORT_WARNING
- #define max_send_message_size 110
- #define max_recv_message_size 15
- #define GET_MAC_MESSAGE_ID 1
- #define SET_MAC_MESSAGE_ID 2
- #define GET_TIMEOFDAY_MESSAGE_ID 3
- #define SET_TIMEOFDAY_MESSAGE_ID 4
- #define GET_CONFIG_MESSAGE_ID 5
- #define SET_CONFIG_MESSAGE_ID 6
- #define ACK_CONFIG_MESSAGE_ID 7
- #define STATUS_UPDATE_MESSAGE_ID 8
- #define RESET_MACHINE_TIME_MESSAGE_ID 9
- #define DEBUG_EEPROM_CLEAR_MESSAGE_ID 10
- #define DEBUG_RESET_SSN_MESSAGE_ID 11
- #define GET_MAC_MESSAGE_Size 3
- #define SET_MAC_MESSAGE_Size 7

- #define GET_TIMEOFDAY_MESSAGE_Size 3
- #define SET_TIMEOFDAY_MESSAGE_Size 7
- #define GET_CONFIG_MESSAGE_Size 3
- #define SET_CONFIG_MESSAGE_Size 14
- #define ACK_CONFIG_MESSAGE_Size 16
- #define STATUS_UPDATE_MESSAGE_Size 64
- #define RESET_MACHINE_TIME_MESSAGE_Size 4

Functions

- uint8_t is_Valid_MAC (uint8_t *mac_address)
- uint8_t is_Valid_CONFIG (uint8_t *config_array)
- uint8_t construct_get_mac_message (uint8_t *message_array, uint8_t *node_id)
- uint8_t construct_get_timeofday_message (uint8_t *message_array, uint8_t *node_id)
- uint8_t construct_get_configuration_message (uint8_t *message_array, uint8_t *node_id)
- uint8_t construct_ack_configuration_message (uint8_t *message_array, uint8_t *node_id, uint8_t *received_configs)
- uint8_t construct_status_update_message (uint8_t *message_array, uint8_t *node_id, uint8_t *temperature_bytes, uint8_t *relative_humidity_bytes, uint8_t *Machine_load_currents, uint8_t *Machine_load_percentages, uint8_t *Machine_status, uint32_t *Machine_status_duration, pseudo_clock *Machine_status_timestamp, uint32_t node_uptime_in_seconds, uint8_t abnormal_activity)
- uint8_t decipher_received_message (uint8_t *message, uint8_t *params)

5.11.1 Macro Definition Documentation

5.11.1.1 _DISABLE_OPENADC10_CONFIGPORT_WARNING

```
#define _DISABLE_OPENADC10_CONFIGPORT_WARNING
```

5.11.1.2 _SUPPRESS_PLIB_WARNING

```
#define _SUPPRESS_PLIB_WARNING
```

5.11.1.3 ACK_CONFIG_MESSAGE_ID

```
#define ACK_CONFIG_MESSAGE_ID 7
```

Message id for acknowledge current sensor configurations message to Server

5.11.1.4 ACK_CONFIG_MESSAGE_Size

```
#define ACK_CONFIG_MESSAGE_Size 16
```

5.11.1.5 DEBUG_EEPROM_CLEAR_MESSAGE_ID

```
#define DEBUG_EEPROM_CLEAR_MESSAGE_ID 10
```

Message id for requesting SSN EEPROM Clear received from Server

5.11.1.6 DEBUG_RESET_SSN_MESSAGE_ID

```
#define DEBUG_RESET_SSN_MESSAGE_ID 11
```

Message id for requesting SSN Reset received from Server

5.11.1.7 GET_CONFIG_MESSAGE_ID

```
#define GET_CONFIG_MESSAGE_ID 5
```

Message id for requesting current sensor configurations from Server

5.11.1.8 GET_CONFIG_MESSAGE_Size

```
#define GET_CONFIG_MESSAGE_Size 3
```

5.11.1.9 GET_MAC_MESSAGE_ID

```
#define GET_MAC_MESSAGE_ID 1
```

Message id for requesting MAC address from Server

5.11.1.10 GET_MAC_MESSAGE_Size

```
#define GET_MAC_MESSAGE_Size 3
```

5.11.1.11 GET_TIMEOFDAY_MESSAGE_ID

```
#define GET_TIMEOFDAY_MESSAGE_ID 3
```

Message id for requesting time of day from Server

5.11.1.12 GET_TIMEOFDAY_MESSAGE_Size

```
#define GET_TIMEOFDAY_MESSAGE_Size 3
```

5.11.1.13 max_recv_message_size

```
#define max_recv_message_size 15
```

Maximum Message size in bytes to receive over the network

5.11.1.14 max_send_message_size

```
#define max_send_message_size 110
```

Maximum Message size in bytes to send over the network

5.11.1.15 RESET_MACHINE_TIME_MESSAGE_ID

```
#define RESET_MACHINE_TIME_MESSAGE_ID 9
```

Message id for machine reset accumulated time received from Server

5.11.1.16 RESET_MACHINE_TIME_MESSAGE_Size

```
#define RESET_MACHINE_TIME_MESSAGE_Size 4
```

5.11.1.17 SET_CONFIG_MESSAGE_ID

```
#define SET_CONFIG_MESSAGE_ID 6
```

Message id for set current sensor configurations message received from Server

5.11.1.18 SET_CONFIG_MESSAGE_Size

```
#define SET_CONFIG_MESSAGE_Size 14
```

5.11.1.19 SET_MAC_MESSAGE_ID

```
#define SET_MAC_MESSAGE_ID 2
```

Message id for set MAC address message received from Server

5.11.1.20 SET_MAC_MESSAGE_Size

```
#define SET_MAC_MESSAGE_Size 7
```

5.11.1.21 SET_TIMEOFDAY_MESSAGE_ID

```
#define SET_TIMEOFDAY_MESSAGE_ID 4
```

Message id for set time of day message received from Server

5.11.1.22 SET_TIMEOFDAY_MESSAGE_Size

```
#define SET_TIMEOFDAY_MESSAGE_Size 7
```

5.11.1.23 STATUS_UPDATE_MESSAGE_ID

```
#define STATUS_UPDATE_MESSAGE_ID 8
```

Message id for status update message to send to Server

5.11.1.24 STATUS_UPDATE_MESSAGE_Size

```
#define STATUS_UPDATE_MESSAGE_Size 64
```

5.11.2 Function Documentation

5.11.2.1 construct_ack_configuration_message()

```
uint8_t construct_ack_configuration_message (
            uint8_t * message_array,
            uint8_t * node_id,
            uint8_t * received_configs )
```

Constructs Acknowledge Configuration message

Parameters

message_array	Byte array to keep
node_id	Identity of SSN, last two bytes of SSN MAC address
received_configs	Byte array of current sensor configurations received from Server

Returns

Message size in bytes

5.11.2.2 construct_get_configuration_message()

```
uint8_t construct_get_configuration_message (
            uint8_t * message_array,
            uint8_t * node_id )
```

Constructs GET_CONFIG request message to retrieve current sensor configurations

Parameters

message_array	Byte array to keep
node_id	Identity of SSN, last two bytes of SSN MAC address

Returns

Message size in bytes

5.11.2.3 construct_get_mac_message()

```
uint8_t construct_get_mac_message (
            uint8_t * message_array,
            uint8_t * node_id )
```

Constructs GET_MAC request message

Parameters

message_array	Byte array to keep
node_id	Identity of SSN, last two bytes of SSN MAC address

Returns

Message size in bytes

5.11.2.4 construct_get_timeofday_message()

```
uint8_t construct_get_timeofday_message (
            uint8_t * message_array,
            uint8_t * node_id )
```

Constructs GET_TimeOfDay request message

Parameters

message_array	Byte array to keep
node_id	Identity of SSN, last two bytes of SSN MAC address

Returns

Message size in bytes

5.11.2.5 construct_status_update_message()

```
uint8_t construct_status_update_message (
            uint8_t * message_array,
            uint8_t * node_id,
            uint8_t * temperature_bytes,
            uint8_t * relative_humidity_bytes,
            uint8_t * Machine_load_currents,
            uint8_t * Machine_load_percentages,
            uint8_t * Machine_status,
            uint32_t * Machine_status_duration,
            pseudo_clock * Machine_status_timestamp,
            uint32_t node_uptime_in_seconds,
            uint8_t abnormal_activity )
```

Constructs periodic status update message for Server

Parameters

message_array	Byte array for keeping the constructed message
node_id	Node identity in two bytes, last two bytes of current MAC address
temperature_bytes	Byte array saving the temperature reading high and low bytes
relative_humidity_bytes	Byte array saving the relative humidity high and low bytes
Machine_load_currents	A byte array of machine load currents calculated by SSN
Machine_load_percentages	A byte array of machine load percentages calculated by SSN
Machine_status	A byte array of machine status (ON/OFF/IDLE) calculated by SSN
Machine_status_duration	An array of machine status duration indicating for how long the machines have been in current state
Machine_status_timestamp	An array of machine status timestamp indicating since when the machines have been in current state
node_uptime_in_seconds	A four byte integer containing the number of seconds indicating for how long the SSN has been awake
abnormal_activity	A single byte indicating NORMAL or ABNORMAL ambient condition based on temperature and humidity readings

Returns

Message size in bytes

5.11.2.6 decipher_received_message()

```
uint8_t decipher_received_message (
            uint8_t * message,
            uint8_t * params )
```

Deciphers the received message and returns whatever data was received with it

Parameters

message	Message byte array received from Server
params	The parameters array to save the received data into in specific order

Returns

Message ID of received message

5.11.2.7 is_Valid_CONFIG()

```
uint8_t is_Valid_CONFIG (
            uint8_t * config_array )
```

Checks the validity of current sensor configurations

Parameters

config_array	Byte array of current configurations

Returns

1 if valid; 0 otherwise

5.11.2.8 is_Valid_MAC()

```
uint8_t is_Valid_MAC (
            uint8_t * mac_address )
```

Checks the validity of MAC address whether it is custom MAC or default MAC address of SSN

Parameters

mac_address	Six bytes of current MAC address of SSN

Returns

1 if valid; 0 otherwise

5.12 src/SSN_API/Drivers/NETWORK/network.c File Reference

```
#include "network.h"
```

Functions

- void WIZ5500_Reset ()
- void open_SPI2 ()
- void setup_Ethernet ()
- unsigned int SPI2_send (unsigned int data)
- void WIZ5500_select (void)
- void WIZ5500_deselect (void)
- void WIZ5500_write_byte (uint8_t wb)
- uint8_t WIZ5500_read_byte ()
- void WIZ5500_write_array (uint8_t *addrBuf, uint8_t *pBuf, uint16_t len)
- void WIZ5500_read_array (uint8_t *addrBuf, uint8_t *pBuf, uint16_t len)
- void WIZ5500_network_initiate (void)
- void WIZ5500_IP_assigned_callback (void)
- void WIZ5500_IP_conflict_callback (void)
- uint8_t Ethernet_get_physical_link_status ()
- void Ethernet_Assign_MAC (uint8_t *this_mac)
- void setup_TIMER2_with_interrupt (float delay_time)
- void stop_TIMER2_with_interrupt ()
- void __ISR (_TIMER_2_VECTOR, IPL4SOFT)
- void Ethernet_get_IP_from_DHCP ()
- int32_t Send_Message_Over_UDP (uint8_t socket_number, uint8_t *message, uint8_t message_byte_length, char *destination_ip, uint16_t destination_port)
- uint8_t is_Message_Received_Over_UDP (uint8_t socket_number)
- uint8_t Recv_Message_Over_UDP (uint8_t socket_number, char *message, uint8_t message_byte_length, char *destination_ip, uint16_t destination_port)

5.12.1 Function Documentation

5.12.1.1 __ISR()

```
void __ISR (
            _TIMER_2_VECTOR ,
            IPL4SOFT  )
```

5.12.1.2 Ethernet_Assign_MAC()

```
void Ethernet_Assign_MAC (
            uint8_t * this_mac )
```

Assigns a MAC address to W5500 chip

Parameters

this_mac	The byte array containing the MAC address

5.12.1.3 Ethernet_get_IP_from_DHCP()

```
void Ethernet_get_IP_from_DHCP ( )
```

Gets an IP from DHCP; does not return until an IP is successfully retrieved

5.12.1.4 Ethernet_get_physical_link_status()

```
uint8_t Ethernet_get_physical_link_status ( )
```

Gets physical link status from the W5500 chip whether a network cable is connected to network or not

Returns

PHY_LINK_ON if connection is available; **PHY_LINK_OFF** otherwise

5.12.1.5 is_Message_Received_Over_UDP()

```
uint8_t is_Message_Received_Over_UDP (
            uint8_t socket_number )
```

Tells whether a message was received or not and how many bytes are there in the buffer

Parameters

socket_number	UDP socket number

Returns

Number of bytes in receive buffer of W5500. May contain more than one message

5.12.1.6 open_SPI2()

```
void open_SPI2 ( )
```

Opens SPI2 for communication

5.12.1.7 Recv_Message_Over_UDP()

```
uint8_t Recv_Message_Over_UDP (
            uint8_t socket_number,
            char * message,
            uint8_t message_byte_length,
            char * destination_ip,
            uint16_t destination_port )
```

Receives a message over UDP

Parameters

socket_number	UDP socket number of SSN
message	The byte array in which the received message will be written into
message_byte_length	Maximum byte count that may be received in a single message
destination_ip	IP of destination server
destination_port	Port of destination server

Returns

Number of bytes in received message

5.12.1.8 Send_Message_Over_UDP()

```
int32_t Send_Message_Over_UDP (
            uint8_t socket_number,
            uint8_t * message,
            uint8_t message_byte_length,
            char * destination_ip,
            uint16_t destination_port )
```

Sends a message over UDP

Parameters

socket_number	UDP socket number
message	Byte array containing the message to send
message_byte_length	Number of bytes of the message to send
desination_ip	IP of destination server
destination_port	Port of destination server

Returns

5.12.1.9 setup_Ethernet()

```
void setup_Ethernet ( )
```

Sets up Ethernet for communication

5.12.1.10 setup_TIMER2_with_interrupt()

```
void setup_TIMER2_with_interrupt (
            float delay_time )
```

Sets up a timer interrupt required to make DHCP requests

Parameters

delay_time	Period of interrupt in seconds

5.12.1.11 SPI2_send()

```
unsigned int SPI2_send (
            unsigned int data )
```

Sends and Receives a single byte over SPI interface

Parameters

data	Single byte to send over SPI

Returns

Single byte received over SPI

5.12.1.12 stop_TIMER2_with_interrupt()

```
void stop_TIMER2_with_interrupt ( )
```

Stops the timer interrupt

5.12.1.13 WIZ5500_deselect()

```
void WIZ5500_deselect (
            void  )
```

Deselects the W5500 chip for communication

5.12.1.14 WIZ5500_IP_assigned_callback()

```
void WIZ5500_IP_assigned_callback (
            void  )
```

Callback function for when IP is received via DHCP

5.12.1.15 WIZ5500_IP_conflict_callback()

```
void WIZ5500_IP_conflict_callback (
            void  )
```

Callback function for when IP conflict occurs

5.12.1.16 WIZ5500_network_initiate()

```
void WIZ5500_network_initiate (
            void  )
```

Initializes the W5500 chip from network communication

5.12.1.17 WIZ5500_read_array()

```
void WIZ5500_read_array (
            uint8_t * addrBuf,
            uint8_t * pBuf,
            uint16_t len )
```

Reads a byte array from W5500 chip

Parameters

addrBuf	Byte array containing the addresses to read from
pBuf	Byte array to write the read values into
len	The number of values to read from W5500

5.12.1.18 WIZ5500_read_byte()

```
uint8_t WIZ5500_read_byte ( )
```

Reads a single byte from W5500 chip

Returns

Single byte received from W5500 chip

5.12.1.19 WIZ5500_Reset()

```
void WIZ5500_Reset ( )
```

Resets W5500 Ethernet offload chip

5.12.1.20 WIZ5500_select()

```
void WIZ5500_select (
            void  )
```

Selects the W5500 chip for communication

5.12.1.21 WIZ5500_write_array()

```
void WIZ5500_write_array (
            uint8_t * addrBuf,
            uint8_t * pBuf,
            uint16_t len )
```

Writes a byte array to W5500 chip

Parameters

addrBuf	Byte array containing the addresses to write at
pBuf	Byte array of values to write at those locations
len	The number of values to write at W5500

5.12.1.22 WIZ5500_write_byte()

```
void WIZ5500_write_byte (
            uint8_t wb )
```

Writes a single byte to W5500 chip

Parameters

wb	Byte to write

5.13 src/SSN_API/Drivers/NETWORK/network.h File Reference

```
#include "../../global.h"
#include <plib.h>
#include <stdio.h>
```

```
#include <string.h>
#include "Ethernet/socket.h"
#include "Internet/DHCP/dhcp.h"
```

Macros

- #define _SUPPRESS_PLIB_WARNING
- #define _DISABLE_OPENADC10_CONFIGPORT_WARNING
- #define WIZ5500_R_COMMON_RTR 0x001A0100
- #define WIZ5500_W_COMMON_RTR 0x001A05F1
- #define WIZ5500_R_COMMON_RCR 0x001B0100
- #define WIZ5500_W_COMMON_RCR 0x001B05F1
- #define setPR2(seconds) (seconds ∗ PERIPH_CLK / 64)
- #define _MAIN_DEBUG_
- #define _DHCP_DEBUG_
- #define SOCK_DHCP 0
- #define MY_MAX_DHCP_RETRY 3
- #define DATA_BUF_SIZE 2048

Functions

- void WIZ5500_Reset ()
- void open_SPI2 ()
- void setup_Ethernet ()
- unsigned int SPI2_send (unsigned int data)
- void WIZ5500_select (void)
- void WIZ5500_deselect (void)
- void WIZ5500_write_byte (uint8_t wb)
- uint8_t WIZ5500_read_byte ()
- void WIZ5500_write_array (uint8_t ∗addrBuf, uint8_t ∗pBuf, uint16_t len)
- void WIZ5500_read_array (uint8_t ∗addrBuf, uint8_t ∗pBuf, uint16_t len)
- void WIZ5500_network_initiate (void)
- void WIZ5500_IP_assigned_callback (void)
- void WIZ5500_IP_conflict_callback (void)
- void setup_TIMER2_with_interrupt (float delay_time)
- void stop_TIMER2_with_interrupt ()
- uint8_t Ethernet_get_physical_link_status ()
- void Ethernet_Assign_MAC (uint8_t ∗this_mac)
- void Ethernet_get_IP_from_DHCP ()
- int32_t Send_Message_Over_UDP (uint8_t socket_number, uint8_t ∗message, uint8_t message_byte_length, char ∗destination_ip, uint16_t destination_port)
- uint8_t is_Message_Received_Over_UDP (uint8_t socket_number)
- uint8_t Recv_Message_Over_UDP (uint8_t socket_number, char ∗message, uint8_t message_byte_length, char ∗destination_ip, uint16_t destination_port)

Variables

- uint8_t gDATABUF [DATA_BUF_SIZE]
- volatile uint32_t msTicks

  Default Network Inforamtion.
- uint32_t prevTick
- wiz_NetInfo WIZ5500_network_information

5.13.1 Macro Definition Documentation

5.13.1.1 _DHCP_DEBUG_

```
#define _DHCP_DEBUG_
```

5.13.1.2 _DISABLE_OPENADC10_CONFIGPORT_WARNING

```
#define _DISABLE_OPENADC10_CONFIGPORT_WARNING
```

5.13.1.3 _MAIN_DEBUG_

```
#define _MAIN_DEBUG_
```

5.13.1.4 _SUPPRESS_PLIB_WARNING

```
#define _SUPPRESS_PLIB_WARNING
```

5.13.1.5 DATA_BUF_SIZE

```
#define DATA_BUF_SIZE 2048
```

5.13.1.6 MY_MAX_DHCP_RETRY

```
#define MY_MAX_DHCP_RETRY 3
```

5.13.1.7 setPR2

```
#define setPR2(
             seconds ) (seconds * PERIPH_CLK / 64)
```

5.13.1.8 SOCK_DHCP

```
#define SOCK_DHCP 0
```

5.13.1.9 WIZ5500_R_COMMON_RCR

```
#define WIZ5500_R_COMMON_RCR 0x001B0100
```

5.13.1.10 WIZ5500_R_COMMON_RTR

```
#define WIZ5500_R_COMMON_RTR 0x001A0100
```

5.13.1.11 WIZ5500_W_COMMON_RCR

```
#define WIZ5500_W_COMMON_RCR 0x001B05F1
```

5.13.1.12 WIZ5500_W_COMMON_RTR

```
#define WIZ5500_W_COMMON_RTR 0x001A05F1
```

5.13.2 Function Documentation

5.13.2.1 Ethernet_Assign_MAC()

```
void Ethernet_Assign_MAC (
            uint8_t * this_mac )
```

Assigns a MAC address to W5500 chip

Parameters

this_mac	The byte array containing the MAC address

5.13.2.2 Ethernet_get_IP_from_DHCP()

```
void Ethernet_get_IP_from_DHCP ( )
```

Gets an IP from DHCP; does not return until an IP is successfully retrieved

5.13.2.3 Ethernet_get_physical_link_status()

```
uint8_t Ethernet_get_physical_link_status ( )
```

Gets physical link status from the W5500 chip whether a network cable is connected to network or not

Returns

PHY_LINK_ON if connection is available; **PHY_LINK_OFF** otherwise

5.13.2.4 is_Message_Received_Over_UDP()

```
uint8_t is_Message_Received_Over_UDP (
            uint8_t socket_number )
```

Tells whether a message was received or not and how many bytes are there in the buffer

Parameters

socket_number	UDP socket number

Returns

Number of bytes in receive buffer of W5500. May contain more than one message

5.13.2.5 open_SPI2()

```
void open_SPI2 ( )
```

Opens SPI2 for communication

5.13.2.6 Recv_Message_Over_UDP()

```
uint8_t Recv_Message_Over_UDP (
            uint8_t socket_number,
            char * message,
            uint8_t message_byte_length,
            char * destination_ip,
            uint16_t destination_port )
```

Receives a message over UDP

Parameters

socket_number	UDP socket number of SSN
message	The byte array in which the received message will be written into
message_byte_length	Maximum byte count that may be received in a single message
destination_ip	IP of destination server
destination_port	Port of destination server

Returns

Number of bytes in received message

5.13.2.7 Send_Message_Over_UDP()

```
int32_t Send_Message_Over_UDP (
            uint8_t socket_number,
            uint8_t * message,
            uint8_t message_byte_length,
            char * destination_ip,
            uint16_t destination_port )
```

Sends a message over UDP

Parameters

socket_number	UDP socket number
message	Byte array containing the message to send
message_byte_length	Number of bytes of the message to send
desination_ip	IP of destination server
destination_port	Port of destination server

Returns

5.13.2.8 setup_Ethernet()

```
void setup_Ethernet ( )
```

Sets up Ethernet for communication

5.13.2.9 setup_TIMER2_with_interrupt()

```
void setup_TIMER2_with_interrupt (
            float delay_time )
```

Sets up a timer interrupt required to make DHCP requests

Parameters

delay_time	Period of interrupt in seconds

5.13.2.10 SPI2_send()

```
unsigned int SPI2_send (
            unsigned int data )
```

Sends and Receives a single byte over SPI interface

Parameters

data	Single byte to send over SPI

Returns

Single byte received over SPI

5.13.2.11 stop_TIMER2_with_interrupt()

```
void stop_TIMER2_with_interrupt ( )
```

Stops the timer interrupt

5.13.2.12 WIZ5500_deselect()

```
void WIZ5500_deselect (
            void  )
```

Deselects the W5500 chip for communication

5.13.2.13 WIZ5500_IP_assigned_callback()

```
void WIZ5500_IP_assigned_callback (
            void  )
```

Callback function for when IP is received via DHCP

5.13.2.14 WIZ5500_IP_conflict_callback()

```
void WIZ5500_IP_conflict_callback (
            void  )
```

Callback function for when IP conflict occurs

5.13.2.15 WIZ5500_network_initiate()

```
void WIZ5500_network_initiate (
            void  )
```

Initializes the W5500 chip from network communication

5.13.2.16 WIZ5500_read_array()

```
void WIZ5500_read_array (
            uint8_t * addrBuf,
            uint8_t * pBuf,
            uint16_t len )
```

Reads a byte array from W5500 chip

Parameters

addrBuf	Byte array containing the addresses to read from
pBuf	Byte array to write the read values into
len	The number of values to read from W5500

5.13.2.17 WIZ5500_read_byte()

```
uint8_t WIZ5500_read_byte ( )
```

Reads a single byte from W5500 chip

Returns

Single byte received from W5500 chip

5.13.2.18 WIZ5500_Reset()

```
void WIZ5500_Reset ( )
```

Resets W5500 Ethernet offload chip

5.13.2.19 WIZ5500_select()

```
void WIZ5500_select (
            void  )
```

Selects the W5500 chip for communication

5.13.2.20 WIZ5500_write_array()

```
void WIZ5500_write_array (
            uint8_t * addrBuf,
            uint8_t * pBuf,
            uint16_t len )
```

Writes a byte array to W5500 chip

Parameters

addrBuf	Byte array containing the addresses to write at
pBuf	Byte array of values to write at those locations
len	The number of values to write at W5500

5.13.2.21 WIZ5500_write_byte()

```
void WIZ5500_write_byte (
            uint8_t wb )
```

Writes a single byte to W5500 chip

Parameters

wb	Byte to write

5.13.3 Variable Documentation

5.13.3.1 gDATABUF

```
uint8_t gDATABUF[DATA_BUF_SIZE]
```

5.13.3.2 msTicks

```
volatile uint32_t msTicks
```

Default Network Inforamtion.

5.13.3.3 prevTick

```
uint32_t prevTick
```

5.13.3.4 WIZ5500_network_information

```
wiz_NetInfo WIZ5500_network_information
```

5.14 src/SSN_API/Drivers/PSEUDO_RTCC/pseudo_rtcc.c File Reference

```
#include "pseudo_rtcc.h"
```

Functions

- void setup_Global_Clock_And_SSN_Half_Second_Heartbeat (uint32_t PERIPH_CLOCK)
- void stop_Global_Clock ()
- void set_ssn_time (uint8_t *this_time)
- void increment_pseudo_clock_time (pseudo_clock *this_clock)
- void increment_this_clock_time (uint8_t *this_clock)

Variables

- uint8_t Days_in_a_Month [12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}

5.14.1 Function Documentation

5.14.1.1 increment_pseudo_clock_time()

```
void increment_pseudo_clock_time (
            pseudo_clock * this_clock )
```

Increments given clock time by one tick or one second

Parameters

this_clock	The pseudo_clock variable clock to increment

5.14.1.2 increment_this_clock_time()

```
void increment_this_clock_time (
            uint8_t * this_clock )
```

Increments given clock time by one tick or one second

Parameters

this_clock	The byte array representing clock to increment

5.14.1.3 set_ssn_time()

```
void set_ssn_time (
            uint8_t * this_time )
```

Sets the SSN global time to give time

Parameters

this_time	Byte array containing hours, minutes, seconds, day, month and year of current time

5.14.1.4 setup_Global_Clock_And_SSN_Half_Second_Heartbeat()

```
void setup_Global_Clock_And_SSN_Half_Second_Heartbeat (
            uint32_t PERIPH_CLOCK )
```

Sets up the global clock and half-second interrupt for SSN

Parameters

PERIPH_CLOCK	Peripheral clock of SSN

5.14.1.5 stop_Global_Clock()

```
void stop_Global_Clock ( )
```

Stops the global SSN clock

5.14.2 Variable Documentation

5.14.2.1 Days_in_a_Month

```
uint8_t Days_in_a_Month[12] = {31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31}
```

5.15 src/SSN_API/Drivers/PSEUDO_RTCC/pseudo_rtcc.h File Reference

```
#include <xc.h>
#include <p32xxxx.h>
#include <plib.h>
#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>
```

Classes

- struct pseudo_clock

Macros

- #define _SUPPRESS_PLIB_WARNING
- #define _DISABLE_OPENADC10_CONFIGPORT_WARNING

Functions

- void setup_Global_Clock_And_SSN_Half_Second_Heartbeat (uint32_t PERIPH_CLOCK)
- void stop_Global_Clock ()
- void set_ssn_time (uint8_t *this_time)
- void increment_pseudo_clock_time (pseudo_clock *this_clock)
- void increment_this_clock_time (uint8_t *this_clock)

Variables

- pseudo_clock ssn_clock
- uint32_t ssn_uptime_in_seconds

5.15.1 Macro Definition Documentation

5.15.1.1 _DISABLE_OPENADC10_CONFIGPORT_WARNING

```
#define _DISABLE_OPENADC10_CONFIGPORT_WARNING
```

5.15.1.2 _SUPPRESS_PLIB_WARNING

```
#define _SUPPRESS_PLIB_WARNING
```

5.15.2 Function Documentation

5.15.2.1 increment_pseudo_clock_time()

```
void increment_pseudo_clock_time (
            pseudo_clock * this_clock )
```

Increments given clock time by one tick or one second

Parameters

this_clock	The pseudo_clock variable clock to increment

5.15.2.2 increment_this_clock_time()

```
void increment_this_clock_time (
            uint8_t * this_clock )
```

Increments given clock time by one tick or one second

Parameters

this_clock	The byte array representing clock to increment

5.15.2.3 set_ssn_time()

```
void set_ssn_time (
            uint8_t * this_time )
```

Sets the SSN global time to give time

Parameters

this_time	Byte array containing hours, minutes, seconds, day, month and year of current time

5.15.2.4 setup_Global_Clock_And_SSN_Half_Second_Heartbeat()

```
void setup_Global_Clock_And_SSN_Half_Second_Heartbeat (
            uint32_t PERIPH_CLOCK )
```

Sets up the global clock and half-second interrupt for SSN

Parameters

PERIPH_CLOCK	Peripheral clock of SSN

5.15.2.5 stop_Global_Clock()

```
void stop_Global_Clock ( )
```

Stops the global SSN clock

5.15.3 Variable Documentation

5.15.3.1 ssn_clock

```
pseudo_clock ssn_clock
```

This is our global SSN clock variable

5.15.3.2 ssn_uptime_in_seconds

```
uint32_t ssn_uptime_in_seconds
```

This is our global SSN clock in number of seconds, for how many seconds has the SSN been awake?

5.16 src/SSN_API/Drivers/TEMPERATURE_SENSOR/temperature_sensor.c File Reference

```
#include "temperature_sensor.h"
```

Functions

- void open_I2C2 ()
- void setup_Temperature_Humidity_Sensor ()
- void I2C2_wait_while_busy ()
- void I2C2_transmit_start_bit ()
- void I2C2_transmit_stop_bit ()
- void I2C2_transmit_restart_bit ()
- void I2C2_transmit_byte (uint8_t byte)
- uint8_t I2C2_receive_byte ()
- void I2C2_ack (void)
- void AM2320_I2C2_Read_Temp_and_Humidity ()
- uint16_t convert_bytes_to_word (int8_t high_byte, int8_t low_byte)
- unsigned short crc16 (unsigned char *ptr, unsigned char len)
- uint8_t CRC_check ()
- uint8_t sample_Temperature_Humidity (uint16_t *temperature, uint16_t *relative_humidity)
- uint8_t sample_Temperature_Humidity_bytes (uint8_t *temperature_bytes, uint8_t *relative_humidity_bytes)
- uint8_t ambient_condition_status ()

5.16.1 Function Documentation

5.16.1.1 AM2320_I2C2_Read_Temp_and_Humidity()

```
void AM2320_I2C2_Read_Temp_and_Humidity ( )
```

Reads the temperature and humidity bytes from AM2320 sensor using I2C. Read values are written into recv_data array

5.16.1.2 ambient_condition_status()

```
uint8_t ambient_condition_status ( )
```

Gets ambient condition status

Returns

NORMAL_AMBIENT_CONDITION if normal; **ABNORMAL_AMBIENT_CONDITION** otherwise.

5.16.1.3 convert_bytes_to_word()

```
uint16_t convert_bytes_to_word (
            int8_t high_byte,
            int8_t low_byte )
```

Converts bytes to word by combining high and low bytes

Parameters

high_byte	High byte of word
low_byte	Low byte of word

Returns

16-bit Word

5.16.1.4 crc16()

```
unsigned short crc16 (
            unsigned char * ptr,
            unsigned char len )
```

Performs CRC check on received data

Parameters

ptr	Byte array containing received data
len	Length of data to check using CRC

Returns

1 if CRC check OK; 0 otherwise

5.16.1.5 CRC_check()

```
uint8_t CRC_check ( )
```

Performs CRC check utilizing the crc16 function

5.16.1.6 I2C2_ack()

```
void I2C2_ack (
            void  )
```

Acknowledges the received data over I2C via a single bit transmission

5.16.1.7 I2C2_receive_byte()

```
uint8_t I2C2_receive_byte ( )
```

Recieves byte over I2C2

Returns

Single byte read over I2C2

5.16.1.8 I2C2_transmit_byte()

```
void I2C2_transmit_byte (
            uint8_t byte )
```

Transmit single byte over I2C

Parameters

byte	Single byte to transmit

5.16.1.9 I2C2_transmit_restart_bit()

```
void I2C2_transmit_restart_bit ( )
```

Transmit single bit for restarting I2C communication

5.16.1.10 I2C2_transmit_start_bit()

```
void I2C2_transmit_start_bit ( )
```

Transmit single bit for starting I2C communication

5.16.1.11 I2C2_transmit_stop_bit()

```
void I2C2_transmit_stop_bit ( )
```

Transmit single bit for stoping I2C communication

5.16.1.12 I2C2_wait_while_busy()

```
void I2C2_wait_while_busy ( )
```

Waits while I2C2 is busy reading or writing

5.16.1.13 open_I2C2()

```
void open_I2C2 ( )
```

Opens I2C2 peripheral

5.16.1.14 sample_Temperature_Humidity()

```
uint8_t sample_Temperature_Humidity (
            uint16_t * temperature,
            uint16_t * relative_humidity )
```

Samples temperature and humidity readings from sensor

Parameters

temperature	Pointer to 16-bit word to save temperature reading
relative_humidity	Pointer to 16-bit word to save relative humidity reading

Returns

1 if CRC check was OK; 0 otherwise

5.16.1.15 sample_Temperature_Humidity_bytes()

```
uint8_t sample_Temperature_Humidity_bytes (
            uint8_t * temperature_bytes,
            uint8_t * relative_humidity_bytes )
```

Samples temperature and humidity readings from sensor

Parameters

temperature_bytes	Pointer to 8-bit byte array to save temperature reading as bytes
relative_humidity_bytes	Pointer to 8-bit byte array to save relative humidity reading as bytes

Returns

1 if CRC check was OK; 0 otherwise

5.16.1.16 setup_Temperature_Humidity_Sensor()

```
void setup_Temperature_Humidity_Sensor ( )
```

Sets up the AM2320 temperature sensor

5.17 src/SSN_API/Drivers/TEMPERATURE_SENSOR/temperature_↩ sensor.h File Reference

```
#include "../../global.h"
#include <stdint.h>
#include <plib.h>
```

Macros

- #define _SUPPRESS_PLIB_WARNING
- #define _DISABLE_OPENADC10_CONFIGPORT_WARNING
- #define AM2320_I2C_Address 0xB8
- #define AM2320_Read_Function_Code 0x03
- #define AM2320_Starting_Address 0x00
- #define AM2320_Num_Bytes_Requested 0x04
- #define MIN_NORMAL_TEMPERATURE 0
- #define MAX_NORMAL_TEMPERATURE 60
- #define MIN_NORMAL_RELATIVE_HUMIDITY 0
- #define MAX_NORMAL_RELATIVE_HUMIDITY 100
- #define NORMAL_AMBIENT_CONDITION 0
- #define ABNORMAL_AMBIENT_CONDITION 1

Functions

- void open_I2C2 ()
- void I2C2_wait_while_busy ()
- void I2C2_transmit_start_bit ()
- void I2C2_transmit_stop_bit ()
- void I2C2_transmit_restart_bit ()
- void I2C2_transmit_byte (uint8_t byte)
- uint8_t I2C2_receive_byte ()
- void I2C2_ack (void)
- void AM2320_I2C2_Read_Temp_and_Humidity ()
- unsigned short crc16 (unsigned char ∗ptr, unsigned char len)
- uint8_t CRC_check ()
- uint16_t convert_bytes_to_word (int8_t high_byte, int8_t low_byte)
- void setup_Temperature_Humidity_Sensor ()
- uint8_t sample_Temperature_Humidity (uint16_t ∗temperature, uint16_t ∗relative_humidity)
- uint8_t sample_Temperature_Humidity_bytes (uint8_t ∗temperature_bytes, uint8_t ∗relative_humidity_bytes)
- uint8_t ambient_condition_status ()

Variables

- uint8_t recv_data [8]

5.17.1 Macro Definition Documentation

5.17.1.1 _DISABLE_OPENADC10_CONFIGPORT_WARNING

```
#define _DISABLE_OPENADC10_CONFIGPORT_WARNING
```

5.17.1.2 _SUPPRESS_PLIB_WARNING

```
#define _SUPPRESS_PLIB_WARNING
```

5.17.1.3 ABNORMAL_AMBIENT_CONDITION

```
#define ABNORMAL_AMBIENT_CONDITION 1
```

5.17.1.4 AM2320_I2C_Address

```
#define AM2320_I2C_Address 0xB8
```

5.17.1.5 AM2320_Num_Bytes_Requested

```
#define AM2320_Num_Bytes_Requested 0x04
```

5.17.1.6 AM2320_Read_Function_Code

```
#define AM2320_Read_Function_Code 0x03
```

5.17.1.7 AM2320_Starting_Address

```
#define AM2320_Starting_Address 0x00
```

5.17.1.8 MAX_NORMAL_RELATIVE_HUMIDITY

```
#define MAX_NORMAL_RELATIVE_HUMIDITY 100
```

Maximum normal ambient relative humidity

5.17.1.9 MAX_NORMAL_TEMPERATURE

```
#define MAX_NORMAL_TEMPERATURE 60
```

Maximum normal ambient temperature

5.17.1.10 MIN_NORMAL_RELATIVE_HUMIDITY

```
#define MIN_NORMAL_RELATIVE_HUMIDITY 0
```

Minimum normal ambient relative humidity

5.17.1.11 MIN_NORMAL_TEMPERATURE

```
#define MIN_NORMAL_TEMPERATURE 0
```

Minimum normal ambient temperature

5.17.1.12 NORMAL_AMBIENT_CONDITION

```
#define NORMAL_AMBIENT_CONDITION 0
```

5.17.2 Function Documentation

5.17.2.1 AM2320_I2C2_Read_Temp_and_Humidity()

```
void AM2320_I2C2_Read_Temp_and_Humidity ( )
```

Reads the temperature and humidity bytes from AM2320 sensor using I2C. Read values are written into recv_data array

5.17.2.2 ambient_condition_status()

```
uint8_t ambient_condition_status ( )
```

Gets ambient condition status

Returns

NORMAL_AMBIENT_CONDITION if normal; **ABNORMAL_AMBIENT_CONDITION** otherwise.

5.17.2.3 convert_bytes_to_word()

```
uint16_t convert_bytes_to_word (
            int8_t high_byte,
            int8_t low_byte )
```

Converts bytes to word by combining high and low bytes

Parameters

high_byte	High byte of word
low_byte	Low byte of word

Returns

16-bit Word

5.17.2.4 crc16()

```
unsigned short crc16 (
            unsigned char * ptr,
            unsigned char len )
```

Performs CRC check on received data

Parameters

ptr	Byte array containing received data
len	Length of data to check using CRC

Returns

1 if CRC check OK; 0 otherwise

5.17.2.5 CRC_check()

```
uint8_t CRC_check ( )
```

Performs CRC check utilizing the crc16 function

5.17.2.6 I2C2_ack()

```
void I2C2_ack (
            void  )
```

Acknowledges the received data over I2C via a single bit transmission

5.17.2.7 I2C2_receive_byte()

```
uint8_t I2C2_receive_byte ( )
```

Recieves byte over I2C2

Returns

Single byte read over I2C2

5.17.2.8 I2C2_transmit_byte()

```
void I2C2_transmit_byte (
            uint8_t byte )
```

Transmit single byte over I2C

Parameters

byte	Single byte to transmit

5.17.2.9 I2C2_transmit_restart_bit()

```
void I2C2_transmit_restart_bit ( )
```

Transmit single bit for restarting I2C communication

5.17.2.10 I2C2_transmit_start_bit()

```
void I2C2_transmit_start_bit ( )
```

Transmit single bit for starting I2C communication

5.17.2.11 I2C2_transmit_stop_bit()

```
void I2C2_transmit_stop_bit ( )
```

Transmit single bit for stoping I2C communication

5.17.2.12 I2C2_wait_while_busy()

```
void I2C2_wait_while_busy ( )
```

Waits while I2C2 is busy reading or writing

5.17.2.13 open_I2C2()

```
void open_I2C2 ( )
```

Opens I2C2 peripheral

5.17.2.14 sample_Temperature_Humidity()

```
uint8_t sample_Temperature_Humidity (
            uint16_t * temperature,
            uint16_t * relative_humidity )
```

Samples temperature and humidity readings from sensor

Parameters

temperature	Pointer to 16-bit word to save temperature reading
relative_humidity	Pointer to 16-bit word to save relative humidity reading

Returns

1 if CRC check was OK; 0 otherwise

5.17.2.15 sample_Temperature_Humidity_bytes()

```
uint8_t sample_Temperature_Humidity_bytes (
            uint8_t * temperature_bytes,
            uint8_t * relative_humidity_bytes )
```

Samples temperature and humidity readings from sensor

Parameters

temperature_bytes	Pointer to 8-bit byte array to save temperature reading as bytes
relative_humidity_bytes	Pointer to 8-bit byte array to save relative humidity reading as bytes

Returns

1 if CRC check was OK; 0 otherwise

5.17.2.16 setup_Temperature_Humidity_Sensor()

```
void setup_Temperature_Humidity_Sensor ( )
```

Sets up the AM2320 temperature sensor

5.17.3 Variable Documentation

5.17.3.1 recv_data

```
uint8_t recv_data[8]
```

The data received from the temperature sensor AM2320, i.e., Control byte, number of bytes' byte, 4 data bytes, 2 CRC bytes

5.18 src/SSN_API/Drivers/UART/uart.c File Reference

```
#include "uart.h"
```

Functions

- void open_UART2 (unsigned int baudrate)
- void setup_printf (unsigned int baudrate)
- int SerialTransmit_UART2 (const char ∗buffer)
- unsigned int SerialReceive_UART2 (char ∗buffer, unsigned int max_size)

5.18.1 Function Documentation

5.18.1.1 open_UART2()

```
void open_UART2 (
            unsigned int baudrate )
```

Opens UART peripheral for communication

Parameters

baudrate	

5.18.1.2 SerialReceive_UART2()

```
unsigned int SerialReceive_UART2 (
            char * buffer,
            unsigned int max_size )
```

Receives a byte array over serial interface

Parameters

buffer	A pointer to a byte array to write the received message into
max_size	Maximum number of bytes expected to be received over serial interface

Returns

0 if successful

5.18.1.3 SerialTransmit_UART2()

```
int SerialTransmit_UART2 (
            const char * buffer )
```

Transmits a byte array over serial interface

Parameters

buffer	A pointer to a byte array to send over the UART

Returns

0 if successful

5.18.1.4 setup_printf()

```
void setup_printf (
            unsigned int baudrate )
```

Sets up the print function at a specific baudrate

Parameters

baudrate	

5.19 src/SSN_API/Drivers/UART/uart.h File Reference

```
#include "../../global.h"
```

Functions

- void open_UART2 (unsigned int baudrate)
- int SerialTransmit_UART2 (const char ∗buffer)
- unsigned int SerialReceive_UART2 (char ∗buffer, unsigned int max_size)
- void setup_printf (unsigned int baudrate)

5.19.1 Function Documentation

5.19.1.1 open_UART2()

```
void open_UART2 (
            unsigned int baudrate )
```

Opens UART peripheral for communication

Parameters

baudrate	

5.19.1.2 SerialReceive_UART2()

```
unsigned int SerialReceive_UART2 (
            char * buffer,
            unsigned int max_size )
```

Receives a byte array over serial interface

Parameters

buffer	A pointer to a byte array to write the received message into
max_size	Maximum number of bytes expected to be received over serial interface

Returns

0 if successful

5.19.1.3 SerialTransmit_UART2()

```
int SerialTransmit_UART2 (
            const char * buffer )
```

Transmits a byte array over serial interface

Parameters

buffer	A pointer to a byte array to send over the UART

Returns

0 if successful

5.19.1.4 setup_printf()

```
void setup_printf (
            unsigned int baudrate )
```

Sets up the print function at a specific baudrate

Parameters

baudrate	

5.20 src/SSN_API/FlashMemory/FlashMemory.c File Reference

```
#include "FlashMemory.h"
```

Functions

- uint8_t FindMACInFlashMemory (uint8_t *SSN_MAC_ADDRESS, uint8_t *SSN_DEFAULT_MAC)
- uint8_t FindSensorConfigurationsInFlashMemory (uint8_t *SSN_CONFIG, uint8_t *SSN_REPORT_INTERVAL, uint8_t *SSN_CURRENT_SENSOR_RATINGS, uint8_t *SSN_CURRENT_SENSOR_THRESHOLDS, uint8_t *SSN_CURRENT_SENSOR_MAXLOADS)

5.20.1 Function Documentation

5.20.1.1 FindMACInFlashMemory()

```
uint8_t FindMACInFlashMemory (
            uint8_t * SSN_MAC_ADDRESS,
            uint8_t * SSN_DEFAULT_MAC )
```

Finds MAC address in EEPROM

Parameters

SSN_MAC_ADDRESS	Six byte array containing the six bytes of SSN MAC address
SSN_DEFAULT_MAC	Six byte array containing the six bytes of default SSN MAC address in case nothing in found in EEPROM

Returns

NO_CONFIG_STATE if MAC address is found in EEPROM; else **NO_MAC_STATE**

5.20.1.2 FindSensorConfigurationsInFlashMemory()

```
uint8_t FindSensorConfigurationsInFlashMemory (
            uint8_t * SSN_CONFIG,
            uint8_t * SSN_REPORT_INTERVAL,
            uint8_t * SSN_CURRENT_SENSOR_RATINGS,
            uint8_t * SSN_CURRENT_SENSOR_THRESHOLDS,
            uint8_t * SSN_CURRENT_SENSOR_MAXLOADS )
```

Finds Current Sensor Configurations in EEPROM

Parameters

SSN_CONFIG	Byte array in which current sensor configurations will be written
SSN_REPORT_INTERVAL	Pointer to byte variable containing SSN status update interval (period of SSN status updates, e.g., 1 sec)
SSN_CURRENT_SENSOR_RATINGS	Byte array in which current sensor ratings will be written
SSN_CURRENT_SENSOR_THRESHOLDS	Byte array in which machine threshold currents will be written to decide IDLE state for machines
SSN_CURRENT_SENSOR_MAXLOADS	Byte array in which machine maximum load currents will be written to calculate load percentages

Returns

NO_TIMEOFDAY_STATE if sensor configurations are found in EEPROM; else **NO_CONFIG_STATE**

5.21 src/SSN_API/FlashMemory/FlashMemory.h File Reference

```
#include "../global.h"
#include "../Drivers/UART/uart.h"
#include "../Drivers/EEPROM/eeprom.h"
```

Functions

- uint8_t FindMACInFlashMemory (uint8_t *SSN_MAC_ADDRESS, uint8_t *SSN_DEFAULT_MAC)
- uint8_t FindSensorConfigurationsInFlashMemory (uint8_t *SSN_CONFIG, uint8_t *SSN_REPORT_INTERVAL, uint8_t *SSN_CURRENT_SENSOR_RATINGS, uint8_t *SSN_CURRENT_SENSOR_THRESHOLDS, uint8_t *SSN_CURRENT_SENSOR_MAXLOADS)

5.21.1 Function Documentation

5.21.1.1 FindMACInFlashMemory()

```
uint8_t FindMACInFlashMemory (
            uint8_t * SSN_MAC_ADDRESS,
            uint8_t * SSN_DEFAULT_MAC )
```

Finds MAC address in EEPROM

Parameters

SSN_MAC_ADDRESS	Six byte array containing the six bytes of SSN MAC address
SSN_DEFAULT_MAC	Six byte array containing the six bytes of default SSN MAC address in case nothing in found in EEPROM

Returns

NO_CONFIG_STATE if MAC address is found in EEPROM; else **NO_MAC_STATE**

5.21.1.2 FindSensorConfigurationsInFlashMemory()

```
uint8_t FindSensorConfigurationsInFlashMemory (
            uint8_t * SSN_CONFIG,
            uint8_t * SSN_REPORT_INTERVAL,
            uint8_t * SSN_CURRENT_SENSOR_RATINGS,
            uint8_t * SSN_CURRENT_SENSOR_THRESHOLDS,
            uint8_t * SSN_CURRENT_SENSOR_MAXLOADS )
```

Finds Current Sensor Configurations in EEPROM

Parameters

SSN_CONFIG	Byte array in which current sensor configurations will be written
SSN_REPORT_INTERVAL	Pointer to byte variable containing SSN status update interval (period of SSN status updates, e.g., 1 sec)
SSN_CURRENT_SENSOR_RATINGS	Byte array in which current sensor ratings will be written
SSN_CURRENT_SENSOR_THRESHOLDS	Byte array in which machine threshold currents will be written to decide IDLE state for machines
SSN_CURRENT_SENSOR_MAXLOADS	Byte array in which machine maximum load currents will be written to calculate load percentages

Returns

NO_TIMEOFDAY_STATE if sensor configurations are found in EEPROM; else **NO_CONFIG_STATE**

5.22 src/SSN_API/global.h File Reference

```
#include <xc.h>
#include <p32xxxx.h>
#include <plib.h>
#include <stdint.h>
#include <stdbool.h>
#include "Drivers/PSEUDO_RTCC/pseudo_rtcc.h"
```

Macros

- #define _SUPPRESS_PLIB_WARNING
- #define _DISABLE_OPENADC10_CONFIGPORT_WARNING
- #define SYSTEM_CLK 60000000
- #define PERIPH_CLK 30000000
- #define SSN_DEFAULT_PORT 8888
- #define RED_LED BIT_2
- #define GREEN_LED BIT_3
- #define SSN_IS_ALIVE 100
- #define SELF_TEST_FAILED_STATE 0
- #define NO_CURRENT_SENSOR_STATE 1
- #define NO_ETHERNET_STATE 2
- #define NO_MAC_STATE 3
- #define NO_CONFIG_STATE 4
- #define ACK_CONFIG_STATE 5
- #define NO_TIMEOFDAY_STATE 6
- #define ABNORMAL_ACTIVITY_STATE 7
- #define NORMAL_ACTIVITY_STATE 8
- #define EEPROM_MAC_LOC 0
- #define EEPROM_CONFIG_LOC 12
- #define EEPROM_MAC_SIZE 6
- #define EEPROM_CONFIG_SIZE 13
- #define TIME_Of_DAY_SIZE 6
- #define NO_OF_MACHINES 4

5.22.1 Macro Definition Documentation

5.22.1.1 _DISABLE_OPENADC10_CONFIGPORT_WARNING

```
#define _DISABLE_OPENADC10_CONFIGPORT_WARNING
```

5.22.1.2 _SUPPRESS_PLIB_WARNING

```
#define _SUPPRESS_PLIB_WARNING
```

5.22.1.3 ABNORMAL_ACTIVITY_STATE

```
#define ABNORMAL_ACTIVITY_STATE 7
```

5.22.1.4 ACK_CONFIG_STATE

```
#define ACK_CONFIG_STATE 5
```

5.22.1.5 EEPROM_CONFIG_LOC

```
#define EEPROM_CONFIG_LOC 12
```

5.22.1.6 EEPROM_CONFIG_SIZE

```
#define EEPROM_CONFIG_SIZE 13
```

5.22.1.7 EEPROM_MAC_LOC

```
#define EEPROM_MAC_LOC 0
```

5.22.1.8 EEPROM_MAC_SIZE

```
#define EEPROM_MAC_SIZE 6
```

5.22.1.9 GREEN_LED

```
#define GREEN_LED BIT_3
```

5.22.1.10 NO_CONFIG_STATE

```
#define NO_CONFIG_STATE 4
```

5.22.1.11 NO_CURRENT_SENSOR_STATE

```
#define NO_CURRENT_SENSOR_STATE 1
```

5.22.1.12 NO_ETHERNET_STATE

```
#define NO_ETHERNET_STATE 2
```

5.22.1.13 NO_MAC_STATE

```
#define NO_MAC_STATE 3
```

5.22.1.14 NO_OF_MACHINES

```
#define NO_OF_MACHINES 4
```

5.22.1.15 NO_TIMEOFDAY_STATE

```
#define NO_TIMEOFDAY_STATE 6
```

5.22.1.16 NORMAL_ACTIVITY_STATE

```
#define NORMAL_ACTIVITY_STATE 8
```

5.22.1.17 PERIPH_CLK

```
#define PERIPH_CLK 30000000
```

5.22.1.18 RED_LED

```
#define RED_LED BIT_2
```

5.22.1.19 SELF_TEST_FAILED_STATE

```
#define SELF_TEST_FAILED_STATE 0
```

5.22.1.20 SSN_DEFAULT_PORT

```
#define SSN_DEFAULT_PORT 8888
```

5.22.1.21 SSN_IS_ALIVE

```
#define SSN_IS_ALIVE 100
```

5.22.1.22 SYSTEM_CLK

```
#define SYSTEM_CLK 60000000
```

5.22.1.23 TIME_Of_DAY_SIZE

```
#define TIME_Of_DAY_SIZE 6
```

5.23 src/SSN_API/SSN_API.h File Reference

```
#include <plib.h>
#include "SystemTests/SystemTests.h"
#include "FlashMemory/FlashMemory.h"
#include "Connection/Connection.h"
#include "Communication/Communication.h"
```

5.24 src/SSN_API/SystemTests/SystemTests.c File Reference

```
#include "SystemTests.h"
```

Functions

- void RunSystemTests ()

5.24.1 Function Documentation

5.24.1.1 RunSystemTests()

```
void RunSystemTests ( )
```

Runs system diagnostic tests for checking if important peripherals are functioning properly; does not return if found faulty. Checks the following peripherals in that order.

- EEPROM Read/Write
- Temperature and Humidity Sensor
- Ethernet Physical Connection

5.25 src/SSN_API/SystemTests/SystemTests.h File Reference

```
#include "../global.h"
#include "../Drivers/UART/uart.h"
#include "../Drivers/EEPROM/eeprom.h"
#include "../Drivers/NETWORK/network.h"
#include "../Drivers/TEMPERATURE_SENSOR/temperature_sensor.h"
```

Functions

- void RunSystemTests ()

Variables

- uint8_t temperature_bytes [2]
- uint8_t relative_humidity_bytes [2]

5.25.1 Function Documentation

5.25.1.1 RunSystemTests()

```
void RunSystemTests ( )
```

Runs system diagnostic tests for checking if important peripherals are functioning properly; does not return if found faulty. Checks the following peripherals in that order.

- EEPROM Read/Write
- Temperature and Humidity Sensor
- Ethernet Physical Connection

5.25.2 Variable Documentation

5.25.2.1 relative_humidity_bytes

```
uint8_t relative_humidity_bytes[2]
```

Relative humidity reading bytes for internal testing only

5.25.2.2 temperature_bytes

```
uint8_t temperature_bytes[2]
```

Temperature reading bytes for internal testing only