STSW45x0C LMAC API
ED1P4

Document history

	Date	Author	Description
ED1P1	10th March 2008	WLAN BU	Initial Draft of LMAC API document
ED1P2	31st March 2008	WLAN BU	Incorporated review comments.
ED1P3	18th July 2008	WLAN BU	Some additional changes.
ED1P4	4th September, 2008	WLAN BU	Edited Legal Notice

Legal notices

For selected STMicroelectronics sales offices fax: France +33 1 47407910; Germany +49 89 4605454; Italy +39 02 8250449; Japan +81 3 57838216; Singapore +65 4820240; Sweden +46 8 7504950; Switzerland +41 22 9292900; United Kingdom and Eire +44 1628 890391; USA +1 781 861 2678

Index

1 Scope

This document defines the LMAC Application Programmers Interface (API).

1.1 Document Overview

Section 2 of this document defines the LMAC data formats, structures and objects. Section 3 uses these definitions to describe the various interconnect mechanisms, initialization, setup, frame transmission and reception, PSM, scanning, etc.

2 Interconnect

2.1 Basic types and compiler settings

The LMAC API is defined by the following basic types:

int8_t	Signed 8 bit type.
int16_t	Signed 16 bit type.
int32_t	Signed 32 bit type.
int64_t	Signed 64 bit type.
uint8_t	Unsigned 8 bit type.
uint16_t	Unsigned 16 bit type.
uint32_t	Unsigned 32 bit type.
uint64_t	Unsigned 64 bit type.

All structures that may appear in arrays are padded to be a multiple of 32 bits in length. The compiler that compiles an application that uses the API shall be configured to round structures to multiples of 32 bit.

2.2 LMAC description

The properties of the LMAC are described by the following structure:

```
struct s_lm_descr
{
   uint16_t modes;
   uint16_t flags;
   uint32_t buffer_start;
   uint32_t buffer_end;
   uint8_t header;
   uint8_t trailer;
   uint8_t tx_queues;
   uint8_t tx_depth;
   uint8_t privacy;
   uint8_t rx_keycache;
   uint8_t tim_size;
   uint8_t pad1;
   uint8_t rates[16];
   uint32_t link;
   uint16_t mtu;
};
```

modes	Bit-mask with supported LMAC modes.
flags	SoftMAC LMAC-UMAC interface definitions. Currently, only LM_DESCR_FLAG_SHMEM is supported
buffer_start	Start address of application managed buffer.

buffer_end	End address of application managed buffer.
header	Minimal LMAC required header space.
trailer	Minimal LMAC required trailer space.
tx_queues	Number of supported (E)DCF Tx queues.
tx_depth	Maximum Tx queue depth.
privacy	Bit-mask with supported LMAC privacy acceleration engines.
rx_keycache	Decryption key cache size.
tim_size	Maximum TIM size in bytes.
rates	PHY supported rates in ascending order.
link	Single-Chip SoftMAC LMAC-UMAC link
mtu	Maximum Transmission Unit

The `modes` bitmask describes the mode supported by the LMAC. The values of the LM_MODE_... enumeration correspond to the bit-numbers in the bit-mask.
The LMAC implements a fixed buffer that is managed by the application. The application must use this to store transmission related structure and data in, such as frames and configuration objects. The application must also configure the part the LMAC must use as receive buffer. The address of the buffer and (implicitly) the buffer's size are specified by `buffer_start` and `buffer_end`. The `header` member specifies the minimum space required to be reserved for the LMAC to maintain private structures in. The application must reserve this heading space for each frame, object or other entity it stores in the application managed buffer.
The LMAC at least supports 5 queues (beacon, scan, management, multicast/broadcast and data queues. Additional priorities queues may be available.
The LMAC may support hardware acceleration for certain privacy mechanisms. The `privacy` bitmask describes the privacy mechanisms supported by the LMAC. The following bits are defined:

- LM_PRIVACC_WEP: WEP accelerator present;
- LM_PRIVACC_TKIP: TKIP accelerator present (requires LM_PRIVACC_WEP);
- LM_PRIVACC_MICHAEL: MICHAEL accelerator present (requires LM_PRIVACC_WEP and LM_PRIVACC_TKIP);
- LM_PRIVACC_CCX_KP: CCX Key Protocol (requires LM_PRIVACC_WEP).
- LM_PRIVACC_CCX_MIC: CCX MIC
- LM_PRIVACC_AES_CCMP: AES CCMP supported.

The LMAC implements a fixed receive/decrypt key cache. The application is responsible for managing and updating the cache. If the frame cannot be decrypted by the LMAC (cache-miss), the frame is passed undecrypted and the application can change the cache's contents if necessary.

2.3 LMAC Headers

2.3.1 Generic properties

Each LMAC header structure starts with a `handle` and a `flags` member. The `handle` is opaque to the LMAC. The `handle` is returned to the application for operations that result in a response or feedback and may be used by the application to match the response to the request. The LMAC uses the LM_FLAG_CONTROL flag of this member to determine whether the header represents a control or data header.
Frames submitted to the application may be unaligned, or may become unaligned due to changes to the frame's header. However, certain LMAC – application implementations require that the structures are aligned before they are submitted to the LMAC. In this case, the application or the LMAC may set the LM_FLAG_ALIGN flag to

indicate that the first byte of the data part of the control structure and data structure contains the number of padding bytes (including the number) heading the actual data.

2.3.2 Control format

Structure

```
Struct  s_lm_control
{
        uint16_t    flags;
        uint16_t    length;
        uint32_t    handle;
        uint16_t    oid;
        uint16_t    pad;
        /* uint8_t  data[]; */
};
```

flags	One of the following flags; • LM_CTRL_OPSET: if set the operation is a set, a get otherwise. A trap also sets this flag, as opposed to a get response. LM_FLAG_CONTROL is always set.
length	Length of data part
handle	LMAC opaque application handle.
oid	Object Identifier
data	Object data

2.3.3 Data format

```
union u_lm_data
{
    struct s_lm_data_out out;
    struct s_lm_data_in in;
};
```

- **Outgoing data**

```
struct s_lm_data_out
{
    uint16_t flags;
    uint16_t length;
    uint32_t handle;
    uint16_t aid;
    uint8_t rts_retries;
    uint8_t retries;
    uint8_t aloft[8];
    uint8_t aloft_ctrl;
    uint8_t crypt_offset;
    uint8_t keytype;
    uint8_t keylen;
    uint8_t key[16];
    uint8_t queue;
    uint8_t backlog;
    uint16_t durations[4];
    uint8_t antenna;
    uint8_t cts;
    int16_t power;
    uint8_t pad[2];
    /*uint8_t data[];*/
};
```

flags	Frame transmission properties • LM_OUT_PROMISC: send frame without protocol interference. The rate at which the frame is sent, is the rate specified by the first entry of the aloft member. • LM_OUT_TIMESTAMP: send frame and insert TSF synchronized timestamp at appropriate offset. (optional) • LM_OUT_SEQNR: the frame's sequence number has already been set by the application. • LM_OUT_BURST: frame may be used to start a burst, of length configured by the EDCF object's maxburst member • LM_OUT_NOCANCEL: prevent PSM automatic cancelation for this frame. • LM_OUT_CLEARTIM: clears TIM for frame's aid. • LM_OUT_HITCHHIKE: hitchhike this frame on an SIFS response to a QoS data frame as a QoS data + acknowledgment frame, irrespective whether QoS poll is set. • LM_OUT_COMPRESS: frame is eligible for compression. • LM_OUT_CONCAT: frame is eligible for concatenation. • LM_OUT_PCS_ACCEPT: frame is eligible for MCDLP LM_FLAG_CONTROL is always cleared.
length	Length of the data part.
handle	LMAC opaque application handle.
aid	AID corresponding to the destination of the frame.
rts_retries	Maximum number of RTS attempts for each transmission attempt that requires an RTS.
retries	Maximum number of transmission attempts for this frame
aloft	ALOFt array with bit-rate, preamble and RTS/CTS-to-self options for first 8 transmission attempts. Each entry consists of a rate index and one of the following flags; • LM_ALOFT_RTS: use RTS/CTS. • LM_ALOFT_CTS: use CTS-to-self. • LM_ALOFT_SP: use Short Preamble. • LM_ALOFT_MASK: • LM_ALOFT_RATE:
aloft_ctrl	Index in Aloft control array as configured by BSS setup object. The index determines the rate and preamble settings for the initial attempt of a RTS or CTS-to-self.
crypt_offset	Specifies the offset at which the frame must

	be in-frame encrypted with the privacy mechanism and key specified.
keytype	Encryption key type • LM_PRIV_NONE • LM_PRIV_WEP • LM_PRIV_TKIP • LM_PRIV_TKIPMICHAEL • LM_PRIV_CCX_WEPMI C • LM_PRIV_CCX_KPMIC • LM_PRIV_CCX_KP • LM_PRIV_AES_CCMP
keylen	Encryption key length
key	Encryption key
queue	Queue number. The following fixed queues are defined; • LM_QUEUE_BEACON • LM_QUEUE_SCAN • LM_QUEUE_MGT • LM_QUEUE_MCBC • LM_QUEUE_DATA • LM_QUEUE_DATA0..3
backlog	Number of backlogged frames for given queue.
durations	Durations of first 4 backlogged frames for given queue.
antenna	Preferred transmission antenna.
cts	cts rate
power	Unused.

- **Incoming data**

```
struct s_lm_data_in
{
    uint16_t flags;
    uint16_t length;
    uint16_t frequency;
    uint8_t antenna;
    uint8_t rate;
    uint8_t rcpi;
    uint8_t sq;
    uint8_t decrypt;
    uint8_t rssi_raw;
    uint32_t clock[2];
    /*uint8_t data[];*/
};
```

flags	Frame receive properties; • LM_IN_FCS_GOOD: FCS matched. • LM_IN_MATCH_MAC: address1 matches local

	mac address. • LM_IN_MCBC: multicast/broadcast bit set. • LM_IN_BEACON: frame is a beacon frame. • LM_IN_MATCH_BSS: bssid matches local bssid. • LM_IN_BCAST_BSS: bssid is broadcast address. • LM_IN_DATA: frame contains data. • LM_IN_TRUNCATED: frame is truncated. • LM_IN_TRANSPARENT:
length	Length of data part.
frequency	Frequency in MHz on which the frame is received
antenna	Antenna on which the frame is received.
rate	Rate index of rate on which the frame is received
rcpi	Received Channel Power Indicator, as returned by the hardware. Conversion in dBm must be done by the application.
sq	Signal Quality (optional)
decrypt	Decrypt status; • LM_DECRYPT_NONE: no decryption required • LM_DECRYPT_OK: decrypted correctly • LM_DECRYPT_NOKEY: not decrypted because key-cache lacked proper key. • LM_DECRYPT_NOMICHAEL: decrypted, but no Michael not verified. • LM_DECRYPT_NOCKIPMIC: • LM_DECRYTP_FAIL_WEP: decrypted WEP engine, ICV failed (note that WEP is also used for TKIP decryption). • LM_DECRYPT_FAIL_TKIP: decrypted, TKIP sequence check failed. • LM_DECRYPT_FAIL_MICHAEL: decrypted, but Michael failed. • LM_DECRYPT_FAIL_CKIPKP: • LM_DECRYPT_FAIL_CKIPMIC: • LM_DECRYPT_FAIL_AESCCMP:
clock	µsec accurate timestamp of hardware clock at end of frame (before OFDM SIFS EOF padding)

2.4 Objects

2.4.1 Actions

- **Setup**

Object

LM_OID_SETUP

```
struct s_lmo_setup
{
    uint16_t flags;
    uint8_t  macaddr[6];
    uint8_t  bssid[6];
    uint8_t  antenna;
    uint8_t  rx_align;
    uint32_t rx_buffer;
    uint16_t rx_mtu;
    uint16_t frontend;
    uint16_t timeout;
    uint16_t truncate;
    uint32_t bratemask;
    uint8_t  sbss_offset;
    uint8_t  mcast_window;
    uint8_t  rx_rssi_threshold;
    uint8_t  rx_ed_threshold;
    uint32_t ref_clock;
    uint16_t lpf_bandwidth;
    uint16_t osc_start_delay ;
} ;
```

flags	BSS setup flags; • LM_SETUP_INFRA • LM_SETUP_IBSS • LM_SETUP_AP • LM_SETUP_TRANSPARENT (optional) • LM_SETUP_PROMISCUOUS • LM_SETUP_HIBERNATE • LM_SETUP_NOACK
macaddr	LMAC MAC address
bssid	BSSID of the BSS for low level frame filtering purposes. Set to FF-FF-FF-FF-FF if the LMAC is not associated with any BSS.
antenna	Receive antenna: • LM_ANTENNA_0 • LM_ANTENNA_1 • LM_ANTENNA_DIVERSITY
rx_align	Alignment for received frames. 0 disables receive alignment, 1 to 4 specifies the alignment of the frame’s data body – independent of the frame’s MAC header length. Alignment is achieved by adding padding bytes and using the LM_FLAG_ALIGN flag.
rx_buffer	Address of receive buffer (ignored after initial setup).
rx_mtu	Maximum Transmission Unit for reception.
frontend	Frontend configuration (1 is default). Defined numbers are: • LM_FRONTEND_DUETTE3, • LM_FRONTEND_DUETTE2,

	• LM_FRONTEND_FRISBEE, • LM_FRONTEND_CROSSBOW, • LM_FRONTEND_LONGBOW
timeout	Beacon timeout in kµsec. If no beacons are received within the given period, the LM_TRAP_NO_BEACON trap is generated.
truncate	Truncate length of frames for which none of the LM_IN_MATCH_MAC, LM_IN_MATCH_MCBC, or LM_IN_MATCH_BSS flags is set[1]. Truncated frames are marked by LM_IN_TRUNCATED.
bratemask	Basic rate mask.
sbss_offset	Sequential BSS parameter (deprecated)
mcast_window	Sequential BSS parameter (deprecated)
rx_rssi_threshold	Received frames with a RSSI below the specified threshold shall be discarded in the LMAC. The value is in hardware-dependent units, and the UMAC needs to convert the dBm value to a RSSI value using the RSSI calibration data.
rx_ed_threshold	Hardware dependent value to set the corresponding baseband register. Not applied if 0.
ref_clock	Clock frequency of the reference oscillator.
lpf_bandwidth	
osc_start_delay	Minimum delay for the start of Oscillator.

Access

Write

Description

Configures the LMAC setup.
LM_SETUP_INFRA configures the LMAC in client infrastructure mode.
LM_SETUP_IBSS configures the LMAC in client IBSS mode.
The LM_SETUP_INFRA and LM_SETUP_IBSS flags are mutually exclusive.
LM_SETUP_TRANSPARENT flag configures the receive frame filter to pass all frames without regard to type and address matching. Frames are still responded to as if in normal operation.
The LM_SETUP_PROMISCUOUS flag configures the EDCF to run in promiscuous mode where all received frames are passed without filtering or acknowledgement. Note that to sent frames in promiscuous mode, the LM_OUT_PROMISC flags must be used when sending the frame.
The LM_SETUP_TRANSPARENT and LM_SETUP_PROMISCUOUS flags are mutually exclusive.
LM_SETUP_HIBERNATE configures the LMAC to go into a low power mode and refrain from participating in any BSS.
LM_SETUP_NOACK configures the LMAC to not send ACK frames to any frame types.
LM_SETUP_TIMESLICE enables the time-slicing mechanism.

- **Scan**

Object

LM_OID_SCAN

[1] Note that this is only useful when the LMAC is setup with either LM_SETUP_TRANSPARENT or LM_SETUP_PROMISCUOUS flags set.

```
struct s_lmo_scan
{
    uint16_t flags;
    uint16_t dwell;
    uint8_t channel[292];
    uint32_t bratemask;
    uint8_t  aloft[8];
    uint8_t rssical[8];
};
```

flags	Scanning flags, one of the following; • LM_SCAN_EXIT: exit from scanning to mode and change to the given frequency after the dwell time expires. • LM_SCAN_TRAP: generate a trap after the dwell time expires. • LM_SCAN_ACTIVE: active scan by sending a probe request on the specified frequency. • LM_SCAN_FILTER: turn on the "japan-filter".
dwell	Dwell time interval in units of kµsec after which a trap is generated.
channel	Channel data (calibration data).
bratemask	Basic rate mask (relative to rate indexing).
aloft	ALOFt control array, sets the basic rate retry pattern for RTS.
rssical	RSSI calibration data

Access

Write

Description

Instruct the LMAC to change frequency and scan on a specified frequency.

- **Trap**

Object

LM_OID_TRAP

```
struct s_lmo_trap
{
    uint16_t event;
    uint16_t frequency;
};
```

event	Numbered LMAC event. Currently defined numbers; • LM_TRAP_SCAN • LM_TRAP_TIMER • LM_TRAP_BEACON_TX • LM_TRAP_FAA_RADIO_ON

	• LM_TRAP_FAA_RADIO_OFF • LM_TRAP_RADAR • LM_TRAP_NO_BEACON • LM_TRAP_TBTT • LM_TRAP_SCO_ENTER • LM_TRAP_SCO_EXIT
frequency	Synthesizer frequency at the moment the trap is generated. Important for RADAR traps.

Access

Trap

Description

Generic trap object for signalling LMAC events.

- **Timer**

Object

LM_OID_TIMER

```
struct s_lmo_timer
{
      uint32_t interval;
};
```

interval	Specifies the interval in units of kµsec after which a trap with event number LM_TRAP_TIMER must be generated.

Access

Write

Description

Generic mechanism for generating an accurately timed LMAC trap.

- **NAV**

Object

LM_OID_NAV

```
Struct s_lmo_nav
{
      uint32_t period;
};
```

period	Specifies a period of time in units of µsec during which the LMAC must set an internal NAV.

Access

Write

Description

Generic mechanism for suspending transmissions for a given period.

2.4.2 Configuration objects

- **EDCF settings**

Object

LM_OID_EDCF

```
struct s_lmo_edcf
{
    uint8_t  flags;
    uint8_t  slottime;
    uint8_t  sifs;
    uint8_t  eofpad;
    struct s_lmo_edcf_queue
    {
        uint8_t  aifs;
        uint8_t  pad0;
        uint16_t cwmin;
        uint16_t cwmax;
        uint16_t txop;
    } queues[8];
    uint8_t  mapping[4];
    uint16_t maxburst;
    uint16_t round_trip_delay;
};
```

flags	EDCF flags
slottime	EDCF slottime in µsec
sifs	EDCF SIFS time
eofpad	802.11g OFDM End of Frame SIFS pad
aifs	AIFS settings per queue
cwmin	CWmin settings per queue
cwmax	CWmax settings per queue
txop	Maximum burst duration
mapping[4]	Maps the LMAC queues for beacons, probes, etc… to EDCF queues.
maxburst	Not used.
round_trip_delay	Extra roundtrip-delay for long-distance links.

Access

Write

Description

Configures the EDCF setup.

- **Key cache settings**

Object

LM_OID_KEYCACHE

```
struct s_lmo_keycache
{
    uint8_t entry;
    uint8_t keyid;
    uint8_t address[6];
    uint8_t pad[2];
    uint8_t keytype;
    uint8_t keylen;
    uint8_t key[24];
};
```

entry	Entry in cache to update
keyid	Privacy key identifier

address	Address of station to which the key belongs
keytype	Decryption key type • LM_PRIV_NONE, • LM_PRIV_WEP, • LM_PRIV_TKIP, • LM_PRIV_TKIPMICHAEL, • LM_PRIV_CCX_WEPMIC, • LM_PRIV_CCX_KPMIC, • LM_PRIV_CCX_KP, • LM_PRIV_AES_CCMP
keylen	Decryption key length
Key	Decryption key data

- **PSM**

Object

LM_OID_PSM

```
struct s_lmo_psm
{
    uint16_t    flags;
    uint16_t    aid;
    struct s_lm_interval
    {
        uint16_t interval;
        uint16_t periods;
    } intervals[4];
    uint8_t     beacon_rcpi_skip_max;
    uint8_t     rcpi_delta_threshold;
    uint8_t     nr;
    uint8_t     exclude[1];
};
```

flags	PSM behaviour flags; • LM_PSM: enter PSM mode. • LM_PSM_MCBC: pass beacons and stay awake for received group addressed frames. • LM_PSM_CHECKSUM: calculate checksum and wake-up application only if the checksum changes. • LM_PSM_DTIM:
aid	AID to monitor TIM bit for. Pass beacon frame if corresponding TIM bit is set.
interval	Listen interval in Beacon periods.
period	Number of listen intervals before switching to the next set of listen intervals in the list.
beacon_rcpi_skip_max	The number of Beacon's to be skipped.
rcpi_delta_threshold	It's the threshold value between current beacon rssi and last trapped beacon rssi.
nr	Number of element identifiers in exclude list.
exclude	Identifiers of elements to exclude from

	checksum calculation.

Access

Write

Description

Configures the LMAC client PSM behaviour.

2.4.3 Frame management objects

- **Tx cancel**

Object

LM_OID_TXCANCEL

```
struct s_lmo_txcancel
{
    uint32_t address[1];
};
```

address	Array of addresses of frames in the application managed buffer. Array shall at least hold one entry.

Access

Write

Description

Cancels frames on the LMAC.

- **Tx feedback**

Object

LM_OID_TX

```
struct s_lmo_tx
      {
            uint8_t     flags;
            uint8_t     retries;
            uint8_t     rcpi;
            uint8_t     sq;
            uint16_t    seqctrl;
            uint8_t     antenna;
            uint8_t     pad;
      };
```

flags	Feedback flags • LM_TX_FAILED: frame was exhaustively retried or cancelled. • LM_TX_PSM: PSM bit was set upon transmission by the LMAC. • LM_TX_PSM_CANCELLED: frame was cancelled by the automatic PSM cancellation mechanism.
retries	Number of retries needed before frame was

	successfully transmitted or cancelled.
rcpi	Received Channel Power Indicator, on acknowledgement frame, if frame was successfully transmitted. Conversion in dBm must be done by the application.
sq	Signal Quality on acknowledgment (optional).
seqctrl	Sequence Control field of the cancelled frame.
antenna	Antenna over which the frame was successfully transmitted.

Access

Trap

Description

Feedback trap of frame transmission mechanism.

- **Burst update**

Object

LM_OID_BURST

```
struct s_lmo_burst
{
    uint8_t  flags;
    uint8_t  queue;
    uint8_t  backlog;
    uint8_t  pad;
    uint16_t durations[32];
};
```

flags	TBD
queue	Queue number
backlog	Number of backlogged frames for given queue
durations[32]	Durations of up-to 32 backlogged frames for given queue

Access

Write

Description

Update object for bursting.

2.4.4 Statistics

- **LMAC statistics**

Object

LM_OID_STATS

```
struct s_lmo_stats
{
    uint32_t valid;
    uint32_t fcs;
    uint32_t abort;
    uint32_t phyabort;
    uint32_t rts_success;
    uint32_t rts_fail;
    uint32_t timestamp;
    uint32_t time_tx;
```

```
    uint32_t noisefloor;
    uint32_t sample_noise[8];
    uint32_t sample_cca;
    uint32_t sample_tx;
};
```

valid	Frames received with a valid FCS.
fcs	Frames received with an invalid FCS.
abort	Partially received frames.
phyabort	Frame receptions aborted based on the PHY header.
rts_success	The number of CTS frames is received in response to an RTS.
rts_fail	The number of RTS frames for which no response CTS frame was received.
timestamp	Timestamp at the time the snapshot was taken.
time_tx	Total time the LMAC transmitted.
noisefloor	The baseband's noisefloor reading at the moment the OID is requested.
sample_noise	Number of RSSI samples per power category.
sample_cca	Number of samples with CCA high.
sample_tx	Number of samples during which the LMAC transmitted.

Access

Read

Description

LMAC statistics.

2.4.5 Hardware

- **LED behaviour**

Object

LM_OID_LED

```
struct s_lmo_led
{
    uint16_t flags;
    uint16_t mask[2];
    uint16_t delay/*[2]*/;
};
```

flags	Bitmask that specifies whether a LED is derived from its hardware function (0) or by software (1).
mask	Array of LED bitmasks.
delay	Delays in kμsec between setting mask[0] and mask[1] (delay[0]), and switching mask[1] to mask[0] (delay[1]). Use delay[1] = 0 to for single-shot changes.

Access

Write

Description

Each supported LED is represented by a bit in flags member and the two mask members. The flags member specifies whether the LED is derived from its hardware function, or set by the software mask. The software masks specify the state of LED initially (mask[0]), and after the first delay expired (mask[1]). The optional second delay specifies the time span after which the LED must return to their initial state (mask[0]) and repeat the pattern.

3 Interfaces & mechanisms

3.1 Object management

All operations on LMAC objects by the application are headed by an `s_lm_control` header. The object data directly trails the `s_lm_control` header. Objects can be of access Read, Write, Read-Write or Trap. The application is responsible for maintaining range, bounds and length constraints.
Write operations are identified by a set LM_CTRL_OPSET flag. The LMAC does not generate a response for write operations.
Read operations are identified by a cleared LM_CTRL_OPSET flag. The data part of the control structure must be large enough to hold the result of the read operation. The LMAC generates a response for read operations, with the same message type as the request.
Trap operations are initiated by the LMAC by definition. A trap can be distinguished from a Read response because the LM_CTRL_OPSET flag is set[2].
Messages with application initiated operations (Read, Write and Read-Write) must be addressed within the application managed buffer. The `address` member in the response message can be used to distinguish between several outstanding operations on the same object. LMAC initiated operations may originate from anywhere, consequently the `address` member of the message is reserved.

3.2 LMAC and BSS setup and synchronization

The **Setup** object is used to setup the LMAC and setup or join a BSS. The object configures LMAC properties like the LMAC's MAC address, and receive buffer setup and BSS settings, like the BSS's BSSID (for frame filtering purposes), frequency of the BSS and which rates must be used as basic rates.

3.2.1 Initial setup

The initial set of the **Setup** object is used to configure the receive buffer and receive MTU. The `rx_buffer` structure member configures the address of the receive buffer within the application managed buffer. The receive buffer occupies the top part of the application managed buffer, starting at the address in `rx_buffer` and ending at the top of the application managed buffer. The `rx_mtu` member specifies the MTU the LMAC must be able to receive. From this data, the LMAC sets up it's receive queue.
In subsequent writes to the **Setup** object, the `rx_buffer` and `rx_mtu` members are ignored.

3.2.2 Setup

The **Setup** object configures the LMAC's mode (client or access point) and BSS type (infrastructure or IBSS), along with BSS some properties like basic rates.
Before a setup is done, the LMAC must be configured at the correct frequency by means of the **Scan** object, with the LM_SCAN_EXIT flag set.
If an IBSS is setup or joined, or an infrastructure is setup in access point mode, the application must submit a fully formatted beacon frame from which the LMAC derives the BSS's parameters like beacon period, DTIM period; etc… in the LMAC's LM_QUEUE_BEACON queue after the **Setup** object is set.
New beacons may be submitted during the lifetime of the BSS, without cancelling the previous beacon. The LMAC returns the previous beacon through the frame feedback mechanism and update the BSS settings according to the settings in the beacon. If the LMAC has a beacon and the setup is changed to a configuration that does not

[2] Note that the access type also disguises a Read response from a Trap; there are no objects with access Read and Trap.

need a beacon, the beacon must be cancelled. The beacon is then returned to the application through the feedback mechanism so the application can free the resource in its buffer management.
After the beacon is submitted, the BSS TSF timer is started and the Beacon is repeated at every TBTT, where the LMAC takes care of adapting dynamic elements like the Timestamp, DTIM count, TIM, etc…

3.3 Frame transmission

3.3.1 Transmission

All frames transmitted by the application to the LMAC contain an `s_lm_data` header, with an embedded `s_lm_data_out` header. The frame data directly trails the `s_lm_data` header. Frames are referenced by the LMAC memory address the frame is transferred to.

3.3.2 Feedback

Upon successful or failed transmission of a frame by the LMAC, the LMAC traps a **Tx feedback** object to the application. The object is transferred over the same channel as frames are.
Once the trap has been received, the application can assume that the frame and its associated memory are not referenced by the LMAC anymore.
If the frame is part of a set of fragmented frames, the remaining fragments must be cancelled by the application.

3.3.3 Cancellation

Once frames have been submitted to the LMAC for transmission, the application may cancel frames through the **Tx Cancel** object, for example because of transmission lifetime expiration. The object is transferred over the same channel as frames are.
The LMAC traps a **Tx feedback** object upon cancellation of the frame. The LM_TX_FAILED flag is set, if the frame was successfully cancelled. A regular feedback object without the LM_TX_FAILED flag is trapped, if the frame could not be cancelled because it was already successfully sent. A frame that was exhaustively retried (which also has the LM_TX_FAILED flag set) can be distinguished from a frame that was cancelled by comparing the number of actual retries in the **Tx feedback** object to the number of retries the frame was supposed to have.
Note that for multicast and broadcasts the LM_TX_FAILED flags are also well defined; if a group addressed frame is successfully cancelled the LM_TX_FAILED flag is set. If a group addressed frames could not be cancelled and was sent, LM_TX_FAILED is cleared.

3.3.4 Bursting

The bursting mechanism is based on the application providing the LMAC with information about the duration of pending frames. This can either be through the duration member of the `s_lm_data_out` structure, or by setting the **Burst update** object.
The piggy-backed method must be used if the application can send frames to the LMAC. The burst update method can be used if the application cannot update the burst-size because the driver does not allow transferring frame messages, if no frames are available to piggy-back the information on, or if an explicit update is necessary.

3.3.5 Encryption

The LMAC may implement hardware acceleration for the at least the encryption part of the privacy mechanism for frame transmission. The SoftMAC must implement the protocol functions. The `crypt_offset` member in the `s_lm_data_out` structure specifies the offset at which the encryption engine must start crypting, using the final key as specified in the same structure. The encryption engine crypts until the end of the frames, possibly extending the frame with an ICV.

3.4 Frame reception

All frames received by the LMAC are transferred to the application by means of an `s_lm_data` header, with an embedded `s_lm_data_in` header. The frame data directly trails the `s_lm_data` header.

3.4.1 Frame matching

The LMAC determines for each received frame the following properties;

- Frame Check Sequence success/failure (LM_IN_FCS_MATCH),
- Address1 matching (LM_IN_MATCH_MAC, LM_IN_MCBC),
- BSS matching (LM_IN_MATCH_BSS, LM_IN_BCAST_BSS),
- Type matching (LM_IN_DATA).

LM_IN_FCS_MATCH is set if the frame's check sequence (FCS) is successful. This is an indication that the frame was received without bit errors.
LM_IN_MATCH_MAC is set if address1 matches the LMAC's MAC address. LM_IN_MCBC is set if address1's group address bit is set.
LM_IN_MATCH_BSS is set by comparing either address2 or address3 to the configured BSSID, depending on whether the FromDS bit in the control field is set (address2) or not (address3). The caveat is that LM_IN_MATCH_BSS is not correct for the following cases;

- PS-Poll frames or frames with the ToDS bits set, with the BSSID in address1. These frames are always addressed to an access point, where address1 must match the access point's MAC address.
- CF-End and CF-End+CF-Ack frames.
- Control frames. These frames may have spurious matches because the match is performed, even if there is no address3.

LM_IN_BCAST_BSS is set if the group address bit of address2 or address3 (depending on the FromDS bit) is set.
LM_IN_DATA is set if the frame is of type data, and is not of sub-type Null.

3.4.2 Decryption

The LMAC may implement hardware acceleration for privacy decrypt. The LMAC at least implements those functions necessary to determine the type of encryption, the offset at which the decryption must start and what key must be used for decryption. The LMAC also implements a SoftMAC managed key-cache to get its keys from. The LMAC may fail to decrypt a frame because of a miss in the key-cache. In this case, the LMAC returns LM_IN_NOKEY. If the LMAC finds a key and decrypts the frame, but the Integrity Check Sequence (ICV) fails because it uses the wrong key, the decrypted frame is returned with status LM_IN_DECFAIL. Failure to decrypt because of unknown or unsupported privacy algorithm returns the encrypted frame with status LM_IN_DECALG.

3.5 PSM

3.5.1 Client mode

- **Infrastructure mode**

The application switches between PSM and CAM (Continuous Awake Mode) modes by setting **PSM** object to indicate desired power save state, by setting or clearing the LM_PSM flag.
If the desired state is PSM, the LMAC sets Power Management bit in the control field on all outgoing frames (this is done when frame is actually transmitted, not earlier).
If desired state is CAM (Continuous Awake Mode), the LMAC doesn't update the control field's Power Management bit (the bit may still be set by application, or left on by earlier transmit attempt).
The LM_TX_PSM bit in the flags member of the **Tx feedback** object indicates the state of the control field's Power Management bit in the acked frame.
To change the PSM state when the transmit queues are empty, the application is responsible for sending a null frame. If the transmit queue is not empty, the application may decide to only use the PSM object to change to PSM mode, and wait for feedback frame. If, due to race conditions, the frame was transmitted with Power Management bit cleared, the application is responsible for sending an additional null frame.
If the desired state is PSM, and a frame with the Power Management bit set was successfully acknowledged and the transmit queues are empty, the LMAC enters PSM.
The **PSM** object's `interval` member specifies the maximum number of beacon intervals for which the LMAC powers down if no data is available for transmission or retrieval. The Listen Interval is synchronized to the access point's DTIM interval.
If the **PSM** object's LM_PSM_MCBC flag is set, and a beacon is received with the TIM's multicast traffic bit set, the LMAC will stay awake until all group addressed frames have been received (e.g. until the LMAC receives a group addressed frame with the control field's More Data bit cleared, or until the next TIM's multicast traffic bit is cleared).
The `aid` structure member of the **PSM** object configures the AID the LMAC must monitor. If the TIM indicates unicast traffic for the given AID, LMAC will stay awake until unicast data is received with control field's More Data bit cleared, or until next TIM.

- **Beacon filtering**

To prevent passing every received beacon to the application (which may have a considerable impact on the application's host power efficiency), the application can specify which beacons the LMAC must pass to the application.
The LM_PSM_CHECKSUM flag configures the LMAC to calculate a checksum over the beacon frame body. The checksum is calculated over the frame-body, starting after the timestamp element. Excluded from the checksum calculation are all flexible elements, with a corresponding element ID in the `exclude` array structure member.
The beacon is forwarded to the application, if the checksum changes from the previous received beacon.
The **Setup** object also passes a `timeout` member that specifies the maximum period of time between Beacons. If within that period no beacon is received, the LM_TRAP_NO_BEACON trap is generated. The application can use this to decide that the connection with the AP is lost. The LMAC resets the timeout for every beacon it receives from the BSS it is joined with. The LMAC also resets the timeout when the application starts or continues a scan.

3.6 Scanning

Scanning is initiated by setting the **Scan** object. The **Scan** object causes the LMAC to stop transmitting from its queues and change frequency. The LMAC exits from scanning mode and returns to the given frequency if the LM_SCAN_EXIT flag is set. If the LM_SCAN_EXIT flag is cleared, the LMAC remains in scanning mode until the **Scan** object is set with the LM_SCAN_EXIT flag set. A scan can also be aborted, or the application can recover from a lingering scan mode by setting the dwell time to 0 and setting the LM_SCAN_EXIT flag.
If the LM_SCAN_TRAP flag is set, the LMAC traps a generic trap with number LM_TRAP_SCAN upon expiry of the dwell time (as specified by the `dwell` structure member). A dwell time of 0 causes an immediate trap.
If the LM_SCAN_ACTIVE flag is set, the LMAC generates a probe request on the specified frequency, according to the rules specified for active scanning.

3.6.1 Probe requests

The probe request for active scanning must be preformatted by the application and sent to the LMAC to a special scan queue. The probe request remains in the queue for active scanning until it is specifically cancelled by the application.

3.7 Filtering

The LMAC provides an option of enabling and disabling Multicast and ARP filtering.

3.7.1 Multicast Filtering

Multicast filtering is used to filter out the unnecessary multicast traffic based on the Destination mac address. Whenever the multicast packet is received by the device it checks whether the destination mac address matches with any of the multicast addresses configured on the device. If this condition holds valid, it implies the packet belongs to the device multicast group and hence it's forwarded to the host and in case of mismatch it's dropped. Since the wireless LAN is the broadcast medium, multicast filtering minimizes the flooding of the multicast packets to the host which reduces the overall power consumption of the device and hence increases the standby time.

Object

LM_OID_GROUP_ADDRESS_TABLE

Structure

```
#define MC_FILTER_ADDRESS_NUM   4

Struct s_lmo_group_address_table
{
   uint16_t   filter_enable;
   uint16_t   num_address;
   uint8_t    macaddr_list[MC_FILTER_ADDRESS_NUM][6];
};
```

filter_enable	Enable or disable the Multicast filtering. 1:- Multicast filtering enabled. 0:- Multicast filtering disabled.
Num_address	Number of multicast groups subscribed by the device.
macaddr_list	Multicast mac address of the groups subscribed.

3.7.2 ARP Filtering

Arp filtering is used to filter out the ARP request broadcast packets to reduce the interrupt activity on host side and reduces the overall power consumption of the device. Using ARP filtering a fixed IP address can be configured in the device and all ARP requests packets corresponding to the configured IP address will be passed to the host rest will be dropped.

Object

LM_OID_ARPTABLE

Structure

```
Struct s_lmo_arp_table
{
   uint16_t   filter_enable;
   uint32_t   ipaddr;
};
```

filter_enable	Enable or disable the ARP filtering. 1:- ARP filtering enabled. 0:- ARP filtering disabled.
ipaddr	IP address for which ARP requests packets needs to be sent to host.

3.8 Statistics

The **LMAC Statistics** object provides the application access to LMAC statistics on medium state, radio performance and timing aspects. All statistics are cumulative; periodical properties can be derived by storing a previous result of a query and subtracting individual members. The exact measurement period can be derived from the `timestamp` member.

3.8.1 Noise histogram

The LMAC samples RSSI, CCA and transmit state at regular periods (typically 8 times per 1 kμsec). The result of RSSI samples (in other words the samples that were taken in a non-CCA and non-transmit state) are categorized in the following ranges;

RSSI category	Power observed at Antenna (dBm)
0	RSSI ≤ -87
1	-87 < RSSI ≤ -82
2	-82 < RSSI ≤ -77
3	-77 < RSSI ≤ -72
4	-72 < RSSI ≤ -67
5	-67 < RSSI ≤ -62
6	-62 < RSSI ≤ -57
7	-57 < RSSI

The total number of samples can be derived by adding the delta of all categories of RSSI samples, CCA samples and transmit state samples.

Appendix

N.A

References

LMAC API header file (lmac_longbow.h)