Software Implementation Design

for

Integrated **T**DMA **E**-ASAP **M**odule (ITEM)

Version 2.0

Authored by Pavel Beneš

Czech Technical University

November 11, 2010

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Document Conventions

Following conventions are used throughout this document:

Italic	is used for directories and filenames, and to emphasise new terms and concepts when they are introduced. Italic is also used to highlight comments in examples.
Bold	is used for nesC keywords.
`Constant width`	is used in text for code examples, elements of a program and to show the contents of files or the output from commands. A reference in text to a word or item used in an example or code fragment is also shown in constant-width font.
`Constant width`	is used in examples to show commands or other text that should be typed literally by the user. (For example, **`make telosb`** instructs you to type "make telosb" exactly as it appears in the text or example.)
`Constant Italic`	is used in examples to show variables for which a context-specific substitution should be made. (The variable *`filename`*, for example, would be replaced by some actual filename.)
“”	are used to identify system messages or code fragments in explanatory text.
`%`	is the UNIX shell prompt.
`[]`	surround optional values in a description of program syntax. (The brackets themselves should never be used.

. . .	stands for text (usually computer output) that's been omitted for clarity or to save space.

1.2 Intended Audience and Reading Suggestions

This document is intended to give a guidance for further development or improvement related to ITEM. It is assumed that the reader has a good knowledge of and experience in TinyOS 2.x and nesC language. It is also assumed that the reader has the knowledge of the article *An Adaptive TDMA Slot Assignment Protocol in Ad Hoc Sensor Networks.*

1.3 Project Scope

The scope of this project was to realise the theoretical functionality suggested by the article *An Adaptive TDMA Slot Assignment Protocol in Ad Hoc Sensor Networks[2]* and related issues concerning distributed systems.

1.4 References

[1] Philip Levis: "TinyOS Programming" at http://csl.stanford.edu/~pal/pubs/tinyos-programming.pdf (June 28, 2005).

[2] Akimitsu Kanzaki, Takahiro Hara and Shojiro Nisho: "An adaptive TDMA slot assignment protocol in Ad Hoc Sensor Networks" in Proceedings of the 2005 ACM symposium on Applied computing, Pages: 1160 - 1165 (2005).

2. Interface

2.1 Hardware Interface

The logical layout of the system is recommended as following. One node should be connected physically to the serial USB port of the computer and act as a Base station, set through the interface (see chapter 3. *ITEM API*), and the remaining nodes are then set in 'normal' mode, as shown in Figure 2.1.1.

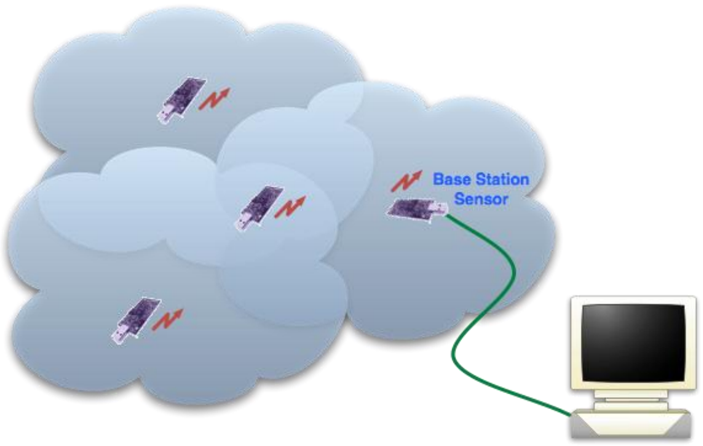


Figure 2.1.1 - Hardware layout of the recommended system.

Once the system is up and running, the output from the Base Station can be monitored with the generated serial output with the following structure:

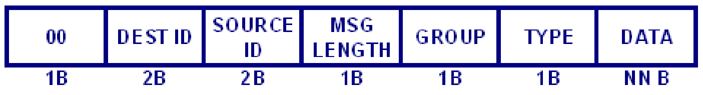


Figure 2.1.2 - Serial output given by the Base Station (Tmote Sky).

Output meaning:

00	– Leading byte
DEST ID	– Destination address
SOURCE ID	– Source address
MSG LENGTH	– DATA length (*nn* bytes)
GROUP	– Message group
TYPE	– Message type
DATA	– Data

! Please note that this is a specific output given by the TelosB/Tmote Sky and can vary from depending on the sensor type used. However, this document is based on the system test of the TelosB/Tmote Sky sensor type and therefore only covers this technology.

The leading framing byte, “`0x00`”, defined by TinyOS, marks the complete packet of the data that is sent. The sixth byte contains the length of the data (in bytes) that the packet is holding. And finally the data itself, sent by the ITEM, starts from the 9th byte with the following stream format:

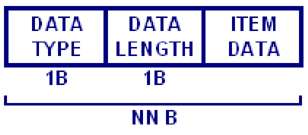


Figure 2.1.3 - ITEM data stream format.

Output meaning:

DATA TYPE	– Data type
DATA LENGTH	– ITEM DATA length ((*nn*-2) bytes)
ITEM DATA	– Item data

The data type can either be an *INF* (Information) package or a *DAT* (Data) package and is defined with the hexadecimal value “`0xC1`” and “`0xC2`” respectively.

2.1.1 Information (INF) Package

The INF package has the following structure:

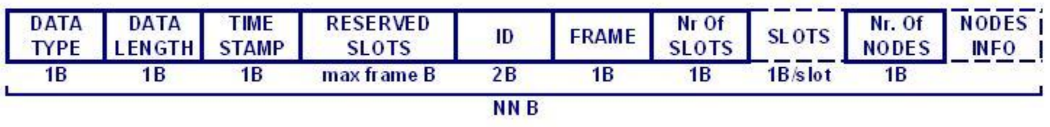


Figure 2.1.1.1 - INF package format.

Output meaning:

DATA TYPE	– Data type (0xC1 = information packet)
DATA LENGTH	– ITEM DATA length ((*nn-2*) bytes)
TIME STAMP	– Message time stamp
RESERVED SLOTS	– Which slots cannot be used (slot 0 = first bit, 0 = reserved, 1 = free)
ID	– Node address
FRAME	– Node frame
Nr Of SLOTS	– Number of node slots
SLOTS	– Node slots
Nr Of NODES	– Number of nodes in NODES INFO
NODES INFO	– Other nodes ifno

The "Nodes Info" is packaged in the following format for EACH node:

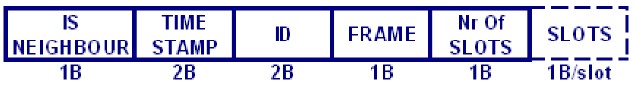


Figure 2.1.1.2 - "Nodes Info" format.

Output meaning:

IS NEIGHBOUR	– Describe, if the node is neighbouring node
TIME STAMP	– Node last time stamp
ID	– Node address
FRAME	– Node frame
Nr Of SLOTS	– Number of node slots
SLOTS	– Node slots

The first byte in the "Nodes Info" describes if the node is a neighbouring node or a hidden node to the Base Station. The byte is interpreted as `0x01` for the neighbouring node and `0x00` for the hidden node.

2.1.2 Data (DAT) Package

The DAT package has the following structure:

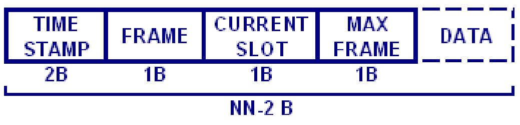


Figure 2.1.2.1 - DAT package format

Output meaning:

TIME STAMP	– Node last time stamp
FRAME	– Node frame
CURRENT SLOT	– Current slot
MAX FRAME	– Max frame size
DATA	– Data

!

The “Time Stamp” is originally set as a **32** bits long unsigned integer value. Bits 1 and 2 are extracted in these packages when it is sent INF packet to the serial port. This gives a time range between 2^8-1 to 2^{24}-1 milliseconds and an accuracy of 255 milliseconds. The reason for the extraction is for minimising the data flow to the serial communication. However, the complete “Time Stamp” is sent when broadcasted within the network.

2.2 Software Implementation

The complete system is designed for the aim of flexibility of modules where each module has it’s own function, as shown in Figure 2.2.1.

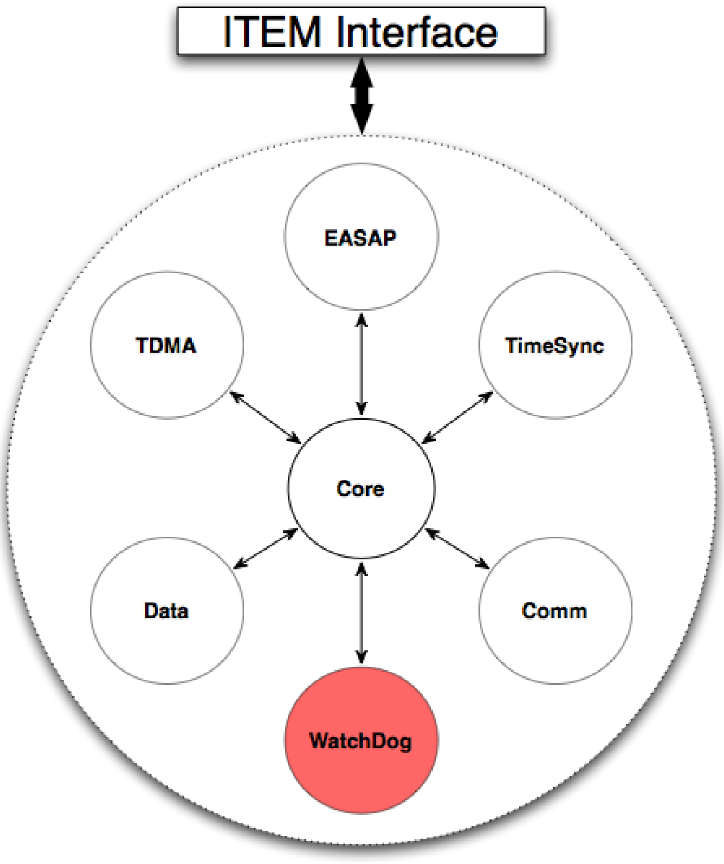


Figure 2.2.1 - ITEM Layout.

The abstraction for the usage of the system is given through the ITEM API interface described in chapter *3. ITEM API*. This interface is provided by the **ITEMC** component and needs to be wired appropriately.

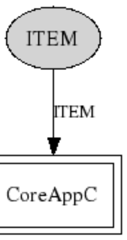


Figure 2.2.2 - Component ITEMC 'wired' to CoreC.

Once 'wired', all the functionality to ITEM is the provided by the ITEM API.

2.3 Modules

Here, all the modules are described in details individually. All modules are implemented as independent components. However, this does **not** apply to the Core where its functionality is to combine all the other components and uses them appropriately.

2.3.1 E-ASAP

2.3.1.1 Wiring

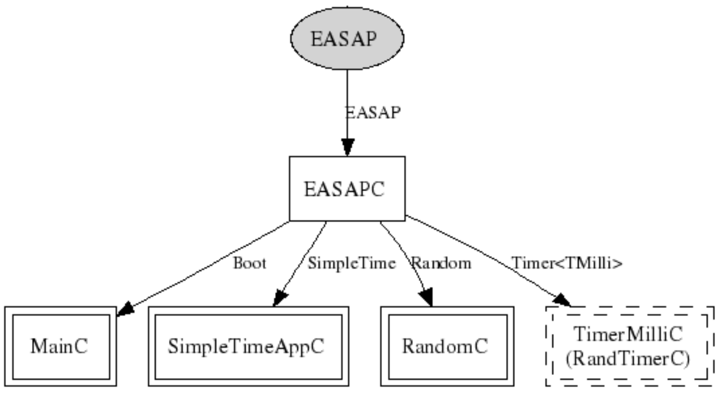


Figure 2.3.1.1.1 Component EASAPC's wiring.

2.3.1.2 Description

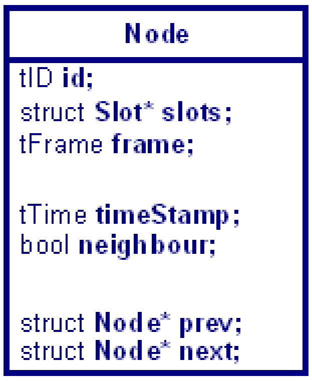
Extended Adaptive TDMA Slot Assignment Protocol's (E-ASAP) behaviour is explained in the article [2] and follows its recommendation. Please refer to the article for the complete description of the E-ASAP protocol. Here, only the implementation is described.

2.3.1.3 Structures

For holding the information of each node within the contention area[1], **Node** structure is defined. This structure is used as a link list, to hold INF, connecting both directions. However, the **first** element in the element is always containing nodes own information.

[1] Contention area - Nodes present within maximum one hop away from the current node.

2.3.1.3.1 INF structure



```
Node
tID id;
struct Slot* slots;
tFrame frame;

tTime timeStamp;
bool neighbour;

struct Node* prev;
struct Node* next;
```

Figure 2.3.1.3.1.1 - Node structure

Using this structure, INF can look like this when in operation:

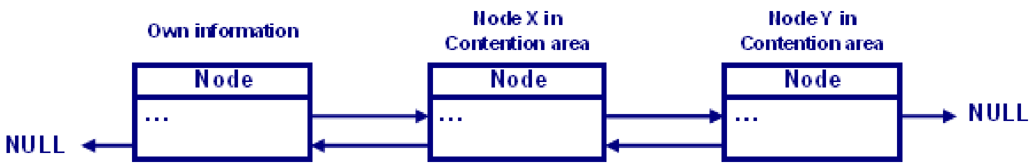
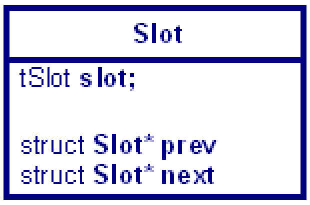


Figure 2.3.1.3.1.2 - INF example

Extending the protocol, each node can have multiple slots. In the same fashion as structure **Node**, the **Slot** structure is used as a link list containing slot number and connecting in both directions as shown in Figure 2.3.1.3.1.2.



```
Slot
tSlot slot;

struct Slot* prev
struct Slot* next
```

Figure 2.3.1.3.1.3 - Slot structure

2.3.1.4 Functionality

As described by the article, the EASAP module handles the network information held by each node and updates this information appropriately. Each part of the implementation is described for the complete understanding of the implementation.

2.3.1.4.1 Getting a new slot

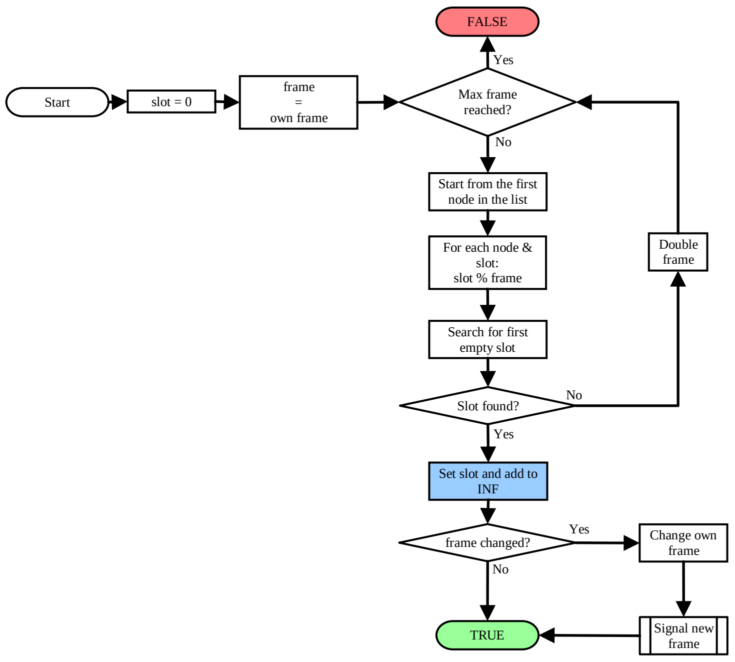


Figure 2.3.1.4.1.1 - Procedure for requesting a new slot.

2.3.1.4.2 Updating node's information

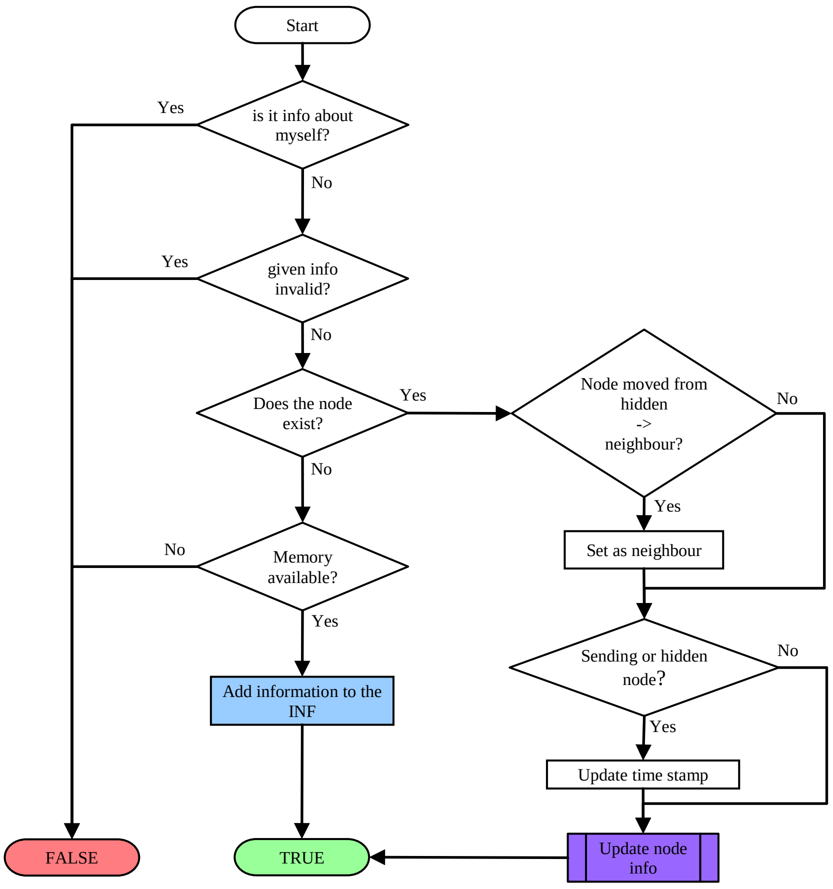


Figure 2.3.1.4.2.1 - Updating received node information sent by neighbouring node

Defined process “Update Node Info” is defined as shown in figure 2.3.1.4.2.2.

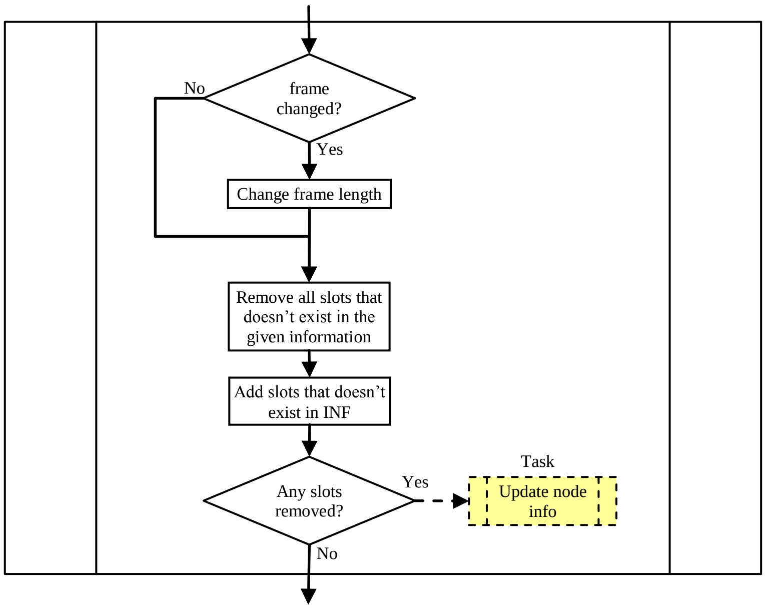

frame changed?
No
Yes
Change frame length
Remove all slots that doesn’t exist in the given information
Add slots that doesn’t exist in INF
Any slots removed?
Yes
No
Task
Update node info

Figure 2.3.1.4.2.2 - Update Node Info

2.3.1.4.3 Removing inactive nodes

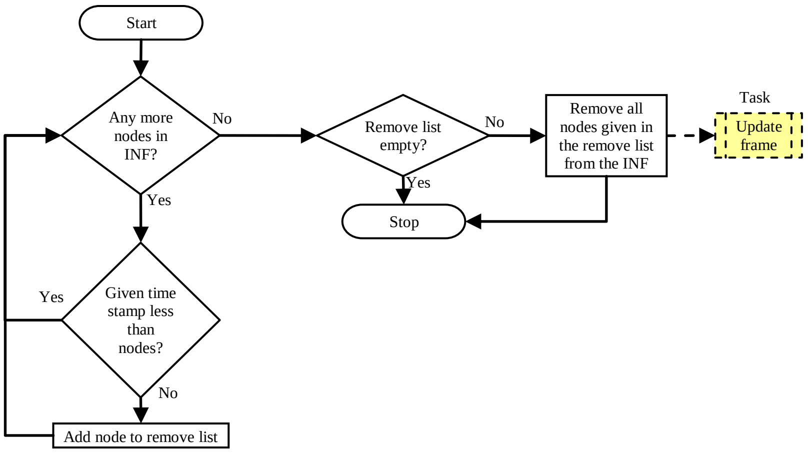


Figure 2.3.1.4.3.1 - Removing all inactive nodes procedure

2.3.1.4.4 Updating frame length

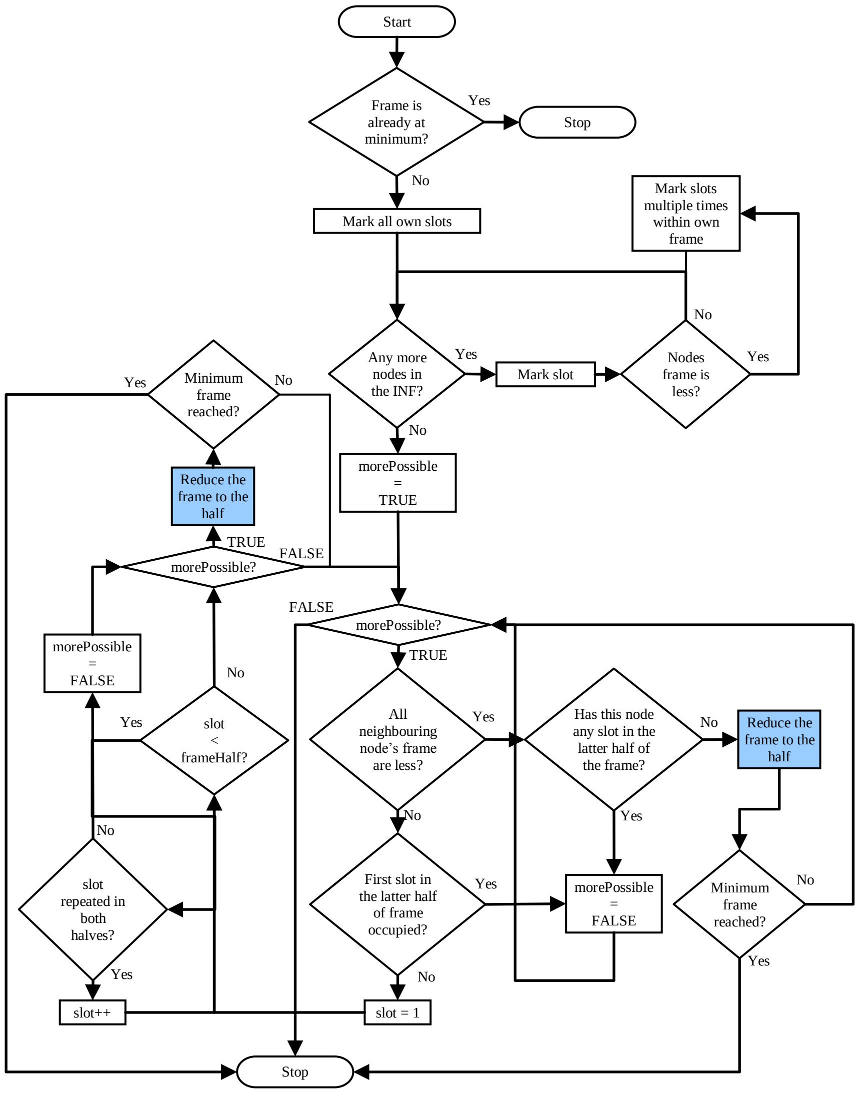

Start
Frame is already at minimum?
Yes
Stop
No
Mark all own slots
Mark slots multiple times within own frame
No
Any more nodes in the INF?
Yes
Mark slot
Nodes frame is less?
Yes
No
morePossible = TRUE
Minimum frame reached?
Yes
No
Reduce the frame to the half
TRUE
morePossible?
FALSE
FALSE
morePossible?
TRUE
morePossible = FALSE
slot < frameHalf?
Yes
No
All neighbouring node's frame are less?
Yes
No
Has this node any slot in the latter half of the frame?
No
Yes
Reduce the frame to the half
slot repeated in both halves?
No
Yes
First slot in the latter half of frame occupied?
Yes
No
morePossible = FALSE
Minimum frame reached?
No
Yes
slot++
slot = 1
Stop

Figure 2.3.1.4.4.1 - Updating frame procedure

Please note that this procedure reduces frame length multiple times so far as it is possible. Example, when many nodes leave, *at the same time*, the network, say the frame length is 16, it will reduce the frame to 4 straight away as long as any of the requirements, given by the article, is fulfilled.

2.3.1.5 API

2.3.1.5.1 Commands

`2.3.1.5.1.1 sigSlotZero`

```
void sigSlotZero(void)
```

Increases the frame length.

2.3.1.5.1.2 getINF

`struct Node* getINF(void)`

Gives the reference to the INF list.

Returns:
Returns the reference to the INF structure (*`EASAP.h`*).

2.3.1.5.1.3 getDAT

`struct DAT* getDAT(void)`

Gives the DAT info EXCEPT the currentSlot.

Returns:
Reference to the `DAT` structure as defined in *`EASAP.h`*.

2.3.1.5.1.4 getSlot

`bool getSlot(uint8_t *slotNr)`

Gets a slot and assigns it to itself, if possible.

Parameters:
`slotNr` - Slot number is set if successful. `0` if not found.

Returns:
`TRUE` if found. `FALSE` if not (Max frame length has been reached).

2.3.1.5.1.5 isOwnSlot

`bool isOwnSlot(uint8_t slot)`

Acknowledges if the given slot belongs to the node or not.

Parameters:
slot - Slot to check if it belongs to this node.

Returns:
`TRUE` if the given slot belongs to this node. Else, `FALSE`.

2.3.1.5.1.6 removeSlot

`bool removeSlot(uint8_t slot)`

Removes own slot from the list, if it exists.

Parameters:
`slot` - Slot to be removed.

Returns:
`TRUE` if removed. `FALSE` if not found.

2.3.1.5.1.7 getMaxFrameLength

```
uint8_t getMaxFrameLength(void)
```

Return the current set maximum frame length within the contention area.

Returns:
Maximum frame length that exist within the contention area.

2.3.1.5.1.8 updateNode

```
bool updateNode(uint16_t id, uint8_t *slots,
                uint8_t frameLength, tTime
                *cLocalTime, bool isNeighbour)
```

Updates the given node's information to the INF list appropriately as defined in E-ASAP protocol. If the node doesn't exist in the list, it is added as new.

Parameters:
`id` - ID of the node.

`pSlots` - Slots assigned to the node. Initial byte must contain the number of slots that are in the stream

`frameLength` - Frame assigned to the node.

`cLocalTime`. - Current local Time.

`isNeighbour` - Is it a neighbour to the current node (Broadcasting node)

Returns:
`TRUE` if added/updated.

`FALSE` if:

- it already exists.
- no memory was available.
- any of id/slot/frame is invalid.

2.3.1.5.1.9 removeNode

```
void removeNode(uint16_t id)
```

Removes the node from the INF list.

Parameters:
`id` - Node to be removed. Discarded if it doesn't exist.

2.3.1.5.1.10 removeInactiveNodes

```
void removeInactiveNodes(tTime *timeStampMark)
```

Removes all inactive nodes from the INF list. Inactive node is given by all nodes that have older time mark than the given argument.

Parameters:
`tStampMark` - Time mark. Oldest allowed time mark for all nodes.

2.3.1.5.1.11 findNode

`struct Node* findNode(uint16_t id)`
Searches for a node by its given id.

Parameters:
`id` - ID of the node.

Returns:
Reference to the found node. NULL if not found.

2.3.1.5.2 Events

2.3.1.5.2.1 frameChanged

`void frameChanged(uint8_t newframe)`
Signals when frame the length has been changed.

Parameters:
`newFrame` - Length of the new frame.

2.3.2 TDMA

2.3.2.1 Wiring

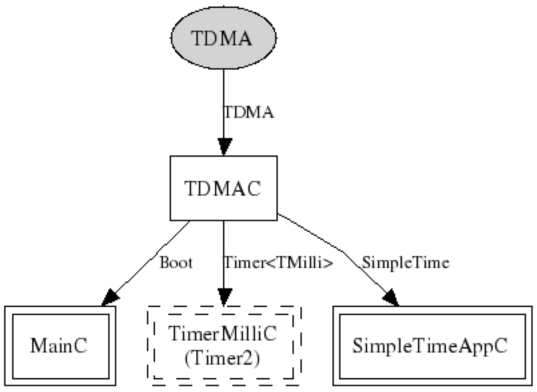


Figure 2.3.2.1.1 - Component TDMAC's wiring.

2.3.2.2 Description

Time Division Multiple Access provides the mechanism where slots, assigned by a fixed interval, are repeated within a frame. A frame can be increased/decreased by requested whenever appropriate.

2.3.2.3 Functionality

2.3.2.3.1 New slot event

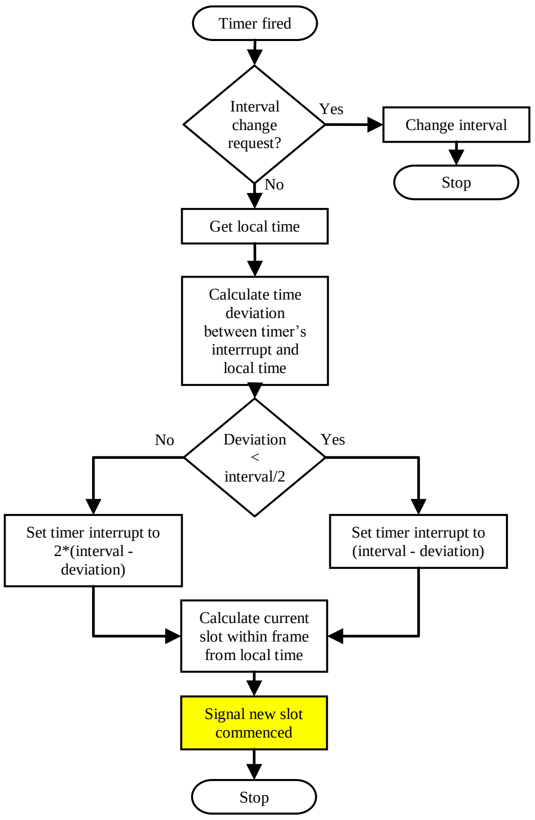

Timer fired
Interval change request?
Yes
Change interval
Stop
No
Get local time
Calculate time deviation between timer's interrrupt and local time
Deviation < interval/2
No
Yes
Set timer interrupt to 2*(interval - deviation)
Set timer interrupt to (interval - deviation)
Calculate current slot within frame from local time
Signal new slot commenced
Stop

Figure 2.3.2.3.1.1 - New slot event given when commenced.

2.3.2.4 API

2.3.2.4.1 Commands

2.3.2.4.1.1 start

```
error_t start(uint8_t sSlot, uint8_t sFrame,
                    uint16_t sInterval)
```

Starts the TDMA mechanism.

Parameters:

`sSlot` - Starting slot number. If the slot is bigger than `sFrame`, it is adjusted to a valid slot within `sFrame`.

`sFrame` - Frame length.

`sInterval` - Slot time length.

Returns:

`SUCCESS` if TDMA is started. Else,

- `TDMA_E_INVALID_INTERVAL` if the value of interval is not between `MIN_INTERVAL` and `MAX_INTERVAL` (*`TDMA.h`*).

- TDMA_E_INVALID_FRAME_LENGTH if the frame length is below MIN_FRAME_LENGTH (TDMA.).
- Error messages from the interface `Timer.start()`.

2.3.2.4.1.2 stop

`error_t stop(void)`

Stops the TDMA mechanism.

Returns:

`SUCCESS` if stopped. `FAIL` if failed to stop due to internal error.

2.3.2.4.1.3 setInterval

`error_t setInterval(uint16_t newInterval)`

Changes the interval length of a slot. The change of the interval will not be in affect before the next slot event.

Parameters:

`newInterval` - The new slot time in milliseconds.

Returns:

SUCCESS if changed. Else,

- `TDMA_E_INVALID_INTERVAL` if the value of interval is not between `MIN_INTERVAL` and `MAX_INTERVAL` (*`TDMA.h`*).
- Error messages from the interface `Timer.start()`.

2.3.2.4.1.4 getInterval

`uint16_t getInterval(void)`

Gives the current slot interval in milliseconds.

Returns:

Slot interval.

2.3.2.4.1.5 setFrame

`error_t setFrame(uint8_t newFrame)`

Changes the frame length.

Parameters:

`newFrame` - The new frame length.

Returns:

`SUCCESS` if changed. Else,

- `TDMA_E_INVALID_FRAME_LENGTH` if the frame length is below `MIN_FRAME_LENGTH` (*`TDMA.h`*).

2.3.2.4.1.6 getFrame

`uint8_t getFrame(void)`

Gives the current frame length.

Returns:
Current frame length.

2.3.2.4.1.7 adjust

`error_t adjust(int16_t msAdjust)`

Adjust the interval for **one** interval only. The adjustment of the interval will not be in affect before the next slot event.

Parameters:
`msAdjust` - Adjustment in milliseconds. Positive/Negative argument advances/regresses the interval.

Returns:
`SUCCESS` if it will be adjusted. Else,

- `TDAM_E_INVALID_ARGUMENT` if `msAdjust` is 0 or not between `MIN_ADJUST` (*`TDMA.h`*) and interval.

2.3.2.4.1.8 getLocalTime

`tTime getLocalTime(void)`

Gets the current local time.

Returns:
Current local time od type struct `tTime` (*`TDMA.h`*).

2.3.2.4.1.9 setLocalTime

`void setLocalTime(tTime t)`

Sets the local time to the given time argument.

Parameters:
`t` - Binary milliseconds of **struct** type `tTime` (*`TDMA.h`*).

2.3.2.4.2 Events

2.3.2.4.2.1 sigNewSlot

`void sigNewSlot(uint8_t slot)`

Signals when a new slot has commenced.

Parameters:
`slot` - current slot

2.3.3 Comm

2.3.3.1 Wiring

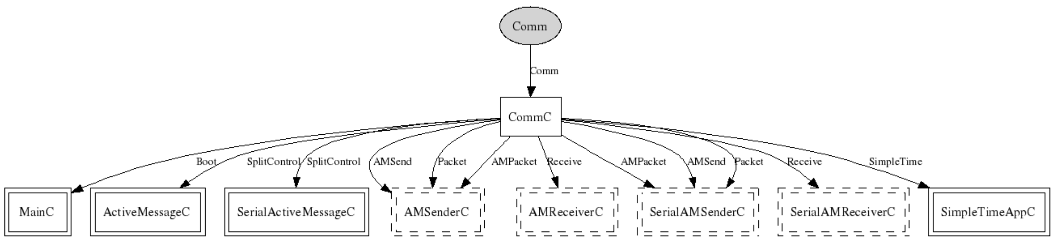
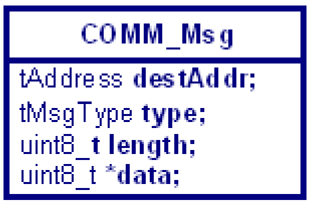


Figure 2.3.3.1.1 - Component CommC's wiring.

2.3.3.2 Description

Comm is responsible for all communication relating to other nodes via radio and computer via the serial. It has a basic functionality. Either it can be requested to transmit a message, encapsulated in `COMM_Msg`, or when a messaged is received, an event is signalled to all the components that are wired to it.

2.3.3.3 Structures



```
COMM_Msg
tAddress destAddr;
tMsgType type;
uint8_t length;
uint8_t *data;
```

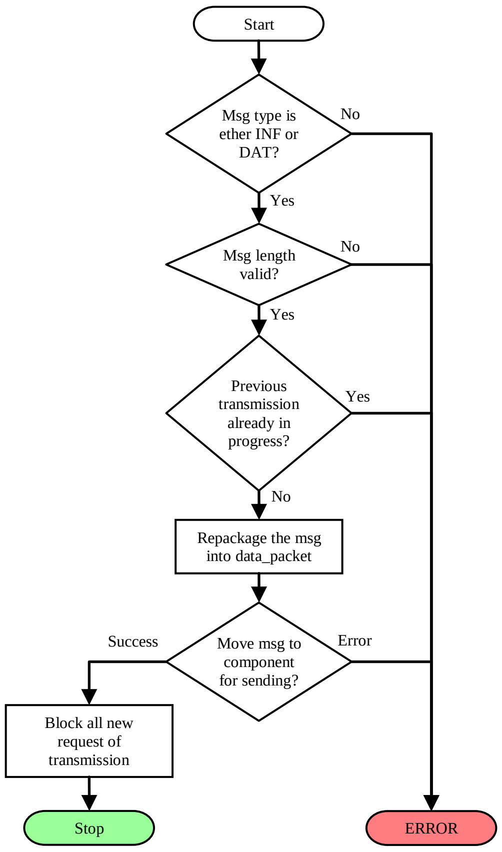
Figure 2.3.3.3.1 - Comm_Msg structure.

`destAddr` can have the range between 0 - 65535. The destination can be `COMM_SERIAL` for sending the data to the serial port, `COMM_BCAST` for sending the data to all nodes in the network or address of the destination node.

`type` can be either INF (information) or DAT (data) package. They are defined as `COMM_MSG_TYPE_INF` and `COMM_MSG_TYPE_DAT` respectively.

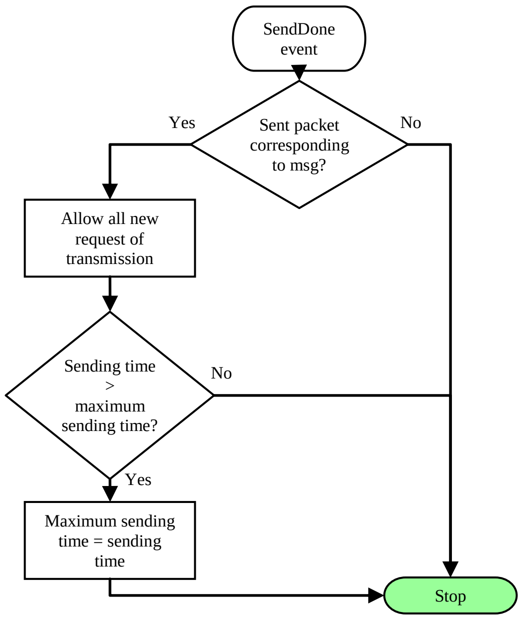
2.3.3.4 Functionality

2.3.3.4.1 Transmitting message


Start
Msg type is ether INF or DAT?
No
Yes
Msg length valid?
No
Yes
Previous transmission already in progress?
Yes
No
Repackage the msg into data_packet
Success
Move msg to component for sending?
Error
Block all new request of transmission
Stop
ERROR

2.3.3.4.1.1 - Transmission flow

2.3.3.4.2 Transmission done event


SendDone event
Yes
Sent packet corresponding to msg?
No
Allow all new request of transmission
Sending time > maximum sending time?
No
Yes
Maximum sending time = sending time
Stop

2.3.3.4.2.1 - Transmission done event

2.3.3.4.3 Receiving message

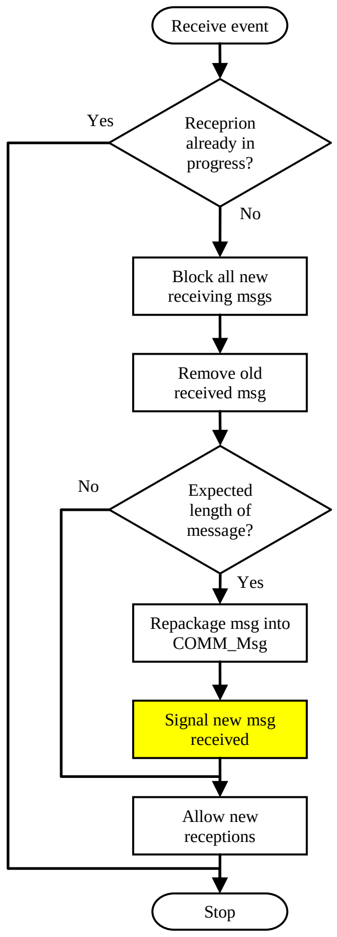

Receive event
Receprion already in progress?
Yes
No
Block all new receiving msgs
Remove old received msg
Expected length of message?
No
Yes
Repackage msg into COMM_Msg
Signal new msg received
Allow new receptions
Stop

Figure 2.3.3.4.3.1 - Reception flow

2.3.3.5 API

2.3.3.5.1 Commands

`2.3.3.5.1.1 transmit`

`error_t transmit(COMM_Msg *msg)`

Transmits the given message. The message can have following parameters.

Destination address:

- `COMM_SERIAL` for serial transmission.
- `COMM_BCAST` for broadcasting the message through RFM.
- `id` of the node to transmit to.

Message type:

- `COMM_MSG_TYPE_INF` for information packet.
- `COMM_MSG_TYPE_DAT` for data packet.

Parameters:
`msg` - Encapsulated message of **struct** `COMM_Msg` to be sent.

Returns:
`SUCCESS` if all went OK. Else,

- `COMM_E_INVALID_TYPE` Invalid type
- `COMM_E_INVALID_LENGTH` Invalid data length. Must be bigger than zero.
- `COMM_E_BUSY` Cannot transmit. Busy transmitting another message.
- `COMM_E_FAILED` Failed to transmit the message.

2.3.3.5.2 Events

2.3.3.5.2.1 received

`error_t received(COMM_Msg *msg)`
Signals when a message has been received.

Parameters:
`msg` - Received message. Defined in *`Comm.h`*.

Returns:
Always `SUCCESS`.

2.3.4 TimeSync

2.3.4.1 Wiring

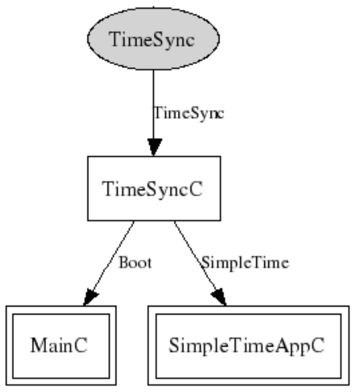


Figure 2.3.4.1.1 - Component TimeSyncC's wiring

2.3.4.2 Description

TimeSync used for the calculation of the time synchronisation between given two different times. It uses the simple algorithm:

$$t_{diff} = \frac{t_{local} + t_{remote}}{2} \qquad (2.3.4.2.1.1)$$

t_{diff} is limited between -32,768 to 32,767 milliseconds.

2.3.4.3 Functionality

2.3.4.3.1 Time difference calculation

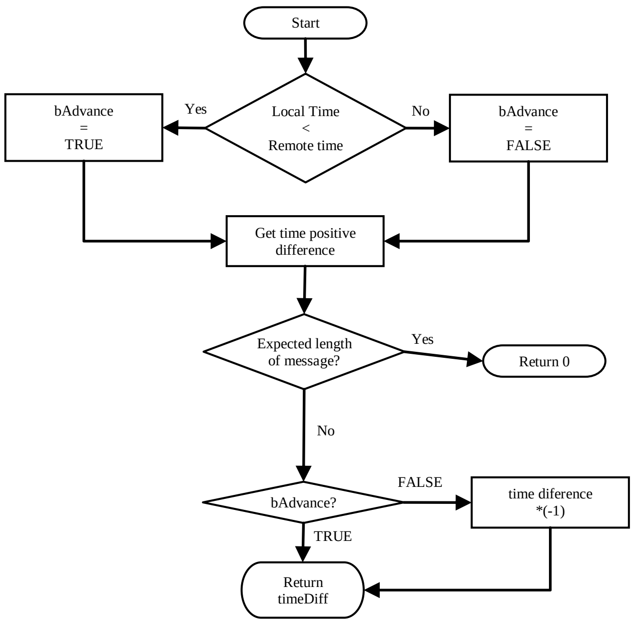


Figure 2.3.4.3.1.1 - Procedure when calculating time difference

2.3.4.4 API

2.3.4.4.1 Commands

2.3.4.4.1.1 transmit

`int16_t getAdjustment(tTime* lTime, tTime* rTime)`

Averages the time difference between given arguments and return the adjustment to be made for local time in milliseconds.

Parameters:

`lTime` – Current local time

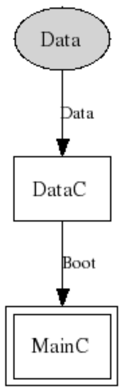
`rTime` – Current remote time

Returns:

Adjustment for local time in milliseconds.

2.3.5 Data

2.3.5.1 Wiring

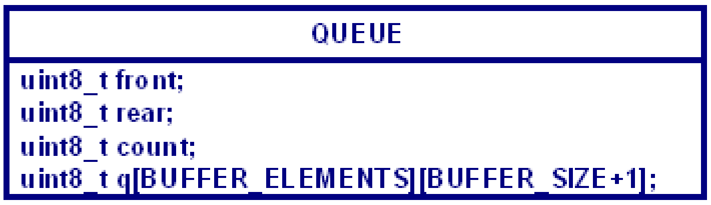


2.3.5.1.1 - Component DataC wiring

2.3.5.2 Description

Data module has two queues that holds the data for both receiving and transmitting data. Each data element is held by each queue element with the maximum size of `BUFFER_DATA_SIZE` and in FIFO order. If any of the queue is full, new data are discarded until an empty queue element is available.

2.3.5.3 Structures



```
QUEUE
uint8_t front;
uint8_t rear;
uint8_t count;
uint8_t q[BUFFER_ELEMENTS][BUFFER_SIZE+1];
```

Figure 2.3.5.3.1 - Queue structure for transmission and reception.

For each element, the first byte contain the data length followed by the data itself. The other variables, `front`, `rear` and `count`, are used internally to control the queues functionality.

2.3.5.4 Functionality

2.3.5.4.1 Setting data to the queue

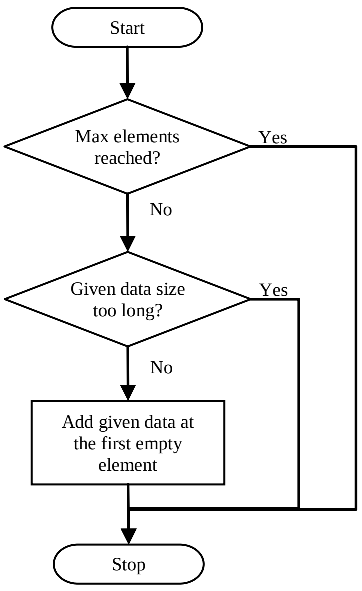

Start
Max elements reached?
Yes
No
Given data size too long?
Yes
No
Add given data at the first empty element
Stop

Figure 2.3.5.4.1.1 - Setting data to the queues

2.3.5.4.2 Retrieving data from the queue

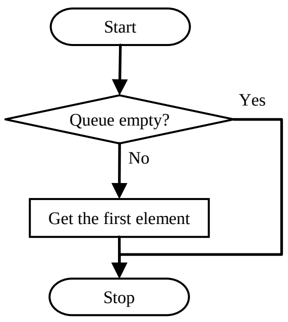


Figure 2.3.5.4.2.1 - Retrieving data from the queue.

2.3.5.5 API

2.3.5.5.1 Commands

2.3.5.5.1.1 setTXData

`error_t setTXData(uint8_t length, uint8_t *pData)`

Adds the given data into the transmission buffer if possible.

Parameters:

`length` - Length of the data to be added.

pData - Reference to the data to be added.

Returns:

`SUCCESS` If all went well. Else,

- `DATA_E_BUFFER_FULL` If the queue is full.
- `DATA_E_DATA_TOO_LONG` If the data length is bigger than the `BUFFER_DATA_SIZE`.

2.3.5.5.1.2 getTXData

`uint8_t getTXData(uint8_t **pData)`
Gets the first element from the transmission buffer (FIFO)

Parameters:
`pData` - Reference to the where data is to be added.

Returns:
Number of bytes that was received. 0 If the buffer is empty or there was no memory available.

2.3.5.5.1.3 setRXData

`error_t setRXData(uint8_t length, uint8_t *pData)`
Adds the given data into the reception buffer if possible.

Parameters:
`length` - Length of the data to be added.

`pData` - Reference to the data to be added.

Returns:
`SUCCESS` If all went well. Else,

- `DATA_E_BUFFER_FULL` If the queue is full.
- `DATA_E_DATA_TOO_LONG` If the data length is bigger than the `BUFFER_DATA_SIZE`.

2.3.5.5.1.4 getRXData

`uint8_t getRXData(uint8_t **pData)`
Gets the first element from the reception buffer (FIFO).

Parameters:
`pData` - Reference to the where data is to be added.

Returns:
Number of bytes that was received. 0 If the buffer is empty or there was no memory available.

2.3.5.5.1.5 TXDataEmpty

`bool TXDataEmpty(void)`
Checks if the TX data buffer is empty

Returns:
`TRUE` if it is empty. `FALSE` otherwise.

2.3.5.5.1.6 RXDataEmpty

`bool RXDataEmpty(void)`

Checks if the RX data buffer is empty.

Returns:
`TRUE` if it is empty. `FALSE` otherwise.

2.3.5.5.1.7 GetTXnrOfFreePlaces

`uint8_t getTXnrOfFreePlaces(void)`

Gets the count of free places in the transmission buffer (FIFO).

Returns:
The count of transmission buffer free places.

2.3.5.5.1.8 GetRXnrOfFreePlaces

`uint8_t getRXnrOfFreePlaces(void)`

Gets the count of free places in the reception buffer (FIFO).

Returns:
The count of reception buffer free places.

2.3.6 Watchdog

2.3.6.1 Wiring

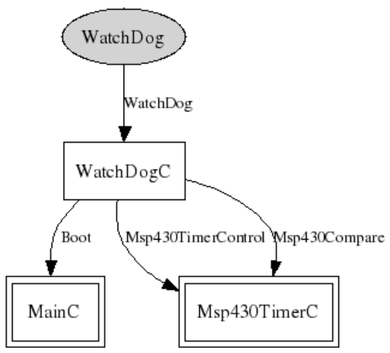


Figure 2.3.6.1.1 - Component WatchdogC's wiring.

2.3.6.2 Description

As any watchdog, its main function is to reset the system if it locks itself up due to an error. As default the reset time on lockup is set to twice the maximum frame length defined by EASAP or if not defined, 8 s.

It can be enabled/disabled by setting following definitions:

```
#define WATCH_DOG 1 //Enable
#define WATCH_DOG 0 //Disable
```

or in the Makefile defining the parameter as following for enabling it:

```
#WatchDog [ENABLE | DISABLE]
CFLAGS += -DWATCH_DOG=ENABLE
```

!

Please note that the watchdog only works for the **MSP430** architecture in this implementation.

2.3.6.3 API

2.3.6.3.1 Commands

2.3.6.3.1.1 start

`void start(void)`

Starts the watch dog monitoring. Once started, the function `touch()` must be called periodically for **not** getting the system restarted.

2.3.6.3.1.2 touch

`void touch(void)`

Makes sure to reset all counters. Must be called periodically.

2.3.7 SimpleTime

2.3.7.1 Wiring

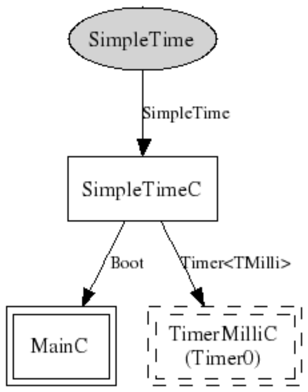


Figure 2.3.7.1.1 – Component SimpleTime's wiring.

2.3.7.2 Description

Time manipulations and calculations module.

2.3.7.3 API

2.3.7.3.1Commands

2.3.7.3.1.1 addUint32

`tos_time_t addUint32(tos_time_t a, uint32_t x)`
Add a unsigned 32 bits integer to a logical time.

Parameters:
`a` – Logical time.

`x` – An unsigned 32 bit integer. If it represent a time, the uint should be binary milliseconds.

Returns:
The new time in tos_time_t format.

2.3.7.3.1.2 addint32

`tos_time_t addint32(tos_time_t a, int32_t x)`
Adds a signed 32 bits integer to a logical time.

Parameters:
`a` – Logical time.

`x` – A 32 bit integer. If it represent a time, the int should be binary milliseconds

Returns:
The new time in tos_time_t format.

2.3.7.3.1.3 compare

`char setRXData(tos_time_t a, tos_time_t b)`
Compare logical time a and b.

Parameters:
`a` – Logical time.

`b` – Logical time.

Returns:
- `1` If a>b.
- `0` If a == b.
- `-1` If a < b.

2.3.7.3.1.4 subtract

`tos_time_t subtract(tos_time_t a, tos_time_t b)`
Subtract logical time b from a.

Parameters:

`a` – Logical time.

`b` – Logical time.

Returns:

The time difference.

2.3.7.3.1.5 subtractUint32

```
tos_time_t subtractUint32(tos_time_t a,
uint32_t x)
```

Subtract a unsigned 32 bits integer from a logical time.

Parameters:

`a` – Logical time.

`x` – A unsigned 32 bit integer. If it represents a time the uint should be binary milliseconds.

Returns:

The result in tos_time_t format.

2.3.7.3.1.6 adjust

```
void adjust(int16_t n)
```

Adjust logical time by n binary milliseconds. This operation will not take effect immediately, the adjustment is done during next clock fire event handling.

Parameters:

`n` – Signed 16 bit integer, positive number advances the logical time, negative argument regress the time.

2.3.7.3.1.7 adjustNow

```
void adjustNow(int32_t x)
```

Adjust logical time by x milliseconds.

Parameters:

`x` – 32 bit integer, positive number advances the logical time, negative argument regress the time.

2.3.7.3.1.8 set

```
void set(tos_time_t t)
```

Sets the 32 bits logical time to a specified value.

Parameters:

`t` – Time in the unit of binary milliseconds type in tos_time_t.

2.3.7.3.1.9 get

`tos_time_t get(void)`

Get current time.

Returns:

Current time in tos_time_t format.

2.3.8 Core

2.3.8.1 Wiring

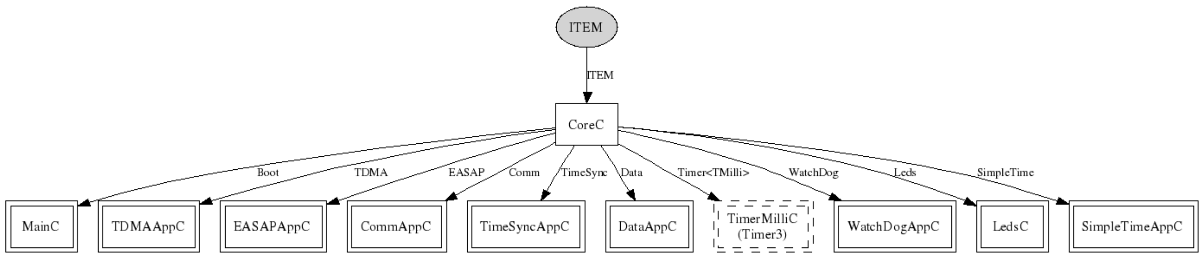


Figure 2.3.8.1.1 - Component CoreC's wiring.

2.3.8.2 Description

Core module's function is to combine all the other modules in ITEM and make them work together. This is to say that it works in the fashion as described in the article *An Adaptive TDMA Slot Assignment Protocol in Ad Hoc Sensor Networks*[2].

2.3.8.3 Functionality

2.3.8.3.1 Comm radio message reception

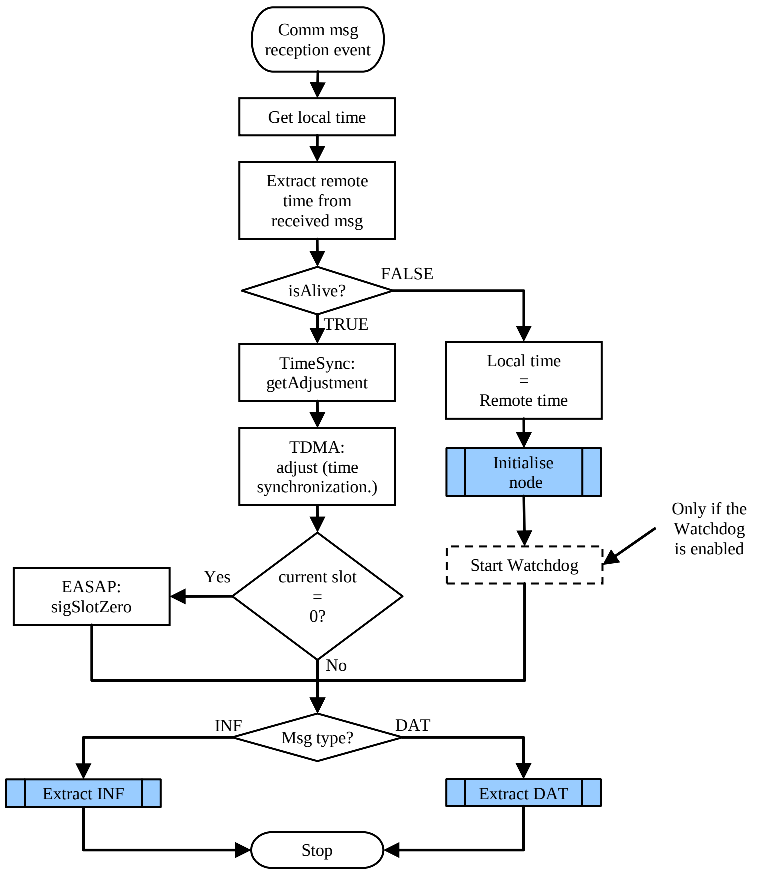


Figure 2.3.8.3.1.1 - Comm radio message reception

2.3.8.3.2 Initialise node

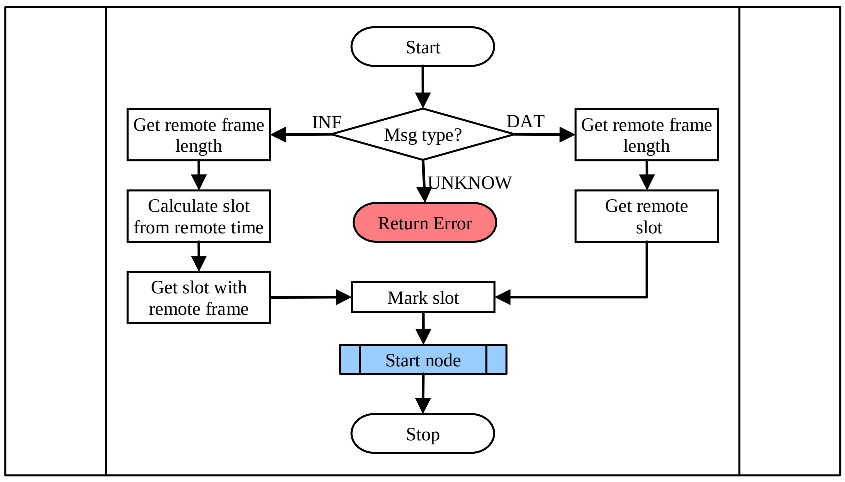


Figure 2.3.8.3.2.1 - Initialising node procedure.

2.3.8.3.3 Starting the node (Making it alive)

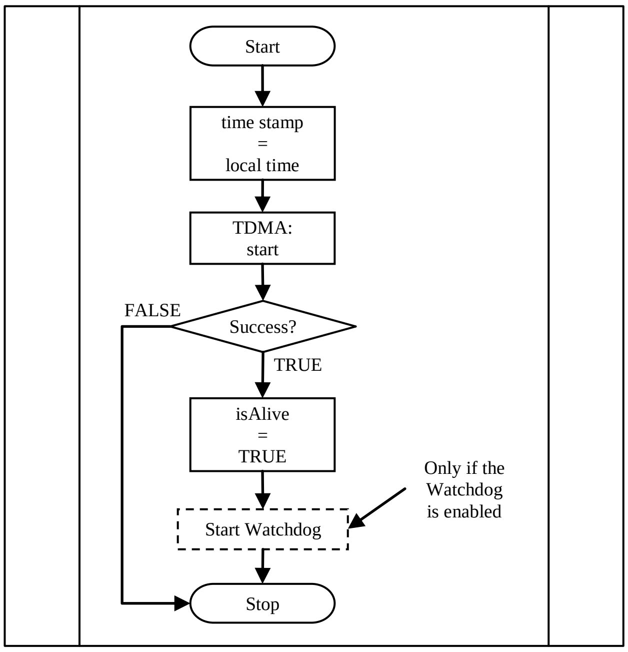


Figure 2.3.8.3.3.1 - Making a new node alive

2.3.8.3.4 Getting a new slot

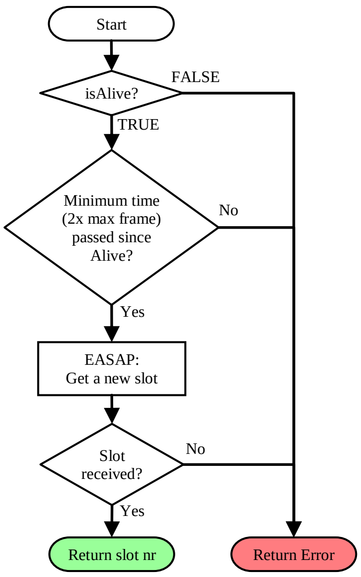


Figure 2.3.8.3.4.1 - Requesting a new slot.

2.3.8.3.5 TDMA slot handling

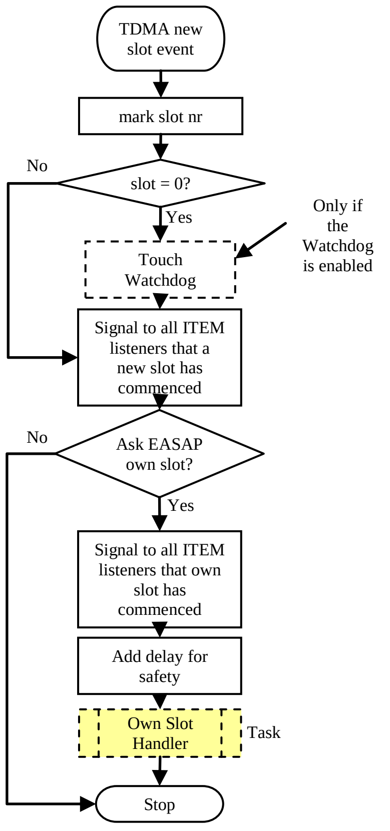

TDMA new slot event
mark slot nr
slot = 0?
No
Yes
Touch Watchdog
Only if the Watchdog is enabled
Signal to all ITEM listeners that a new slot has commenced
Ask EASAP own slot?
No
Yes
Signal to all ITEM listeners that own slot has commenced
Add delay for safety
Own Slot Handler
Task
Stop

Figure 2.3.8.5.1 Slot handling

2.3.8.3.6 Own slot handling

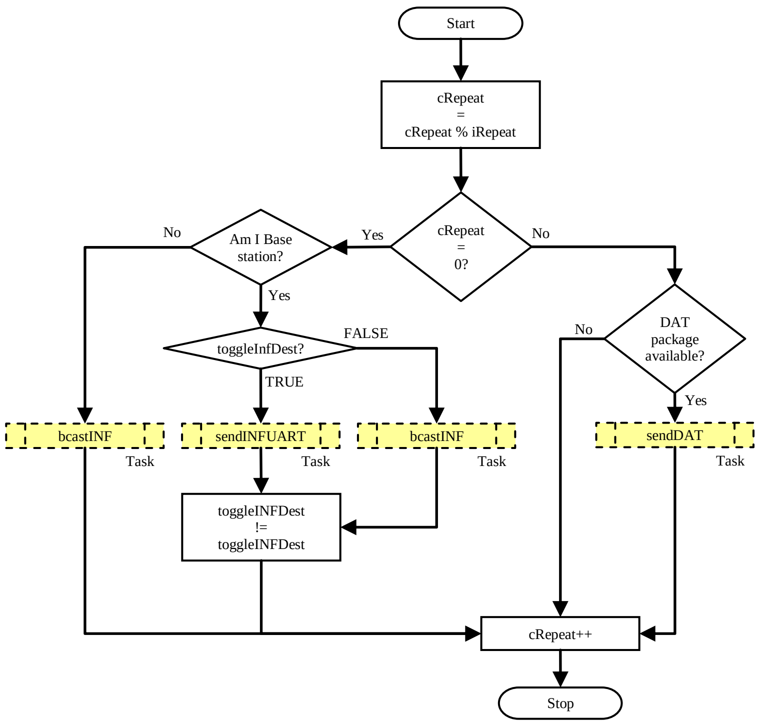


Figure 2.3.8.3.6.1 - Own slot handling.

2.3.8.3.7 PackageINF

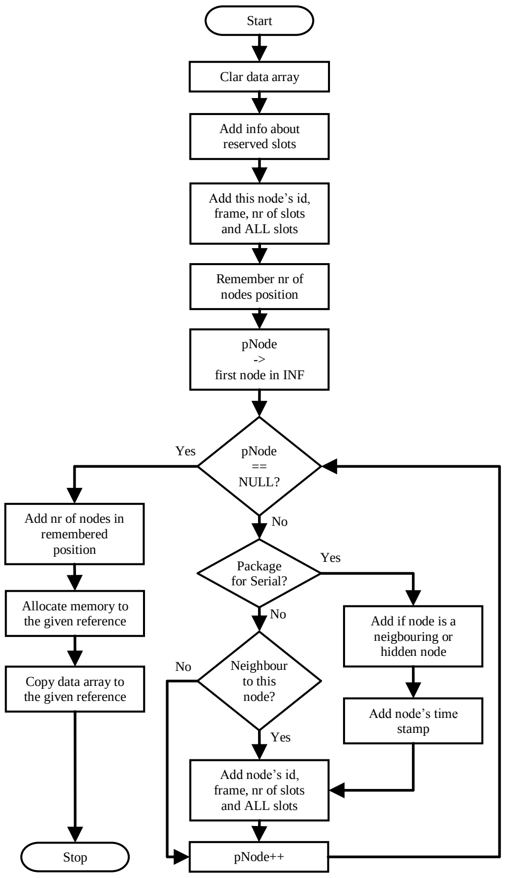

Start
Clar data array
Add info about reserved slots
Add this node's id, frame, nr of slots and ALL slots
Remember nr of nodes position
pNode -> first node in INF
pNode == NULL?
Yes
No
Add nr of nodes in remembered position
Allocate memory to the given reference
Copy data array to the given reference
Stop
Package for Serial?
Yes
No
Add if node is a neigbouring or hidden node
Add node's time stamp
Neighbour to this node?
No
Yes
Add node's id, frame, nr of slots and ALL slots
pNode++

Figure 2.3.8.3.7.1 - Packaging INF structure into a stream

2.3.8.3.8 ExtractINF

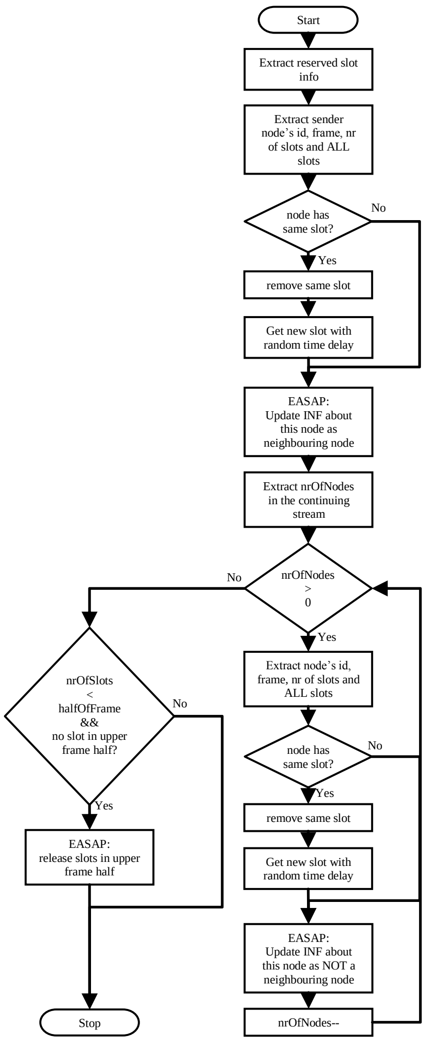

Start
Extract reserved slot info
Extract sender node's id, frame, nr of slots and ALL slots
node has same slot?
No
Yes
remove same slot
Get new slot with random time delay
EASAP: Update INF about this node as neighbouring node
Extract nrOfNodes in the continuing stream
nrOfNodes > 0
No
Yes
Extract node's id, frame, nr of slots and ALL slots
node has same slot?
No
Yes
remove same slot
Get new slot with random time delay
EASAP: Update INF about this node as NOT a neighbouring node
nrOfNodes--
nrOfSlots < halfOfFrame && no slot in upper frame half?
No
Yes
EASAP: release slots in upper frame half
Stop

Figure 2.3.8.3.8.1 - Extracting received INF package.

2.3.8.3.9 INF transmission to radio/serial

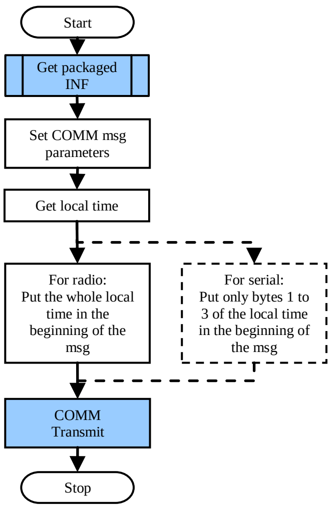

Start
Get packaged INF
Set COMM msg parameters
Get local time
For radio: Put the whole local time in the beginning of the msg
For serial: Put only bytes 1 to 3 of the local time in the beginning of the msg
COMM Transmit
Stop

Figure 2.3.8.3.9.1 - Transmitting INF stream package.

2.3.8.3.10 Extracting DAT

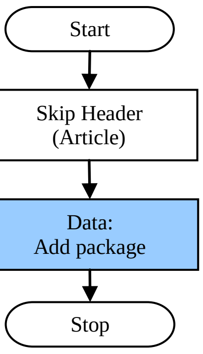


Figure 2.3.8.3.10.1 - Extracting DAT package.

2.3.8.3.11 DAT transmission to radio/serial

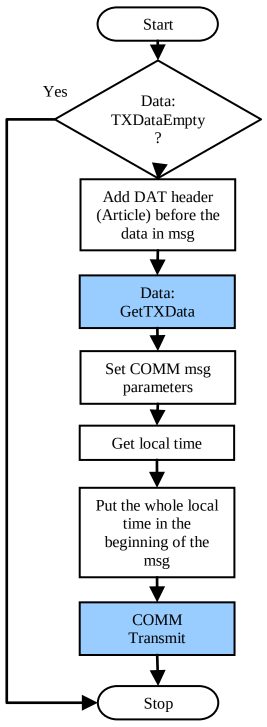

Start
Yes
Data: TXDataEmpty ?
Add DAT header (Article) before the data in msg
Data: GetTXData
Set COMM msg parameters
Get local time
Put the whole local time in the beginning of the msg
COMM Transmit
Stop

Figure 2.3.8.3.11.1 - Transmitting data to Radio/Serial

2.3.8.3.12 Comm SendDone event handling

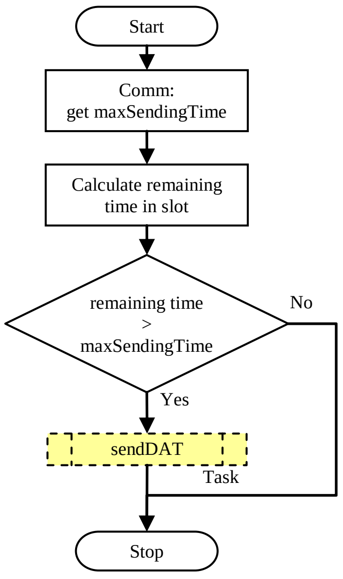


Figure 2.3.8.3.12.1 – SendDone event handling

2.3.8.3.13Removing Inactive nodes

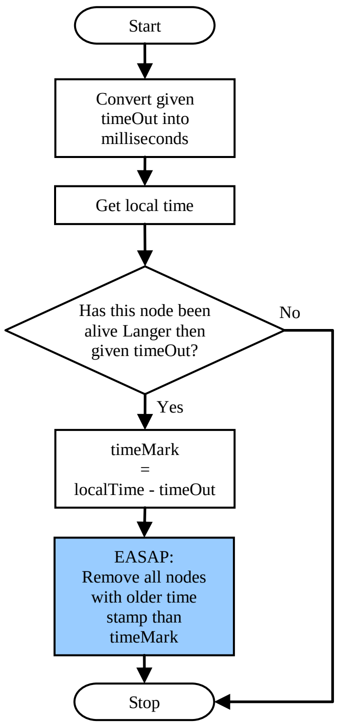


Figure 2.3.8.3.13.1 - Removing all inactive nodes procedure.

2.3.8.4 API

There is no API for the Core module It’s function is to combine all other modules and simply to make them to work together as defined in the article[1].

3. ITEM API

3.1 Commands

3.1.1 `setBase`

`void setBase(bool bVal)`

Sets this node's behaviour as base station or not.

Parameters:

`bVal` - `TRUE` if it should function as the base node. `FALSE` if as a normal node on the network.

3.1.2 `setINFRepitition`

`error_t setINFRepitition(uint8_t repitition)`

Sets the INF package broadcasting frequency.

Parameters:

`reptition` - How often it should be broadcasted. Every 2nd, 3rd etc. Only value above MIN_REPEAT_VAL are valid.

3.1.3 `getSlot`

`error_t getSlot(uint8_t *slotNr)`

Gets a slot and assigns it to itself, if possible.

Parameters:

`slotNr` - slot number if successfull. 0 if not found.

Returns:

SUCCESS if slot is received. Else,

- `ITEM_E_MIN_TIME_NOT_PASSED` If define (2 x max frame length) not passed.
- `ITEM_E_GETTING_SLOT` if max frame length has been reached.

3.1.4 `startDefault`

`error_t startDefault(void)`

Starts the ITEM module with default parameters. Must be called to make the node be part of the network. Default parameters are:

- starting slot: 0
- frame: 4
- interval: 1000 ms

Returns:

`SUCCESS` if all went ok. Else,

- `TDMA_E_ALREADY_RUNNING` if the TDMA is active. Please stop it first by using method `stop()`.
- `TDMA_E_INVALID_INTERVAL` if the value of interval is not between `MIN_INTERVAL` and `MAX_INTERVAL` (*`TDMA.h`*).
- `TDMA_E_INVALID_FRAME_LENGTH` if the frame length is below `MIN_FRAME_LENGTH` (*`TDMA.h`*).
- Error messages from the interface `Timer.start()`.

3.1.5 start

```
error_t start(uint8_t sSlot, uint8_t sFrame, uint16_t
               sInterval)
```

Starts the ITEM module with given parameters.

Parameters:

`sSlot` - Starting slot number. If the slot is bigger than `sFrame`, it is adjusted to a valid slot within `sFrame`.

`sFrame` - Frame[2] length.

`sInterval` - Slot time length.

Returns:

`SUCCESS` if TDMA is started. Else,

- `TDMA_E_INVALID_INTERVAL` if the value of interval is not between `MIN_INTERVAL` and `MAX_INTERVAL` (*`TDMA.h`*).
- `TDMA_E_INVALID_FRAME_LENGTH` if the frame length is below `MIN_FRAME_LENGTH` (*`TDMA.h`*).
- Error messages from the interface `Timer.start()`.

3.1.6 removeInactiveNodes

```
error_t removeInactiveNodes(uint16_t timeOut)
```

Removes all nodes from the INF that are older than timeOut seconds from their last transmission.

Parameters:

`timeOut` - Time passed in seconds from last transmission.

Returns:

`SUCCESS` if all went OK. Else, `ITEM_E_INVALID_VALUE` if the given parameter is lower than `MIN_TIME_PASSED` as defines in *`ITEM.h`*.

2 Frame - A frame length is defined as a combination of slots that are repeated. For example a frame of 4 contains 4 slots that are repeated continuously.

3.1.7 getDATPkg

`uint8_t getDATPkg(uint8_t **pPkg)`

Gives the first entered package from the buffer.

Parameters:

`pData` - Reference to the where data is to be added.

Returns:

Number of bytes of containing in the stream of `pPkg`.

3.1.8 setDATPkg

`error_t setDATPkg(uint8_t length, uint8_t *pData)`

Sets the data package for transmission.

Parameters:

`length` - Length of the data int the given stream.

`pData` - Reference to the data itself.

Returns:

`SUCCESS` If all went well. Else,

- `DATA_E_BUFFER_FULL` If the data buffer is full.
- `DATA_E_DATA_TOO_LONG` If the data length is bigger than the `BUFFER_DATA_SIZE` as defined in *`Data.h`*.

3.1.9 nrOfFreePlacesTransData

`uint8_t nrOfFreePlacesTransData(void)`

Get count of free places in data queue for transmission.

Returns:

Count of free places in transmission buffer.

3.1.10nrOfFreePlacesRecData

`uint8_t nrOfFreePlacesRecData(void)`

Get count of free places in data queue for reception.

Returns:

Count of free places in reception buffer.

3.1.11nrOfOccupiedSlots

`uint8_t nrOfOccupiedSlots(void)`

Get count of occupied slots (Own slots + neighbours).

Returns:

Count of occupied slots.

3.1.12reserveSlot

`bool reserveSlot(tSlot slot)`

Reserve slot.

Parameters:
`slot` – Reserved slot.

Returns:
`TRUE` If reserved. Else, `FALSE`

3.1.13releaseOwnedSlots

`void releaseOwnedSlots(void)`

Release all owned slots.

3.2 Events

3.2.1sigOwnSlot

`void sigOwnSlot(tSlot slot)`

Signal when a new slot occupied by this node has commenced.

Parameters:
`slot` – New slot number.

3.2.2sigSlot

`void sigSlot(tSlot slot)`

Signal when a new slot has commenced.

Parameters:
`slot` – New slot number.

3.2.3dataReceived

`void dataReceived(void)`

Signals when and for each data(DAT) package reception. Each package is in the buffer as in FIFO order.

4. Pre-processor commands

When using the ITEM API, many of the definitions can be changed. This is highly recommended to be done in the makefile of the module that is *using* ITEM. Please note, if not defined in the makefile, default values are set.

Each definition is set as following:

```
CFLAGS += -DDEFINITION=VALUE
```

Please be reminded on the fact that each definition should be add (+=) or there might be some definition lost and may result into a different behaviour.

4.1 Watchdog
For enabling or disabling the watchdog.

```
WATCH_DOG=[ENABLE | DISABLE]
```

4.2 Base station node
To make the node act as a base node or as a *normal* node. The main function for the base node is to send data to serial port.

```
IS_BASE_STATION=[TRUE | FALSE]
```

4.3 Max data length
This parameter limits the maximum DATA length allowed to be sent for each transmission, whether it is to radio or serial.

```
TOSH_DATA_LENGTH=NR_OF_BYTES
```

4.4 Minimum TDMA frame length
Minimum frame length allowed in the contention area.

```
MIN_FRAME_LENGTH=LENGTH
```

4.5 Maximum TDMA frame length
Maximum frame length allowed in the contention area.

```
MAX_FRAME_LENGTH=LENGTH
```

4.6 TDMA slot interval length
Slot length in milliseconds.

```
DEFAULT_INTERVAL=MS_LENGTH
```

4.7 Time Out for node presence
Maximum time allowed, in seconds, for each node to respond for assuming it's still part of the network.

```
TIME_OUT=SECS
```

4.8 Radio transmission range
This parameter sets the power of the radio transmission. A value can be set between 0 -25 where 0 is predefined power level and 255 is the maximum.

```
CC2420_DEF_RFPOWER=POWER
```