SAF400x

Overall SW user manual

Rev. 01.08 — 9 November 2017 **User manual**

Document information

Info	Content
Keywords	Software, SAF400x, User Manual
Abstract	Describes the overall SW mechanisms for the SAF400x family

Revision history

Rev	Date	Description
1.00	20170124	initial version
1.01	20170221	Refined doc structure
1.02	20170411	Added usage of some SoftConfig settings, example protocol messages and configuration of use cases
1.03	20170511	Added section on domain Simulcast
1.04	20170518	Corrected command for Simulcast
1.05	20170706	Added section on CM SoftConfig parameters
1.06	20170807	Rephrased some of the text in the Simulcast section
1.07	20170829	Added references to HD documentation
1.08	20171109	Updated transport protocol section for message double buffering

Contact information

For more information, please visit: http://www.nxp.com

1. Introduction

This software user manual is to be used by customers creating radio control software or middleware for a SAF400x - or SAF777x-based radio reception system. It is to be used in conjunction with the SAF400x/SAF777x protocol documents and subsystem user manuals. This overall software user manual provides an overview of the SAF400x and SAF777x interfaces. It describes the internal system partitioning as well as the low level transport protocol to be used for controlling each of the sub-systems within the SAF400x or SAF777x radio SoCs. It also provides an overview of the various use cases supported by SAF400x and SAF400x and how the devices can be configured to support different combinations of analog and digital radio receivers. More information on controlling the features of receivers for a specific radio standard (AM/FM/WX, DAB, HD-Radio or DRM) can be found in the software user manual for the specific radio standard.

2. System and software architecture overview

The software system overview for SAF400x and SAF777x is discussed in the following sections. The blue and white blocks in the figures are relevant to software. For each block, the software functions that are implemented on it are specified.

2.1 SAF400x system overview

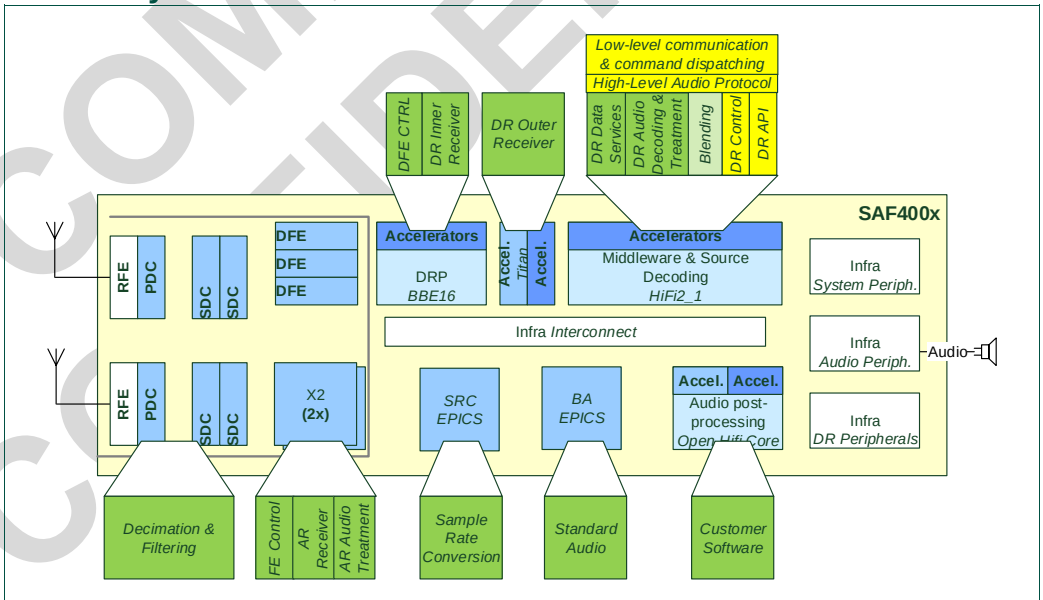

Fig 1. SAF400x software system overview

The SAF400x (hardware plus software) is a global multi-standard radio receiver. The main functionality provided is:

- AM/FM/WX analog radio reception

- DAB/DRM/HD Radio digital radio reception
- Built-in standard audio processing
- OpenCore for advanced audio processing (customer software)

2.2 SAF777x system overview

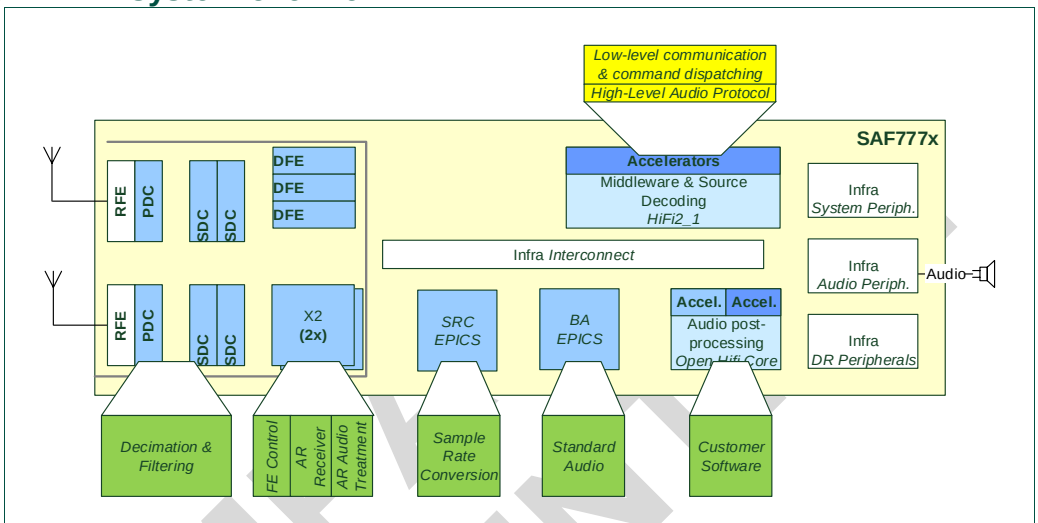

Fig 2. SAF777x software system overview

The SAF777x is an analog radio receiver derived from the SAF400x by removing DR-specific hardware and software, and adapting the remaining components to run standalone. Main functionality provided is:

- AM/FM/WX analog radio reception
- Built-in standard audio processing
- OpenCore for advanced audio processing (customer software)

3. Control sub-systems

This SoC integrates many sub-systems. Each sub-system has defined its own protocol, called domain protocol message in this document. These domain protocol messages will be encapsulated in the transport protocol layer (see section 9). By encapsulating the various domain specific protocol messages, a shared way of communicating between host and system is possible.

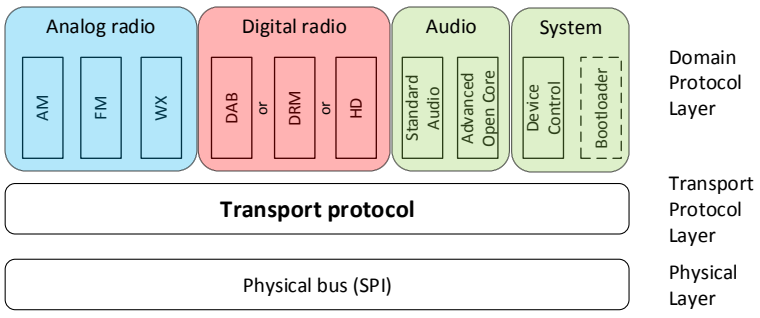

Fig 3. Position of the domain protocols in the communication stack

Each sub-system or domain will define messages to control it. These messages will be described in a separate document per sub-system or domain like FM, AM, WX, DAB, DRM, HD, etc.

Table 1. Overview of domain specific documentation

AM		**Standard Audio**	
	AM Protocol		Audio Protocol
	AM User Manual		Audio User Manual
FM			Audio Protocol Header file
	FM Protocol	**Advanced Open Core**[1]	
	FM User Manual	**Device Control**	
WX			Device Control Protocol
	WX Protocol	**Boot Loader**	
	WX User Manual		Boot Loader Protocol
DAB[1]			Boot Loader User Manual
	DAB Protocol	**Soft Config**	
	DAB User Manual		Softconfig User Manual
DRM[1]			
HD[1]			
	HD User Manual		
	HD API documentation (RX_*)		

[1] If applicable for this system release. A system release will typically cover only one DR standard.

3.1 Control execution architecture

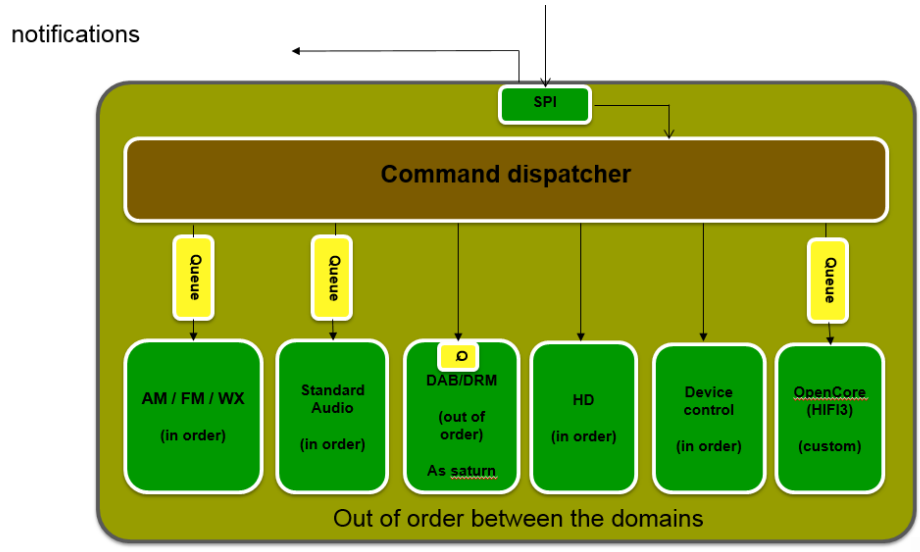

Fig 4. Queuing in the communication stack

The control execution architecture, with the position of domain protocols is shown in Fig 4.

The command dispatcher links the command and request messages (API message) from the application (via SPI) to the appropriate domain.

3.1.1 Targeted API throughput for different domains

The API throughput for Mercury system is based on the domain that is enabled. The entire system is targeted to handle a maximum of 60 commands per 10 ms. The numbers in this whole section are targets and need to be confirmed.

3.1.2 Analog radio

The analog radio can receive up to 20 commands per 10 ms.

3.1.3 Digital radio (DAB/DRM)

When the digital radio (DAB/DRM) is compliant to the protocol, it can receive up to 60 commands per 10 ms; for example, when it has only three unacknowledged commands pending. The feature is same as in Saturn.

3.1.4 HD digital radio

The HD digital radio can receive up to 20 commands per 10ms budget. As with Saturn (defined by DTS), HD radio does not have a queue.

3.1.5 Standard Audio

The maximum message payload for standard audio is 128 bytes. The specifications based on the type of commands are:

- Memory r/w commands
  - Maximum 50 commands per 10 ms
  - Maximum of 12 address updates in one command (see Audio protocol document)
- Audio control command
  - One control command counts as 4 Memory r/w commands

The maximum total command size sent in 10 ms would be 256 bytes (memory and control)

3.1.6 Device control

The device control domain can handle up to 20 commands per 10 ms.

3.1.7 Timing of command replies

The timing at which the replies to commands are generated will vary between domains.

For the Device Control and Standard Audio domains, the command replies will, typically, be generated very quickly, as soon as the commands have been fully processed.

For the Analog Radio and DAB/DRM domains, the replies will also be generated quickly, but they can indicate that the action triggered by the command has been started. This is the case for a Tune_command. In this case, the reply will be available as soon as the tuning has started. However, the tuning will take some time after that, and its end will be signaled by a notification.

For the HD domain, the timing of the reply still has to be defined.

3.1.8 Using multiple SPI ports

Not support yet. To be updated later.

4. Security flow

To be described in a later update.

4.1 Keycodes

Refer to [SAF400x_SoftConfig_UM].

4.2 SoftConfig images

Refer to [SAF400x_SoftConfig_UM].

4.3 Open Core images

TBD.

5. Open Core development

Open core development is not supported for the current release, more information will follow in a later SAF400x/SAF777x customer package.

6. SoftConfig

The SoftConfig image for SAF400x/SAF777x contains information on the configuration of the device, including hardware and software features to be enabled. The SoftConfig image also includes a keycode vector. This keycode vector defines which commercial features to be activated.

The SoftConfig image must always be provided at boot and NXP will provide a SoftConfig image with a default hardware and software feature configuration. However, it will be possible for the customer to modify some of these settings and use a custom SoftConfig content.

In the subsequent sections, the common (non-domain specific) SoftConfig options are described. Refer to the different SAF400x UM documents for the domain specific SoftConfig options.

6.1 Pin Multiplexing

The SAF400x includes pin multiplexing on most digital pins. For each application, the SoftConfig parameters need to be modified to configure pins to the required functionality. Please refer to [SAF400x_DS] for the functionality options for each pin.

Note that in some cases, a SAF400x function is not available unless the pin multiplexing is also correctly configured. For example, if GPO9 is meant to be used as the primary host port interrupt, then the function GPO_9 needs to be configured as the function for the SAF400x pin GPIO_9 (the default function is NOT_USED).

6.2 Miscellaneous

Additionally, the SAF400x SoftConfig generic section includes the following application specific settings.

Table 2 Generic SoftConfig miscellaneous settings

Parameter	Description	Default	Optional selection values
ClockOut	XTAL output enable setting	XTAL_output_enabled	no_XTAL_output XTAL_output_enabled
ClockSrc	external clock source of the IC	XTAL_input	XTAL_input clock_input

The parameter *ClockOut* is be used to enable the XTAL buffer output to clock another device such as the TEF710x. By default, this output is enabled. Refer to [SAF400x_AN] on how to use this output for a specific application.

The parameter *ClockSrc* is used to determine the source of the clock of the SAF400x. The clock input is always taken from the signal on pins XTAL_IN and XTAL_OUT. However, depending on whether the signal on those pins is provided directly by a crystal of by one of the other devices of the SAF400x family will require different internal settings. The selection must be indicated using the *ClockSrc* parameter. By default, the input from the crystal is selected. Refer to the [SAF400x_AN] for information on the options to route the clock in a multi IC system.

Refer to [SAF400x_SoftConfig_UM] on how to generate a SoftConfig image with the required settings for a customer application.

7. Booting the SAF400x/SAF777x

Refer to [SAF400x_Boot] for the SAF400x/SAF777x boot process.

After boot, the application will be started and an Up notification will be send to all active ports. The notification is described in the device control protocol document. This notification indicates what domains are started, and if errors are detected during application start. In case of an error, details can be requested with device control commands.

8. Physical interfaces

8.1 Interface overview

The Mercury SoC interface block diagram is shown below.

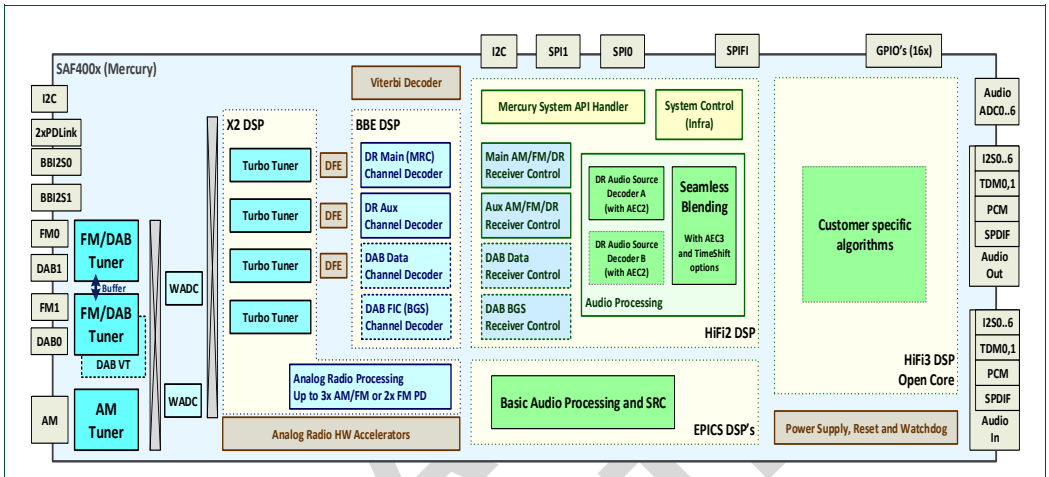

Fig 5. Mercury SoC interface

The following interfaces are identified:

- RF inputs: two DAB3 inputs, two FM inputs, and one AM input
- Two baseband I2S inputs/outputs and two Phase-Diversity (PD) in/out links
- Audio input and audio output interfaces (I2S, TDM, PCM, SPDIF, and analog inputs)
- A SPIFI (Quad-SPI) and two SPI interfaces
- Two I2C interfaces
- One UART interface
- GPIO interface with 16 I/O lines

The next sections will give a detailed overview of each device interface.

8.1.1 RF input

Mercury RF input is processed in the following sub-systems: Radio Front-End (RFE) analog, RFE digital, and Digital Radio Processing (DRP).

The RFE analog accepts an antenna signal and produces a wideband radio signal from a radio band and feeds it to the RFE digital sub-system. The main building blocks for the RFE analog are two hybrid FM/DAB3 tuners with the corresponding balanced RF inputs (there are two FM inputs: FM0, FM1; and two DAB3 inputs: DAB0, and DAB1), an AM tuner with a corresponding unbalanced RF input, and two wideband ADCs.

The RFE digital down-samples a wideband radio signal to the bandwidth and sample rate that are required for the respective radio standard. In case of FM band, the signal is also demodulated by the RFE digital, and converted into audio by the processing chain inside

the RFE digital. The main building blocks for the RFE digital are four turbo tuners, four analog radio processing accelerators, and three analog radio receivers. For Digital Radio, the signal output of the RFE digital is fed into the Digital Radio Processor (DRP) sub-system which further processes the down-sampled IF signal, and performs channel decoding and source decoding. The main building blocks for the DRP are three Digital Front-End (DFE), a BBE DSP, and a HiFi2 DSP.

8.1.2 Baseband inputs and outputs

The SAF400x RF and baseband inputs and outputs are designed such that:

- The RF inputs can be routed to either or both WADC's
- The four turbo tuners can each take input from either WADC
- The two external IIPD interfaces can either be input or output and can only be used for routing of FM baseband signals for Phase Diversity (PD).
- The two external BBI2S interfaces can be either input or output and can be used to route baseband signals to support digital radio decoding. This includes baseband signals required for MRC support

8.1.3 Audio inputs and outputs

The audio data interfaces are described in [SAF400x_Audio_UM]. The interfaces can be controlled using the commands described in [SAF400x_Audio_Protocol] and are required to be setup for each application as part of the SoftConfig.

8.1.4 SPI interfaces

The CarDSP system has three SPI interfaces to communicate with host processors and to control a flash device. See 8.2 for an overview of the recommended control port configurations.

The host communication is designed as to implement a two-layered protocol structure: a generic transport protocol designed to, on the one hand, maximally utilize the available bandwidth and, on the other hand, reduce the interrupt load on the host processor and maintain low latency for message and notification processing (see section 9); domain protocols implementing the domain specific controls. The generic transport protocol has been implemented with support to enable the host to implement error detection and retry mechanisms.

For host communication, the CarDSP operates as an SPI slave. The host is in charge of monitoring the synchronization state and load state. If necessary, the host shall take appropriate measures to re-sync or resolve the overload state. The transport protocol is implements a mechanism to allow the host to determine the transport sync state and initiate resynchronization if required (see 9.1.4).

The SPI clock speed used for host communication is recommended to be at least 10 MHz in order to enable a response time of 1 millisecond.

The SPI port configuration is fixed in the current release. SPI1 should be considered as the main port SPI, with SPI0 as the secondary SPI port. The configuration of the SPI ports is as follows:

Parameter	SPI0 (secondary)	SPI1 (primary)
SPI Mode	Mode1 (clock phase on second edge; clock polarity low)	Mode1 (clock phase on second edge; clock polarity low)
Interrupt GPIO	GPO3	GPO2
Interrupt Mode	Active Low	Active Low

Each transport protocol frame is 256 bytes.

8.1.5 UART interface

The UART interface is not used in a radio application during normal use, but can used to allow NXP to assist customers in debugging/monitoring during customer application development. It is recommended that customers design their radios with this interface available, at least, during application development.

The speed and protocol used depend on the use case.

8.1.6 GPIOs

The SAF400x or SAF777x GPIO pins can be connected to different functions in the device. The following steps must be followed in order to connect a GPIO pin to sub-system function:

1. Update the SoftConfig settings to configure the required SAF400x/SAF777x pin as General Purpose Input or Output
2. If using the sub-system assignment statically, update the SoftConfig settings to assign the pin to the required sub-system
3. Generate the SoftConfig image and use it to boot the device
4. If modifying the sub-system assignment dynamically, issue the API command to connect the GPIO to a sub-system via ConnectGPIOtoSubsystem_cmd (see [SAF400x_DevControl_Protocol])
5. If applicable, issue the sub-system command to connect the GPIO to a sub-system function.

   Example: to connect a GPIO to the Audio Fast Mute function, the command APCConnectGPIO_cmd (see [SAF400x_Audio_Protocol]) must be used.

The GPIO can then either be controlled or read by the functions in the connected sub-system. Example: if connected to sub-system DevCtrl sub-system, the GPIO can be controlled via command WriteGPIO_cmd (see [SAF400x_DevControl_Protocol]).

When a GPIO is connected to the DevCtrl domain, the host can use SPI messages to control or read out the level of the GPIO of the SAF400x/SAF777x. This allows the device to operate as a IO expander.

8.2 Control port configuration

The configuration of the control port(s) used by the host to control the devices of the SAF400x family needs to take into account the data throughput required for the communication interface. The table below shows a set of possible control port configurations including the minimum required clock speed for those ports.

Table 3 Control port configuration options

Option	Device	Analog Radio	Digital Radio Control	Digital Radio data services	Data rates
1	SAF777x	SPI	x		SPI[1] > 2 MHz (4 x FM)
2	SAF400x	SPI	SPI		AR SPI[1] > 2 MHz DR SPI[1] > 5 MHz
3	SAF400x	SPI			SPI[1] > 10 MHz
4	SAF400x	SPI		SPI	AR+DR control SPI[1] > 3 MHZ DR data SPI[1] > 3 MHz

The control of the Audio and DevControl sub-systems can be combined in either of the control ports used.

As described in section 5, the CarDSP device includes one DSP core open for customer algorithm implementation. If the bandwidth requirements for the host to communicate with the customer algorithm exceed the capabilities of sharing the same SPI interface with the generic CarDSP control, it is advised to reserve one SPI interface solely for communication with the open DSP core.

Table 4 Control port configuration options

Option	OpenCore
OC1	Dedicated SPI
OC2	-shared-

[1] The SPI clock speed recommendations are to be verified with real data.

9. Transport protocol

A transport protocol is defined for the SAF400x/SAF777x that determines a fixed length for messages, thereby also removing the burden on the host of processing a large amount of short messages. This enables reducing the frequency with which the host has to swap execution between threads in order to serve hardware events.

The host control protocol stack is divided into two layers:

- A transport protocol layer, taking care of the OSI Transport and lower layers. It encapsulates domain protocol messages into an, for the physical layer, optimal format. Currently the SPI (slave) is supported. The transport protocol layer is described in this document.
- A domain protocol layer, above the OSI Transport layer, that specifies the actual commands, replies and notifications that are exchanged between the host and the designated sub-system. For each sub-system mentioned below a domain specific protocol is defined and described in a separate document.
  - DAB [1] (see [SAF400x_DAB_Protocol])
  - HD [1] (see [RX_*])
  - DRM [1]
  - AM (see [SAF400x_AM_Protocol])
  - FM (see [SAF400x_FM_Protocol])
  - WX (see [SAF400x_WX_Protocol])
  - Standard Audio (see [SAF400x_Audio_Protocol])
  - Device control (see [SAF400x_DevControl_Protocol])
  - Advanced OpenCore [1]

[1] If applicable for this system release. A system release will typically cover only one DR standard.

9.1 Control protocols and software API

The SAF400x CarDSP family merges the audio, analog and digital radio domains in one device. The transport protocol supports the combination of the requirements of the different control APIs while supporting a single SPI control interface. The requirements vary from a low data rate with low latency to a one with high data rate and a high latency.

The goal of the transport protocol is to reduce the interrupt load on the host. This is done by enabling the host to send large blocks without the need to check pins or responses from the target when using SPI. This enables the host to use a DMA controller to send or read multiple frames without any interaction of the CPU. At the same time the processing power and memory of the target cores are limited. The sub-systems can handle a fixed amount of commands in a fixed time frame[2]. The command dispatcher is able to buffer a number of commands to spread a burst of incoming commands over time. If the buffers

[2] Note that the amount of buffers is chosen as such that all normal use cases will work even in a stretched situation with a 20% overload.

are almost completely used the host will be informed via a flow control[3] mechanism for SPI.

9.1.1 Transport protocol for SPI

The transport protocol uses fixed size block sizes. Each block is called a frame. Each frame can hold multiple segments with different sizes. Each segment will hold a payload corresponding to a specific protocol stack. In this way, the latency is fixed and related to the SPI clock speed.

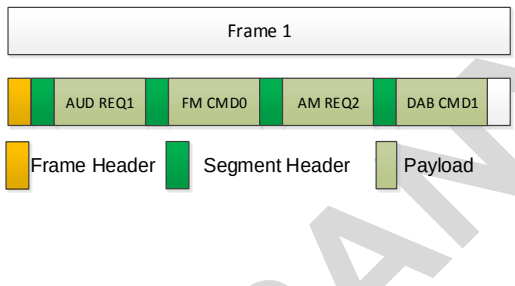

Fig 6. Example of an SPI frame

The frame size can be configured in the target before boot (using SoftConfig). Frames from host to target and target to host will use the same fixed frame size. In this document a frame size of 256 bytes is taken as a typical example.

For every frame sent from the host to the target, there is also a frame sent from the target to the host. When the target has no data to send to the host, it will send an empty frame.

In case the host has no data to send, but still wants to read from the target, an empty must be sent while reading a target frame.

An empty frame is defined as a frame that holds one segment with the *ProtocolId* set to *Empty*.

The syntax of a frame is given in Table 5.

[3] Flow control on message level is not yet fully defined and will be subject to change in the near future.

Table 5. Frame syntax

Syntax	Number of bits	Type
DataLinkFrame {		
FrameSync	16	uimsbf
FrameCounterValid	1	bslbf
FrameCounter	7	uimsbf
Synced	1	bslbf
Ignore	1	bsblf
LoadStatus	2	bsblf
RFA	1	bslbf
FrameLength	3	uimsbf
repeat {		
DataLinkSegment	N * 8	uimsbf
} until (DataLinkSegment == empty) || (DataLinkSegment == last)		
Padding	M	uimsbf
CRC	16	uimsbf
}		

FrameSync: Is a synchronization string to uniquely start the frame with the value 0xB649.

FrameCounterValid: Set to ‘1’ to indicate that the FrameCounter can be interpreted. This aids the synchronization process as described in section 9.1.4.

FrameCounter: For host to target the *FrameCounter* is a sequence number that is normally increased for each frame. For the target to host communication it indicates the last accepted frame from the host. See section 9.1.4 for more details.

Synced: Is set to ‘1’ as soon as the other side has synchronized itself to the stream. See section 9.1.4 for the synchronization process.

Ignore: Is set to ‘1’ to mark this frame as a dummy frame. A dummy frame has no sequence number and can be used to reduce the load on the host See 0 for more details. Also a dummy frame shall be used to request a re-transmit of a previous frame if a CRC error on a target to host frame is detected, see Section 9.1.4.6 for more information.

LoadStatus: Only used in target to host frames.

Table 1. Load status definition

Value	Meaning	Required action
0_b00	Normal load	Nothing
0_b01	Message load is critical.	Hold new requests until *LoadStatus* goes to normal
0_b10	CPU overload.	Hold new request until *LoadStatus* goes to normal. Target will generate ignore frames too.
0_b11	Overload, previous frame is handled.	Hold new request until load status goes to normal. Request

		retransmission for this and other outstanding frames.

The generation and desired handling of the LoadStatus field is described in CpuOverload.

FrameLength: Indicates the length of the frame in power of 2 steps starting at 32. This means that the smallest frame is 32 bytes long where the frame can be maximal 4096 bytes long.

$$Length = 32 * 2^{FrameLength}$$

A typical the frame in application mode of 256 bytes long shall be indicated by a *FrameLength* of 3. Thus 256 = 32 * 2^3 = 32 * 8 = 256.
The *FrameLength* shall be configured via softconfiguration and is fixed over the application lifetime.

DataLinkSegment: The SPI frame contains minimal one *DataLinkSegment*. In case of an empty segment the frame is empty. In case of none empty segment the only or last one is marked as being the last one, using the moreFollowingFlag. The syntax of a *DataLinkSegment* is given in section 9.1.2.

Padding: The frame is padded with 0's until the CRC field.

CRC: The cyclic redundancy check is calculated according the CCITT CRC-16 polynomial: $X^{16} + X^{12} + X^5 + 1$ over frame length, excluding the 2 byte CRC word. The initial value is 0xFFFF and the result is inverted.

9.1.2 Data link segment

Each protocol message is encapsulated in one or more data link segments. Protocol messages, which are larger than a frame, shall be split over multiple segments in different frames. The number of segments needed to carry such a message cannot be determined on beforehand. Since the system allows other protocols to be inserted as they appear, a larger message in transport can be delayed a bit. In that frame the segment will be a bit smaller while another protocol segment was inserted. For this reason, the last segment is indicated as last and the first as first. In between can be number of segments can vary based on other traffic. The order of segments shall not change and is safeguarded by the *FrameCounter* and *CRC* of the datalink frame.

The syntax of the data link segments is given in Table 6.

Table 6. Syntax of Data Link segment

Syntax	Number of bits	Type
DataLinkSegment {		
MoreFollowing	1	bslbf
FirstSegment	1	bslbf
LastSegment	1	bslbf
ProtocolId	5	uimsbf
Encrypted	1	bslbf
Priority	2	uimsbf
Rfa	1	bslbf
SegmentLength	12	uimsbf
If (FirstSegment == ‘1’ AND LastSegment == ‘0’) {		
TotalMessageLength	32	uimsbf
}		
for (i = 0 ; i < SegmentLength; i++) {		
payloadData	8	uimsbf
}		
}		

MoreFollowing: Set to 1 to indicate that after this segment another segment can be found in this frame. This option is used when a frame holds more than one segment.

When a protocol message does not fit in one frame, the message is packed into multiple segments over multiple frames. The *FirstSegment* and *LastSegment* flags indicates whether the message is segmented or not. In case of a segmented message the first header is extended with a length field. This allows the receiver to allocate memory for collecting the segments into.

FirstSegment: If set this is a new segment. If the *LastSegment* is not set then the message is segmented over multiple frames. The *TotalMessageLength* is added to the header.

LastSegment: If set this is the last segment. The combinations of the first and last segment flags are summarized in Table 7.

Table 7. Segmentation control

FirstSegment	LastSegment	Description
0	0	A middle segment of a multi segment message
0	1	The last segment of a multi segment message
1	0	The first segment of a multi segment message. The *TotalMessageLength* is provided
1	1	A single segment

ProtocolId: Indicator for the protocol being transferred.

Table 8. Protocol ID

ProtocolId	Description
0x00	Empty
0x01	Device Control
0x02	Standard Audio
0x03	FM Radio
0x04	AM Radio
0x05	WX Radio
0x06	DAB Digital Radio
0x07	DRM Digital Radio
0x08	HD Digital Radio
0x09	OpenCore
0x0A	NXP Bootloader
0x1F	TransportProtocol[4]

Encrypted: If set, the payload is encrypted. (Is not supported at the moment).

Priority: Allows up to two[5] different levels of priorities for the same protocol. E.g. when a large protocol message is being transmitted occupying multiple frames, a priority message could be inserted in one of the frames to allow fast responses to commands. Initially priority will be set to 0. When a higher priority message needs to be transferred before the previous message from the same protocol is being transmitted, a segment will be transmitted with increased priority level compared previous pending message of the same protocol (in this case 1).

Note: Only supported for DAB and DRM digital audio since these protocols can have large data messages which can be up to 80 kB large.

SegmentLength: this field indicates the length of the payload data within this segment.

TotalMessageLength: In case of a multi segment message the total length equal to the sum of all payloads of each segment. The receiver can use this value to allocate enough memory for the complete message.

9.1.3 Payload structure

As part of the message payload, the Transport Protocol includes a Unique Sequence Number (USN) in commands, replies and notifications This allows the host to keep track of the processed commands and of the incoming notifications

[4] To support retry requests of SPI frames which needs to be handled on the Transport layer.

[5] The number of priority levels is set to two for the sake of simplicity. It does not make sense of keep track of more than one long message.

The host adds a USN in the payload of commands and the same value will be returned in the corresponding command reply. Additionally, the system will generate, itself, a USN and will include it in the notifications

In order to differentiate the USN from commands and replies from that of notifications, and to facilitate tracking the replies, it is recommended that the host generates the USN in the commands in sequential order, and in the range of 0 to 127. The system will generate the notifications in the range 128 to 255.

All messages include a receiver-handle next to a message specifier (opcode). A handle uniquely identifies the receiver instance for which the message is meant. A system handle (receiver handle with value equal to 0x00) is used in all messages that relate to resource creation or deletion and to resources common to all receivers. After a receiver instance is destroyed, the handle may be reused for creating a new receiver.

Note: The USN and receiver handle are now only implemented in the protocol of the DAB domain. These will be implemented in a later release for the remaining domains.

9.1.4 Handling of SPI frames

The transport protocol involves several actions like synchronization, frame bundling, flow control and error handling. The previous two paragraphs described the grammar of a SPI frame and data segment. This paragraph describes these actions in more detail by following the possible states of the command dispatcher. Each state transition is described in text and graphical form.

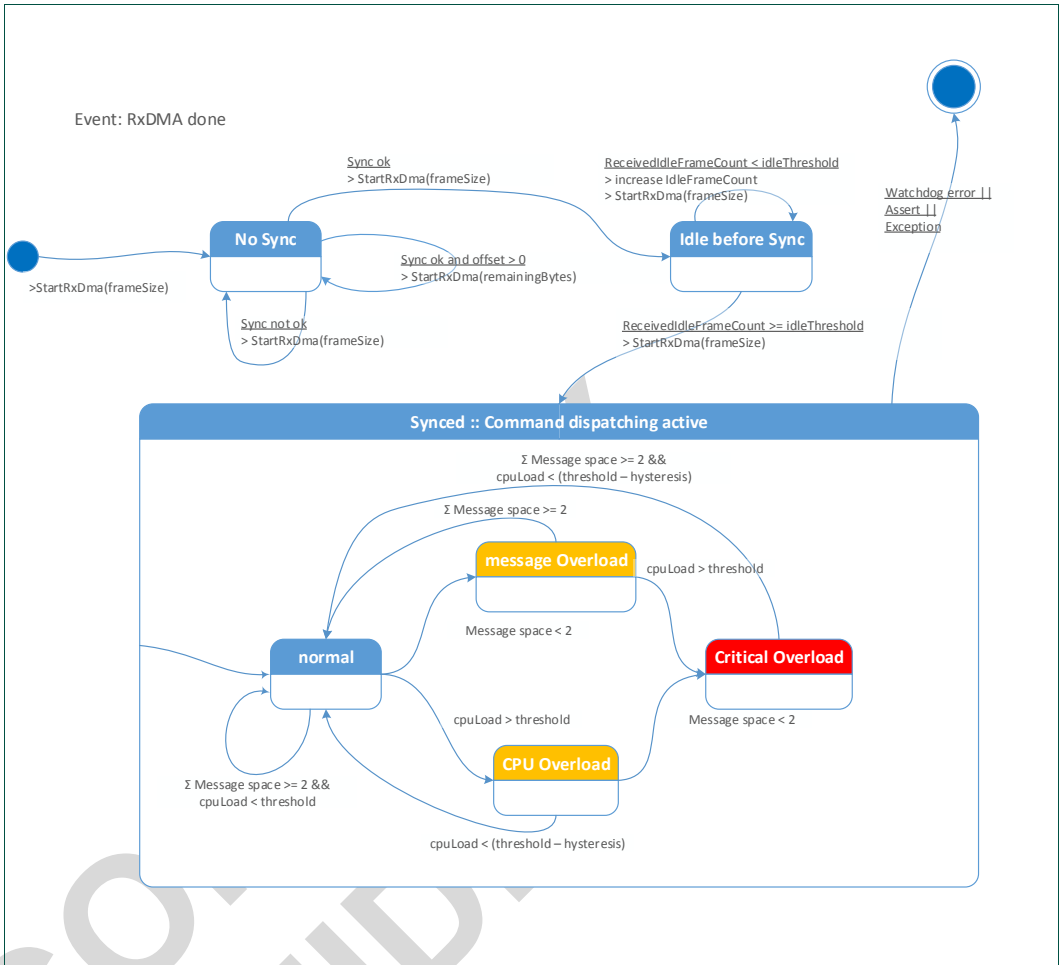

Fig 7. Target side state machine

Before discussing the states and the transitions, the template for the physical sequence of events is given.

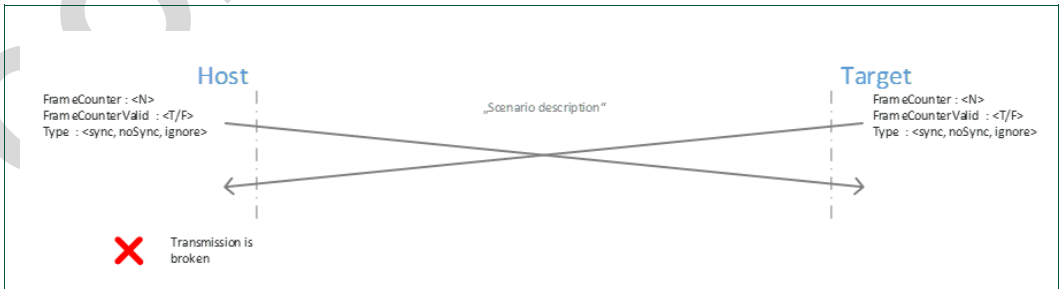

Fig 8. Template for physical sequence of events

In this template, the exchange of two SPI frames is depicted. For most scenarios, the FrameCounter and Type of the frame is enough. Typically, the target has scheduled a frame for exchange while processing the just received frame. This means that the result of this frame is transferred two frames later.
The number of exchange actions can be extended if the scenario includes multiple frames. If something goes wrong the cross marks the broken frame.

The state machine is triggered by the DMA transaction complete event. The actual transport of messages takes place in the Synced state. The method to get in that state is described below.

9.1.4.1 No Sync state

Upon start of the system there is no sync from the target point of view. Since the host can start reading from the SPI communication channel before the target has initialized the transaction, the initial frame could not start at the expected location. However, there is a chance that he host starts after the target has setup the SPI communication channel. Therefore, the target starts with a valid frame without the sync bit set. The host is typically reading smaller frames to search for the *FrameSync* pattern. If the *FrameSync* pattern is found the remaining bytes can be read.

Now, from the FrameSync to the end of frame the CRC should be correct. The target received only zeros. Which means that after the host has read enough data to receive a correct frame, the target did not receive a valid frame. In fact, it received nothing; the target starts another txDma with a frame without a valid FrameCounter and cleared Synced flag. The received frame does contain a valid FrameSync pattern and the CRC is valid. If the FrameSync was on the start of the receive buffer, a valid sync is established.

The next frame becomes a valid *FrameCounter*, the *FrameCounterValid* and *Synced* flag set to one.

The next state will be “*idle before sync*”.

Below the sequence of exchanged frames are given:

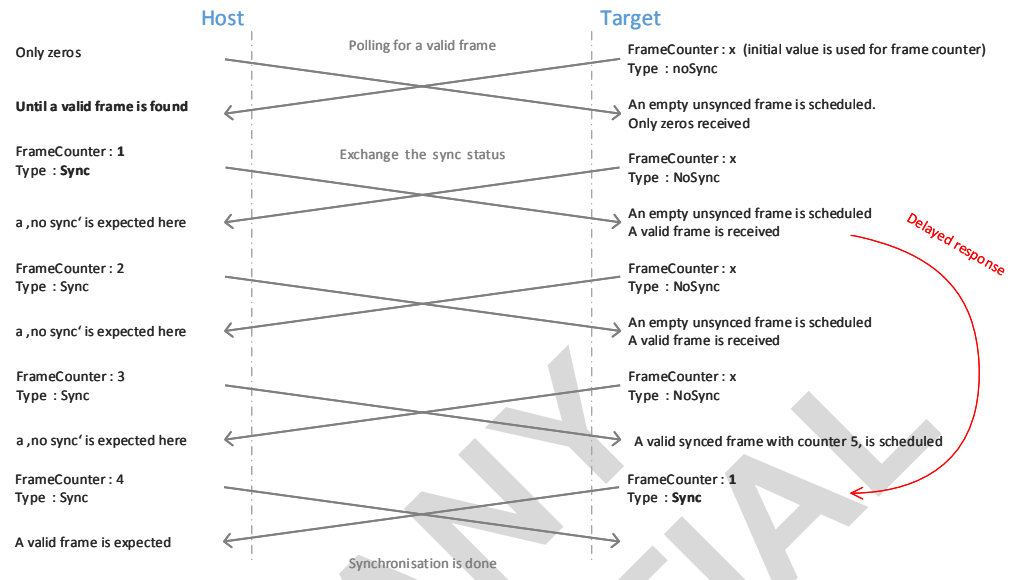

Fig 9. Initialization sequence

The algorithm to establish a datalink between host and target can also be visualized in a flowchart as given below.

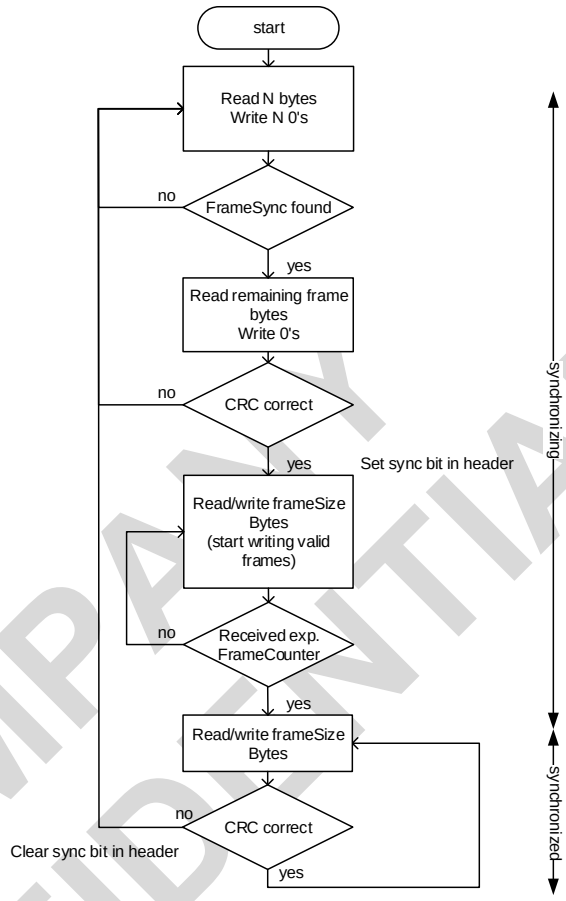

Fig 10. SPI synchronization procedure

If during operation the synchronization is lost, the host is responsible for re-establishing the datalink.

9.1.4.2 Idle before sync state

Currently, due to a hardware issue, the Target to Host communication is rather unreliable. To assure a reliable communication the target waits a number of exchange cycles before entering the synced state, where the frames may carry payload. In this state the target counts down until zero. Upon reaching zero the state will be *Synced*.

9.1.4.3 Synced state

In the synced state, the datalink is stable where actual data communication can take place. In the state machine diagram, the 'Synced' state is modeled as a super state in which a few sub-states are defined, since the connection is synchronized in a **Normal**, **overload**, **CpuOverload** and **MessageOverload** state.

Note: A re-definition of the **CpuOverload** and **MessageOverload** states is under analysis and may be subject to modification in a future release.

Within the synced state transmissions can be corrupted without losing sync. Although a retransmission is not causing a state change, it will be discussed here.

Normal

In the normal synced state, the command dispatcher is active with dispatching commands towards the sub-systems and collecting the messages from the sub-systems. As long as the CPU load is within the given limits and the distribution of the messages towards can be guaranteed.
In this state, the sub-system are messages in one or more segments dispatched towards the sub-systems. For the other direction, the sub-systems are polled by using the *protocolId* for messages to send.

a) Host to target

The dispatching loop is traversing over the segments inserted in frame. Each found segment is offered the sub-system which takes care of, building the message in case of a multi-segment message and handing it over the sub-system. The sub-systems inform, at the last segment, the available space for more messages which is reflected in the LoadStatus field.

b) Target to host

The polling loop is going round-robin along the protocolId's. This means that the lower ID has priority over higher ID's. The sequence is chosen carefully to support reliable error handling and short response times for the protocols requiring that. Digital radio protocols have longer messages with low latency.

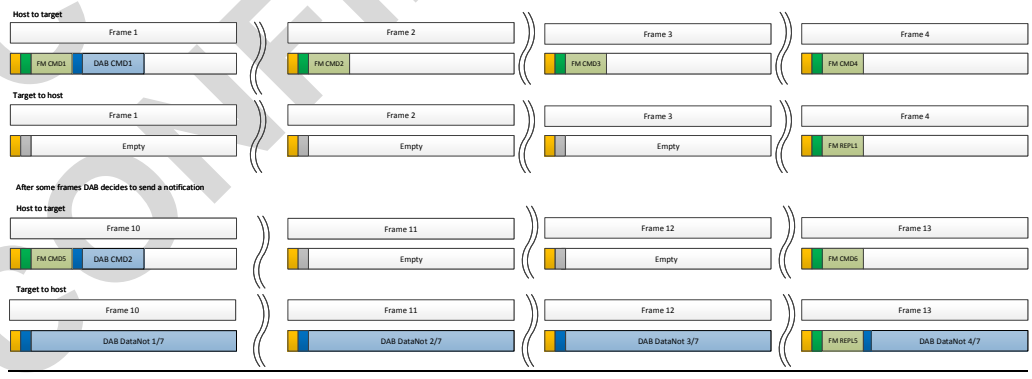

Fig 11. Dispatch and poll example

The example above depicts how the frames are built up with commands where at a certain moment in time a reply is placed in front of a data notification. For this reason the amount of segments needed for a message cannot be announced at the start of message.

c) CPU time guard

During the dispatching loop and polling loop, the CPU time is recorded. In a timer the accumulated time consumption is compared with a predefined threshold. If the time consumption is larger than the threshold the state becomes **cpuOverload**.

d) Message load guard

During the dispatching loop, each receiving sub-system is asked about the available space for new messages. If one of the sub-system reports a free space for less than two messages, the state becomes **messageOverload**.

These sub-states will be discussed later on. The state is reported in the target2host frame header in the *LoadStatus* field.

CpuOverload

This section may by updated as per the flow control under definition.

As soon as the accumulated CPU load over a period of N milliseconds surpasses the threshold, the CpuOverload state is entered. This state is notified in the LoadStatus field.

Note: The exact behavior in this state is still subject to research. The effects of different scenarios are still to be determined. The actions of the host are given and should be followed.

To reduce the load of the communication sub-system the host can increase the time between the frames to reduce the processing load. If the host is listening to the trigger GPIO line the CPU load can generally be lower. However during bursts of commands the load can be too much. Also in this case the time between the frames should be increased until the load status enters the normal state.
As described in the state machine, the normal state is entered when the CPU load has dropped below a lower boundary.

How such an overload case can be detected and handled is given in the sequence of events below.

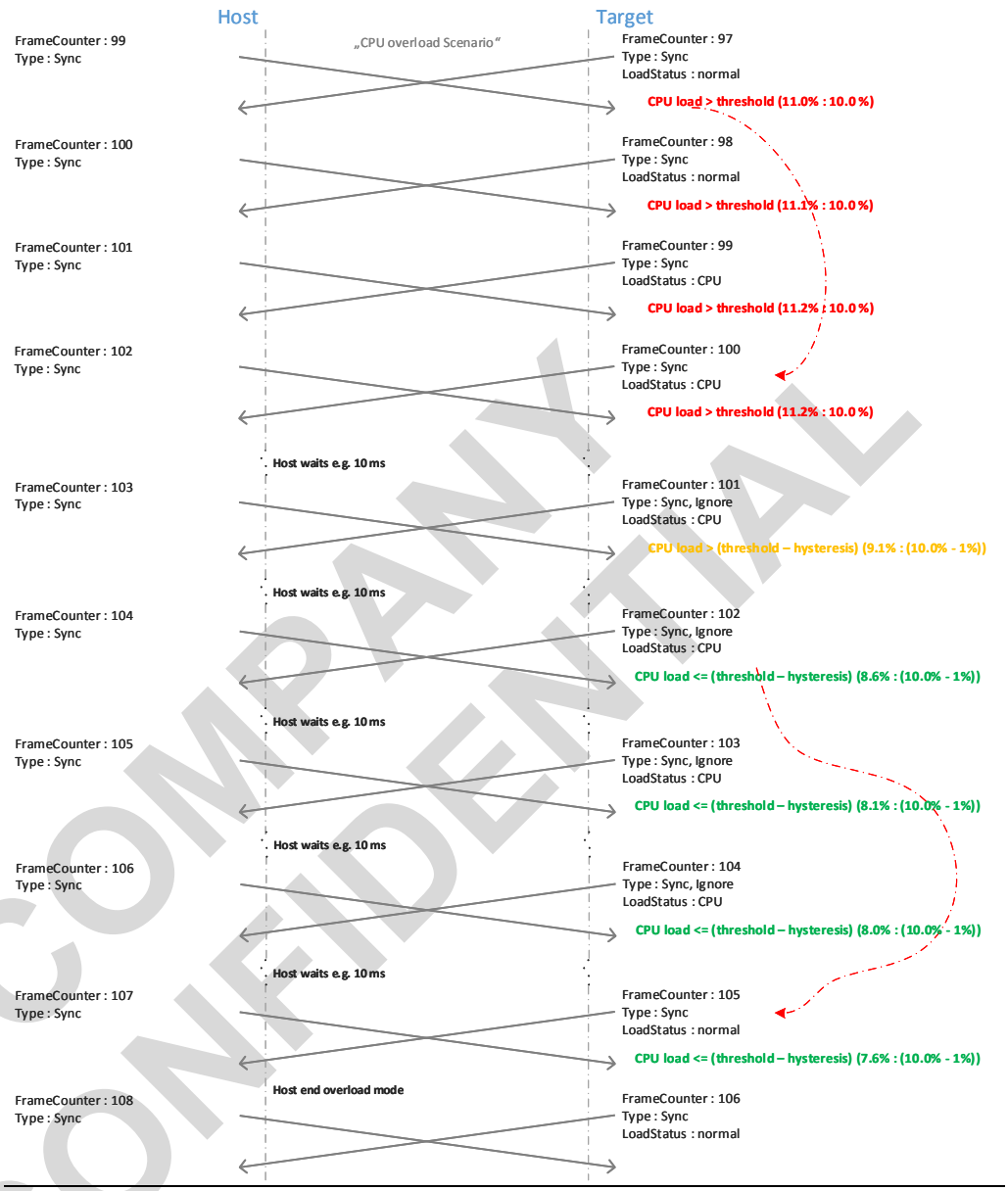

Fig 12. CPU overload scenario

Note: The values of the hysteresis and behavior in case of a host which ignores the load status is still to be defined.

MessageOverload

The message overload state is not handled in the existing firmware. Exceeding the subsystem command capacity will result in additional commands being dropped. This behavior is subject to change and will be updated as per the flow control under definition.

CriticalOverload

The critical overload state is not handled in the existing firmware. This behavior is subject to change and will be updated as per the flow control under definition.

9.1.4.4 End state after watchdog reset

If the system has reset itself due to a watchdog timeout, the host could not be informed on beforehand. The host typically discovers a CRC error. Then it could be that the system is reset or about to reset. Before the watchdog resets the system some time passes by. In this time frame of up to 100 milliseconds the system shall not reply to any command.

As soon as the primary boot loader is active it writes one word (0x0A) in the SPI FIFO's. If the host continuously reads this value it is likely to communicate with the primary bootloader and not the application anymore.

The user can confirm that the primary boot loader is active, using a ping command which replies a user defined byte. By actively sending a ping command the host can determine whether the system is reset, not responding or in application mode.

A ping command has the following sequence.

Host to target: 0x0A 0xFF 0x55 0xAA 0xFF <*userbyte*>

Target to host: <*userbyte*>

From a transport protocol, the synchronization starts over again after booting the application.

9.1.4.5 Target triggered mode

In target triggered mode the system indicates the presence of data via a GPIO line. The target asserts this line as soon as at least one message is present to be transferred. It remains asserted until it has generated an empty frame. Typically, the target triggers the host in case of a notification[6] which needs to be send to the host.

As soon as the target has asserted the GPIO, the host can start exchanging a SPI frame. The first frame is expected to be empty, whereon the next frame will be filled with messages. The host shall continue to exchange the frames at least until it has nothing to send and receives an empty frame.

9.1.4.6 Transport protocol handling

In case of lost data from target to host or vice versa, the transport protocol allows a request for retransmission of a previous sent frame. The frame in which the request is send shall be an ignore frame containing no valid sequence number and a resend request only. The resend request is put in a DataLinkSegment for the TransportProtocol protocolId. See Table 9 resend request message as defined below:

[6] Notification is a message which is send on initiative by the target.

The TransportProtocol_ResendRequestMsg message to be sent is a resend request message as defined below:

Table 9. Syntax of Resend Request message

Syntax	Number of bits	Type
TransportProtocol_ResendRequestMsg {		
ActionId	8	uimsbf
RequestedFrameCounterValid	1	bslbf
RequestedFrameCounter	7	uimsbf
}		

ActionId: A resend request has action id 0x1.

RequestedFrameCounterValid: A frame to be resent has always a valid FrameCounter. Thus value 1.

RequestedFrameCounter: The FrameCounter of the frame which has been sent before and requested to be sent again. In case of an invalid or expired frame, the target shall report a sync loss. The depth in which frames can be requested depends on the TX buffer size which can be defined via a soft configuration. Default is 8 frames deep.

The sequence of frames where a corrupted frame has been received by the host is shown hereafter.

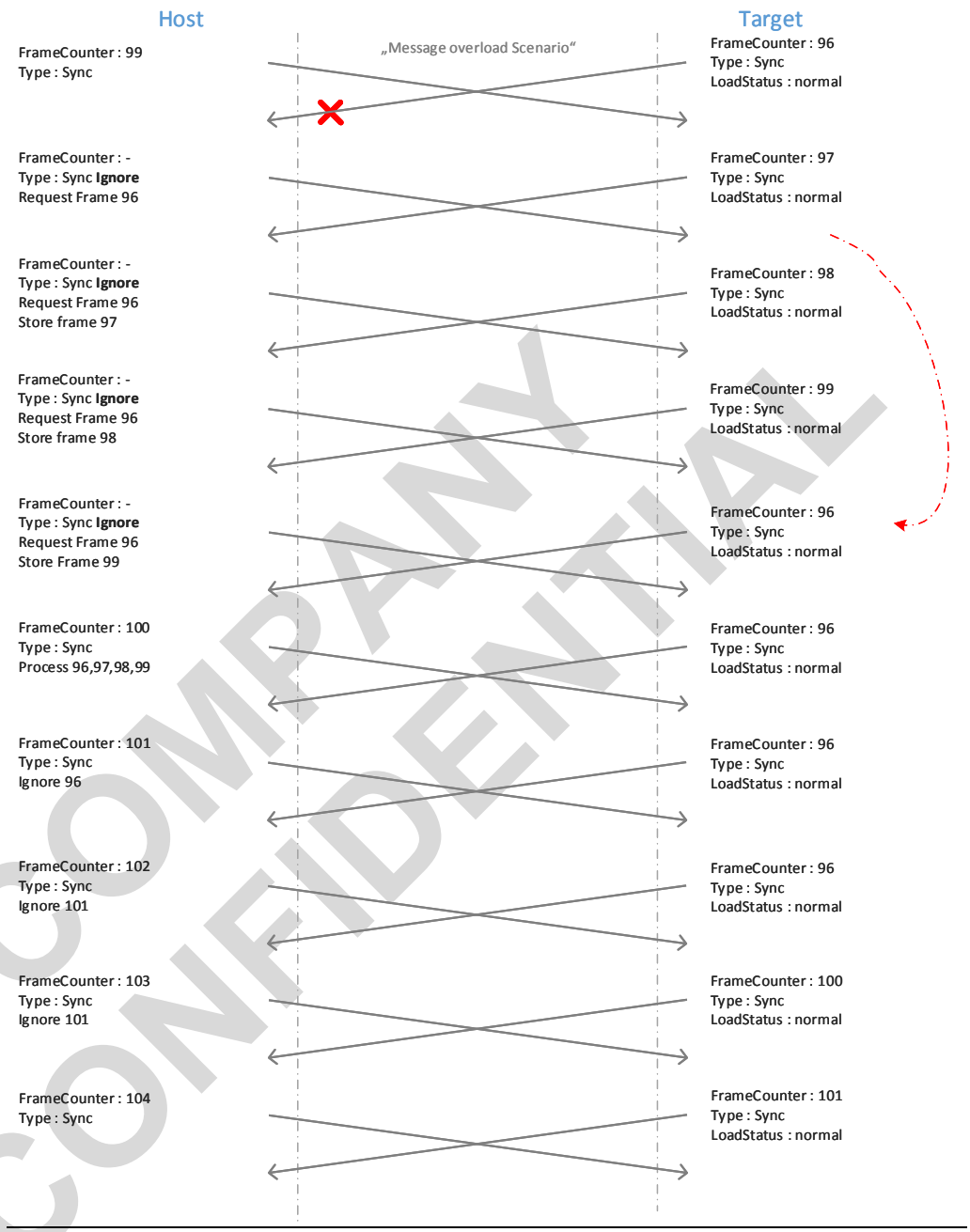

Fig 13. Sequence of frames after receiving a corrupted frame

9.1.5 Examples

To clarify the behavior of the data link layer in more detail a few scenarios are worked out in more detail.

Starting from a simple to a complex scenario the meaning and behavior of the variables are explained.

9.1.5.1 Simple command and reply

The host sends a DAB command to the digital radio sub-system using the SAF400x transport protocol. Internally, the field *protocolId* (see Table 6) is parsed and the command is routed to the appropriate sub-system. In this particular example, a reply is generated by the DAB stack. This reply can, internally, be generated, at the earliest, in the frame after which the command was received. It can be transmitted to the host, at the earliest, one frame later. This is illustrated in Fig 14.

The reply to the DAB command received in Frame 1 is sent back to the host in Frame 3.

9.1.5.2 Long message from target to host

The host has configured the target such that the DAB stack forwards back large data messages. If a data message is larger than can be accommodated in one frame, it will be split over multiple frames. This is illustrated in Fig 15.

The fields *FirstSegment* and *LastSegment* (see Table 6) to indicate the start and the end segment of the message.

9.1.5.3 Long message from target to host with short indications interleaved

Again, the host has configured the target such that the DAB stack forwards back large data messages as in the previous example, but another sub-system, such as FM, also has a short notification to send. The FM short notification has a higher priority based on the *protocolId* value. The FM notification will be inserted at the start of a frame, leaving the remainder of the frame to be used for the ongoing transmission of the large DAB message. This is illustrated in Fig 16.

The *DataLinkSegment* header of the short FM notification will use the field *MoreFollowing* (see Table 6) to indicate that there are more segments in the same frame. The *FirstSegment* and *LastSegment* fields of the FM notification will indicate that it is the first and last segment of the notification, while the *FirstSegment* and *LastSegment* fields of the DAB message will indicate that it is a longer message spread over multiple frames.

9.1.5.4 Long message from target to host interleaved with a short command reply

This example combines the examples in 9.1.5.1 and 9.1.5.2. The target has a large DAB message to send, but also receives a DAB command for which it has to send a reply back to the host. The DAB command is issued with a higher priority than the DAB notification has been configured for, which results in the DAB reply being inserted at the start of a frame, with the remainder of the frame used to transmit part of the DAB notification. This is illustrated in Fig 17.

The MoreFollowing field in the DAB reply *DataLinkSegment* header will indicate that there is another segment in this frame. In this specific example, both segments are the last for their respective messages, and that will be indicated by the *LastSegment* field of the *DataLinkSegment* header of each of the messages.

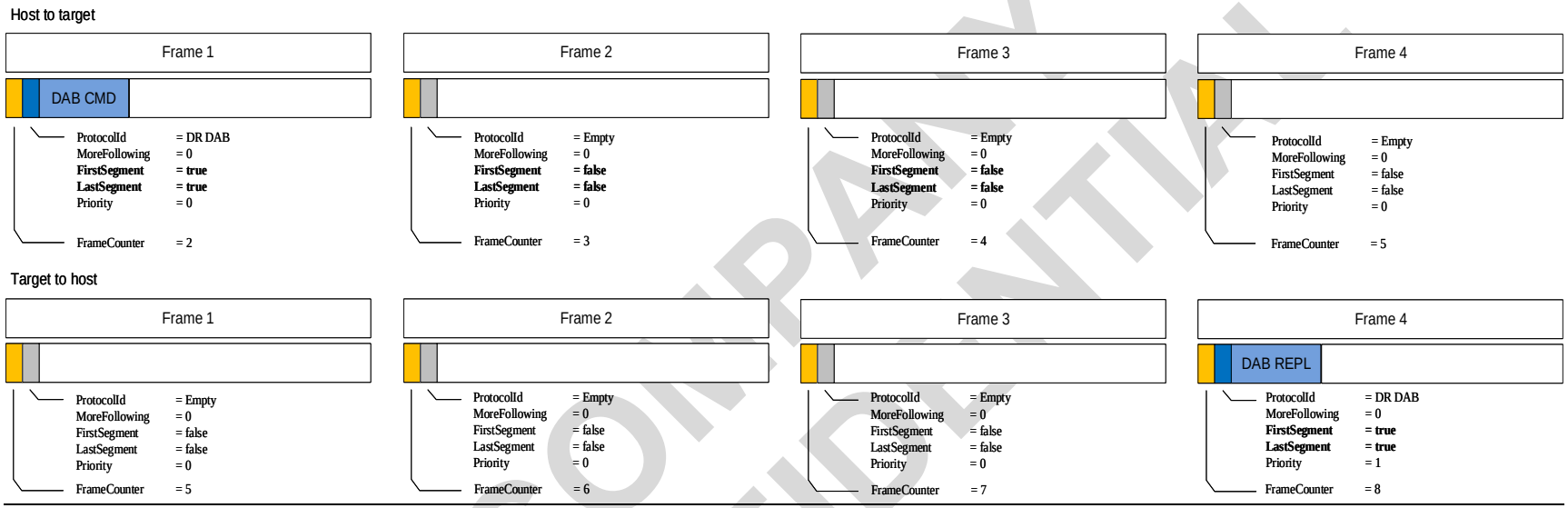

Fig 14. Simple command with reply

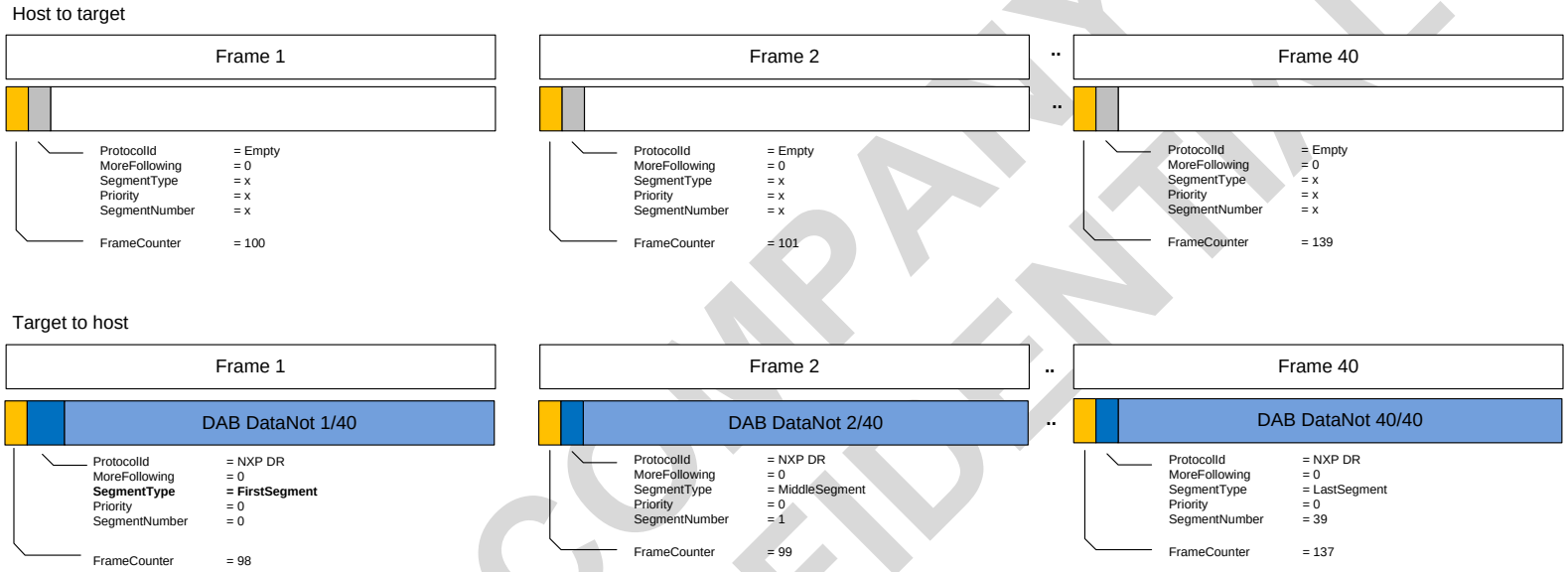

Fig 15. Long message to the host

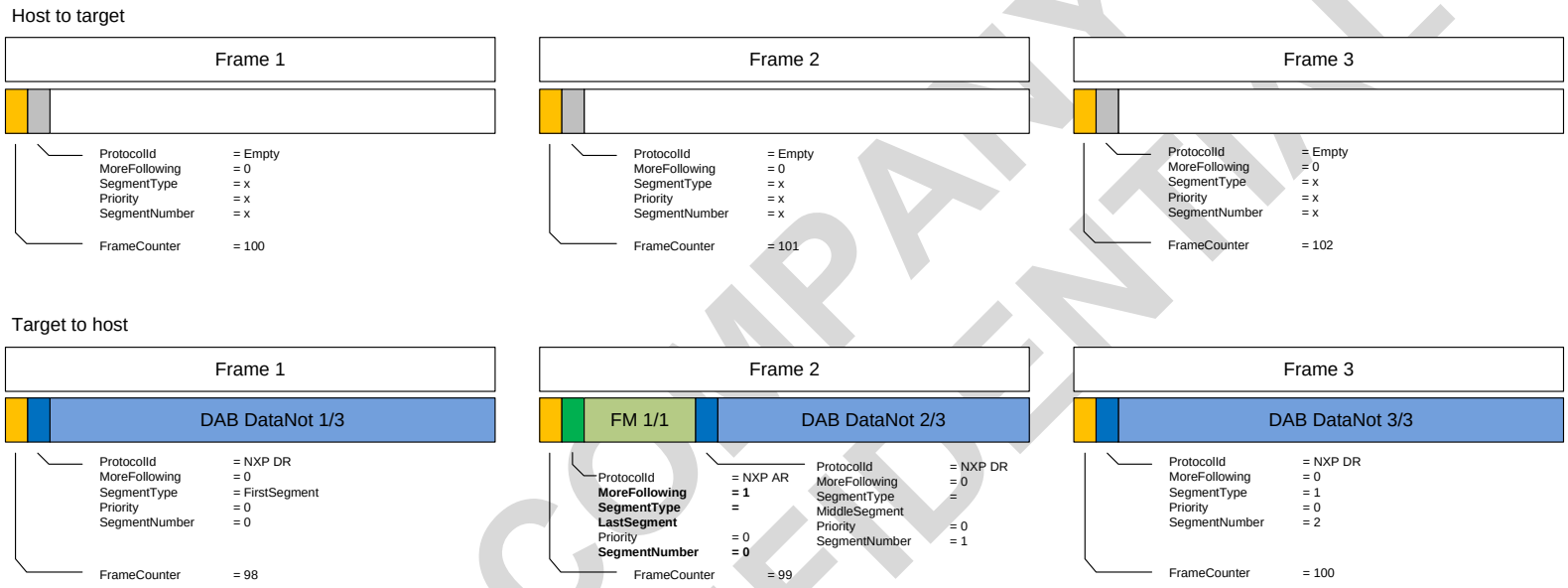

Fig 16. Large message with short notification

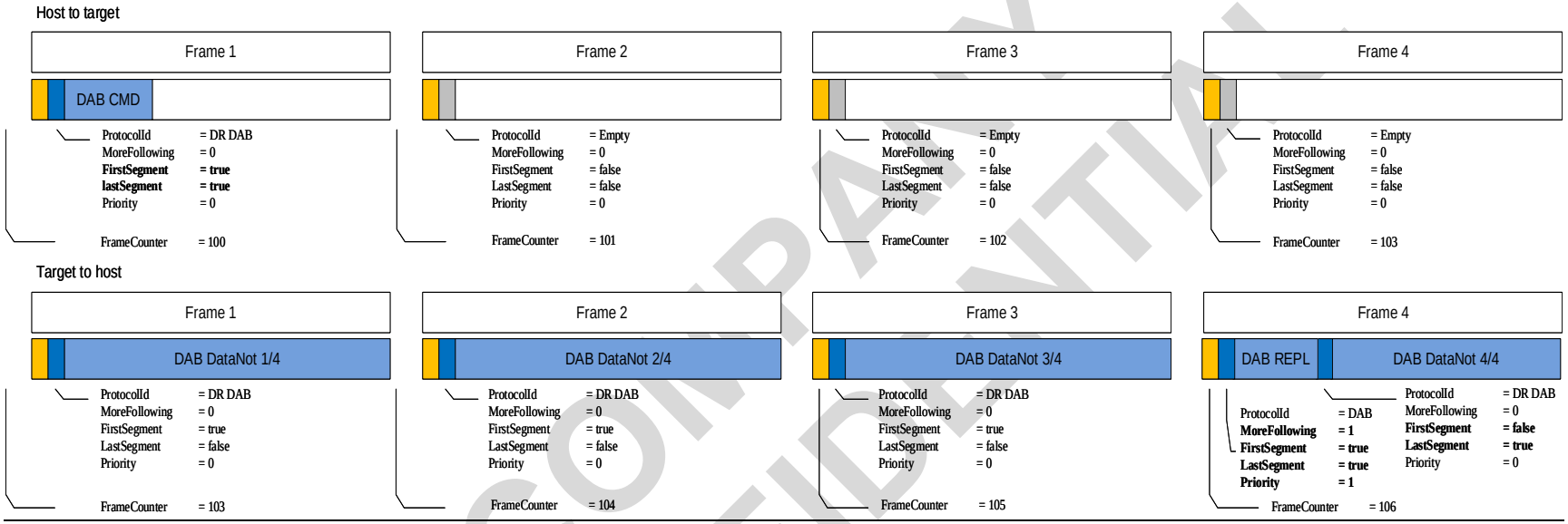

Fig 17. Large message with command reply

9.2 Example of SPI traffic between host and SAF400x

This section shows an example of a set of messages between host and SAF400x that illustrate a minimum set up to configure the SAF400x to output FM audio as example for radio developers.

1- SAF400x to host: Up Notification

```
B6 49 84 83 E1 00 04 00 15 89 00 66 00 05 80 00 F3 00 01
<Padding> <CRC>
```

- **FrameSync:** `0xB649`
- **FrameCounterValid** and **FrameValid:** `0x84`
  - **FrameCounterValid:** `0b1`
  - **FrameCounter:** `0b0000100`
- **Synced**, **Ignore**, **LoadStatus** and **FrameLength**: 0x83
  - **Synced:** `0b1`
  - **Ignore:** `0b0`
  - **LoadStatus:** `0b00`
  - **FrameLength:** `0b011`
- **DataLinkSegment:** `0xE1000400158900660005800 0F30001`
- **Padding**: `n times 0x00`
- **CRC**: `2 bytes`

Data Link Segment 1: `0xE1000400158900`

- **MoreFollowing**, **FirstSegment**, **LastSegment** and **ProtocolId**: `0xE1`
  - **MoreFollowing:** `0b1`
  - **FirstSegment:** `0b1`
  - **LastSegment:** `0b1`
  - **ProtocolId:** `0b00001 -. Device Control`
- **Encrypted**, **Priority**, **SegmentLength**: `0x0004`
  - **Encrypted:** `0b0`
  - **Priority:** `0b00`
  - **SegmentLength:** `0x004`
- **payloadData**: `0x00158900`
  - **opcode**: `0x0015 – Up_not`
  - **analogRadio_started**: `0b1`
  - **standardAudio_started**: `0b1`
  - **advancedAudio_started**: `0b0`

        - **digitalRadio_started**: `0b1`

Data Link Segment 2: `6600058000F30001`

- **MoreFollowing**, **FirstSegment**, **LastSegment** and **ProtocolId**: `0x66`
    - **MoreFollowing:** `0b0`
    - **FirstSegment:** `0b1`
    - **LastSegment:** `0b1`
    - **ProtocolId:** `0b00110 -. DAB Digital Radio`
- **Encrypted**, **Priority**, **SegmentLength**: `0x0005`
    - **Encrypted:** `0b0`
    - **Priority:** `0b00`
    - **SegmentLength:** `0x005`
- **payloadData**: `0x8000F30001`
    - **USN**: `0x80`
    - **DabApiMsg**: `0x00F30001`
        - **SystemHandle**: `0x00`
        - **OpCode**: `0xF3 - Up_not`
        - **Majorversion**: `0x00`
        - **Minorversion**: `0x01`

2- Host to SAF400x: Get HW Information

```
B6 49 87 83 61 00 02 00 02 <Padding> <CRC>
```

- **FrameSync:** `0xB649`
- **FrameCounterValid** and **FrameValid:** `0x87`
    - **FrameCounterValid:** `0b1`
    - **FrameCounter:** `0b0000111`
- **Synced**, **Ignore**, **LoadStatus** and **FrameLength**: 0x83
    - **Synced:** `0b1`
    - **Ignore:** `0b0`
    - **LoadStatus:** `0b00`
    - **FrameLength:** `0b011`
- **DataLinkSegment:** `0x610002`
- **Padding**: `n times 0x00`
- **CRC**: `2 bytes`

Data Link Segment: `0x6100020002`

- **MoreFollowing**, **FirstSegment**, **LastSegment** and **ProtocolId**: `0x61`
  - **MoreFollowing:** `0b0`
  - **FirstSegment:** `0b1`
  - **LastSegment:** `0b1`
  - **ProtocolId:** `0b00001 -. Device Control`
- **Encrypted**, **Priority**, **SegmentLength**: `0x0002`
  - **Encrypted:** `0b0`
  - **Priority:** `0b00`
  - **SegmentLength:** `0x002`
- **payloadData**: `0x0002`
  - **opcode**: `0x0002`

3- SAF400x to host: HW Information reply

```
B6 49 87 83 61 00 0A 00 02 00 00 40 00 01 00 01
01 <Padding> <CRC>
```

- **FrameSync:** `0xB649`
- **FrameCounterValid** and **FrameValid:** `0x87`
  - **FrameCounterValid:** `0b1`
  - **FrameCounter:** `0b0000111`
- **Synced**, **Ignore**, **LoadStatus** and **FrameLength**: 0x83
  - **Synced:** `0b1`
  - **Ignore:** `0b0`
  - **LoadStatus:** `0b00`
  - **FrameLength:** `0b011`

DataLinkSegment: `0x61000A00020000400001000101`

- **Padding**: `n times 0x00`
- **CRC**: `2 bytes`

Data Link Segment: `0x61000A00020000400001000101`

- **MoreFollowing**, **FirstSegment**, **LastSegment** and **ProtocolId**: `0x61`
  - **MoreFollowing:** `0b0`
  - **FirstSegment:** `0b1`

        - **LastSegment:** `0b1`
        - **ProtocolId:** `0b00001 -. Device Control`
    - **Encrypted**, **Priority**, **SegmentLength**: `0x000A`
        - **Encrypted:** `0b0`
        - **Priority:** `0b00`
        - **SegmentLength:** `0x00A`
    - **payloadData**: `0x00020000400001000101`
        - **opcode**: `0x0002 - GetHWinformation_repl`
        - **returnCode**: `0x00`
        - **variant**: `0x4000`
        - **engineeringType**: `0x01`
        - **package**: `0x00`
        - **tunerSerie**: `0x01`
        - **icVersion**: `0x01`

4- Host to SAF400x: Audio unmute

`B6 49 B2 83 62 00 08 08 00 01 10 9B 01 7F F0 <Padding> <CRC>`

- **FrameSync:** `0xB649`
- **FrameCounterValid** and **FrameValid:** `0x87`
    - **FrameCounterValid:** `0b1`
    - **FrameCounter:** `0b0110010`
- **Synced**, **Ignore**, **LoadStatus** and **FrameLength**: 0x83
    - **Synced:** `0b1`
    - **Ignore:** `0b0`
    - **LoadStatus:** `0b00`
    - **FrameLength:** `0b011`

DataLinkSegment: `0x620008080001109B017FF0`

- **Padding**: `n times 0x00`
- **CRC**: `2 bytes`

Data Link Segment: `0x620008080001109B017FF0`

- **MoreFollowing**, **FirstSegment**, **LastSegment** and **ProtocolId**: `0x62`
    - **MoreFollowing:** `0b0`
    - **FirstSegment:** `0b1`

- **LastSegment:** `0b1`
- **ProtocolId:** `0b00002 -. Audio`

- **Encrypted**, **Priority**, **SegmentLength**: `0x0008`
  - **Encrypted:** `0b0`
  - **Priority:** `0b00`
  - **SegmentLength:** `0x008`
- **payloadData**: `0x080001109B017FF0`
  - **opcode**: `0x0800 - APCSetSAPMemory_cmd`
  - **DiscreteAddressesCount**: `0b0001`
  - **DataType**: `0b0001 - Y - Memory Single Precision Data`
  - **DestinationAddress**: `0x09B - SAP_Mute_Y_S_P`
  - **DataLength**: `0b0001`
  - **Data**: `0x7FF0 - Unmute`

10. Tuner resources and resource management

10.1 Tuner configuration

The SAF400x family firmware does not include a dedicated sub-system to manage tuners, tuner interfaces, and tuner processing chains. The tuner resources are controlled by the analog/digital radio sub-systems that make use of them. An internal central tuner resource management entity then ensures that the same component is never allocated more than once.

A typical radio with a combination of SAF400x/SAF777x/TEF7100x ICs will often receive more than one service at a time, do a background scan in parallel, and sometimes receive multiple radio standards in parallel. Under such conditions, a wideband frontend usually caters to more than one turbo tuner.

By combining the internal wideband tuners (WBTs), or an external tuner with the internal turbo tuners, a wide range of use cases can be covered. Fig 18 depicts how the resources available in a single SAF400x IC are available through the SAF400x system API. The figure also shows the API commands to handle each resource.

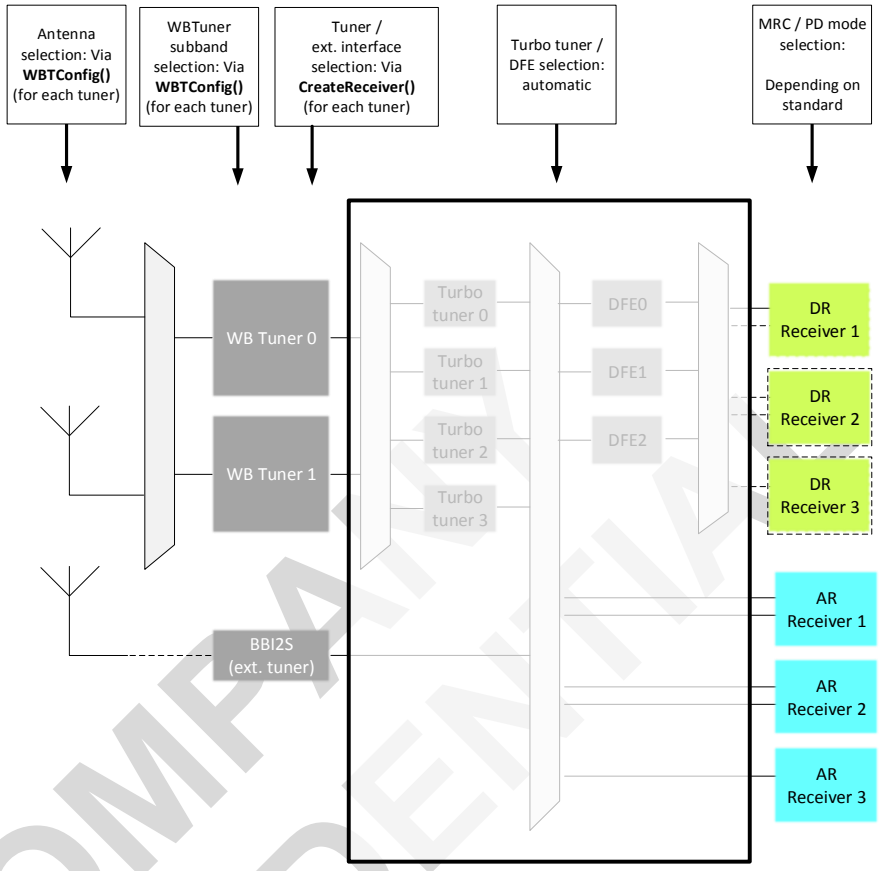

Fig 18. Abstract view of WBT and receiver configuration

10.1.1 Wideband ADC

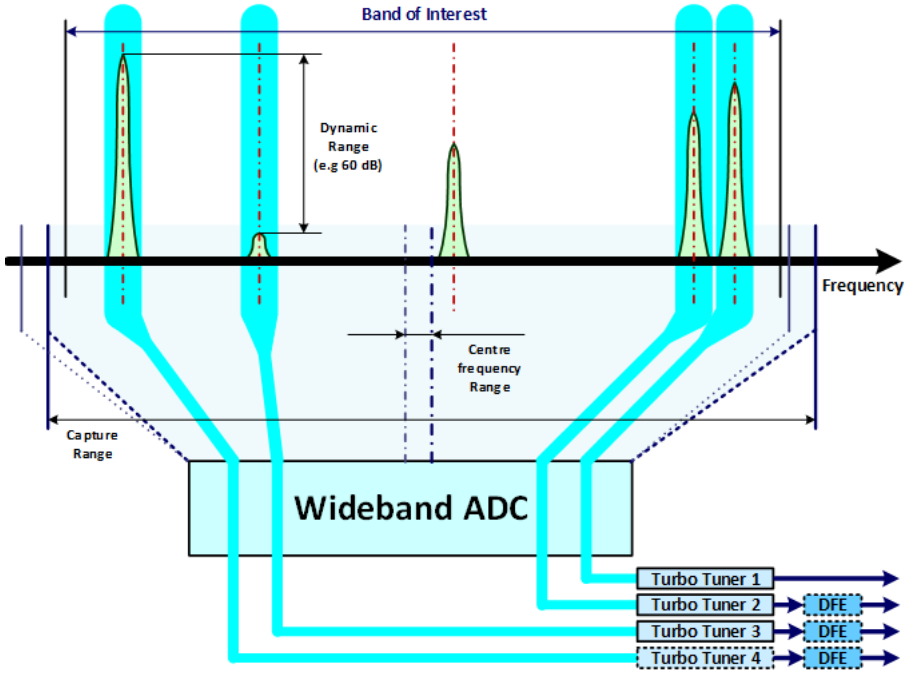

Fig 19. Wideband ADC concept

The tuners implemented by the SAF4000x family devices consist of WideBand Tuners (WBT) which allow wideband processing capturing half or the entire spectrum of a band of the respective radio standard in one go.

The RF signal from the antenna is fed to the WBT. The WBT produces a wide band radio signal which is then down sampled according to the bandwidth and sample rate of the respective radio standard using the wide band ADC (WADC). The sampling range of the WADC is limited by the capture range of the WBT.

Each device implements two WADCs, each of which can sample one sub-band via one of the AM, FM and DAB tuners. Supported sub-bands are up to 40 MHz wide, and are shown in Fig 20.

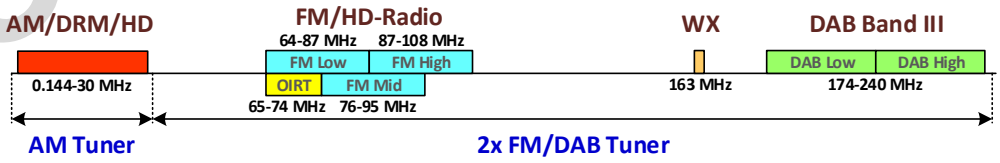

Fig 20. Sub-band settings

The WADC samples one sub-band via one of the AM, FM and DAB tuners. Out of the selected sub-bands, up to four different radio carrier frequencies are demultiplexed/demodulated using four turbo tuners. Three of these carrier frequencies are routed to the digital radio sub-system via Digital Front Ends (DFE); see Fig 19.

As the multiplexing of the wideband signal is performed in digital domain, no splitting loss occurs. As a result, each turbo tuner is able to deliver a digital signal with the same noise figure as the attached WBT. This allows processing up to four radio streams with different frequencies selected out of the two different radio bands received by the two WBTs.

10.1.2 Receiver lifecycle

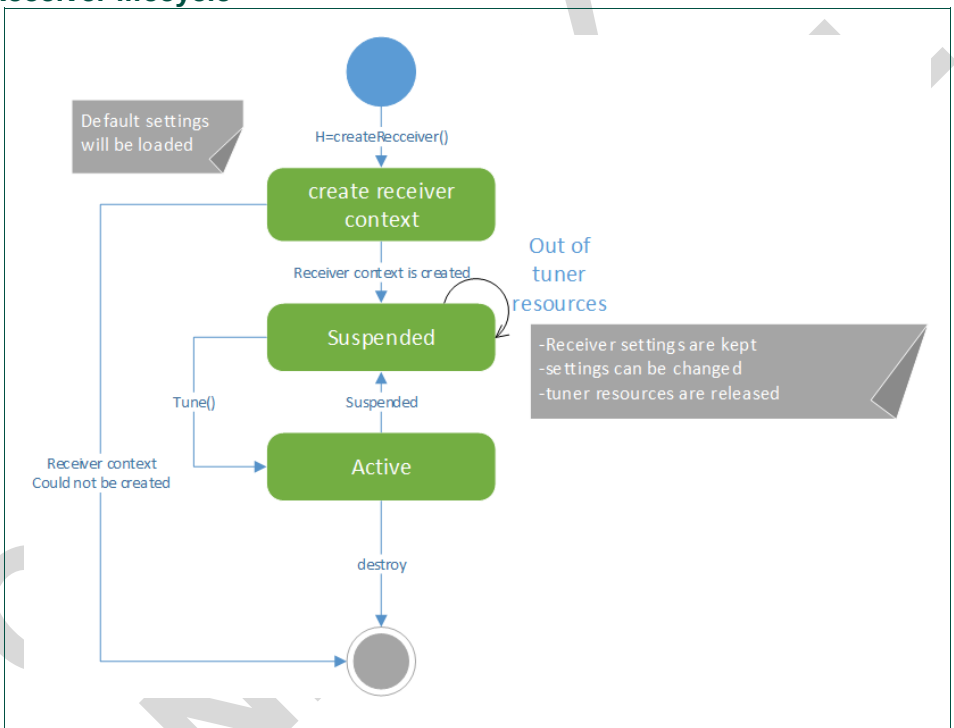

Fig 21. Receiver lifecycle

The SAF400x/SAF777x system carries out the following actions for Radio Receivers:

1. Create Receiver: creates a receiver context that can be referred to with the returned handle. Created receivers can be activated sequentially for operation in different use cases or can be activated together for parallel operation in a multi-receiver use case.
2. Tune Receiver: tunes the receivers to the chosen frequencies. They are activated and connected to the required front end resources (DFEs, turbo tuners, selected WADCs).
3. Suspend or sleep mode: puts an active receiver to sleep or suspend mode. The receiver releases its hardware resources for other analog, or digital, radio use cases. Even after putting the active receiver in to sleep mode, the API settings of the receiver are maintained for later use.

4. Destroy Receiver: destroys a receiver context releasing all claimed resources and deleting the receiver handle and all the API settings.

Note that while 'Suspended', the context of a Receiver is maintained allocated in memory. This means that the maximum amount of receivers that can be created simultaneously must include the count of the suspended receivers.

The total amount of receivers that can be allocated simultaneously are: 3xFM, 3xAM, 1xWX, 2xDAB full receivers plus 1 FIG receiver.

The overview of the maximum amount of receivers for the HD and DRM domains will be updated in a future release.

10.2 Conceptual description of a tuner configuration

The combination of different SAF400x family devices allow realizing a high number of system use cases.

The following conventions are used when describing a system use case (see Fig 22):

- A light green box depicts the CarDSP that is used (SAF400x/SAF777x/TEF710x)
- Grey boxes inside the green box represent the two wideband tuners (in the case of SAF400x or SAF777x)
- Boxes inside the grey box depict turbo tuners, which are connected to the respective wideband tuner. Depending on the radio standard, different colors are used for this box
- The text inside the colored box describes the radio standard used (e.g. FM), and to which antenna the tuner is connected.

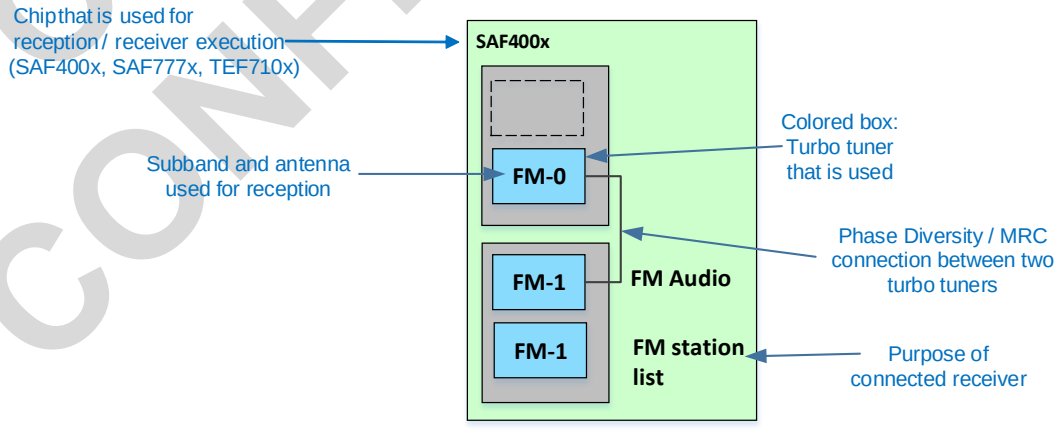

Fig 22. Tuner configuration illustration

The use case representation in Fig 22 serves as the basic building block for all the supported system use cases, which are given in the next sections.

10.3 Conceptual description of the WBT setting

Fig 23 shows the different sub-bands that are supported by the SAF400x and SAF777x WBTs. In order to depict the sub-bands, along with their respective WBT settings, in a use case setup, this document uses the following graphical notation:

- Sub-bands relevant for the use case are represented by green-framed grey boxes. In case the sub-band is currently not covered by a WBT, the box frame is dashed, and the box is left empty
- WBTs are represented by grey arrows below the respective sub-band boxes
- Receivers and their allocated WBTs are represented by small boxes. The color depends on the reception standard used.

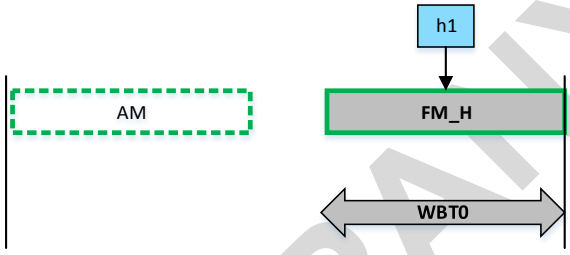

Fig 23. WBT setting illustration

10.4 Use case setup and use case transitions

The following section provides an overview of the command sequences required to set up specific example use cases.

10.4.1 Setup of triple FM reception

The following sequence shows an example of how to set up triple FM reception on WBT0.

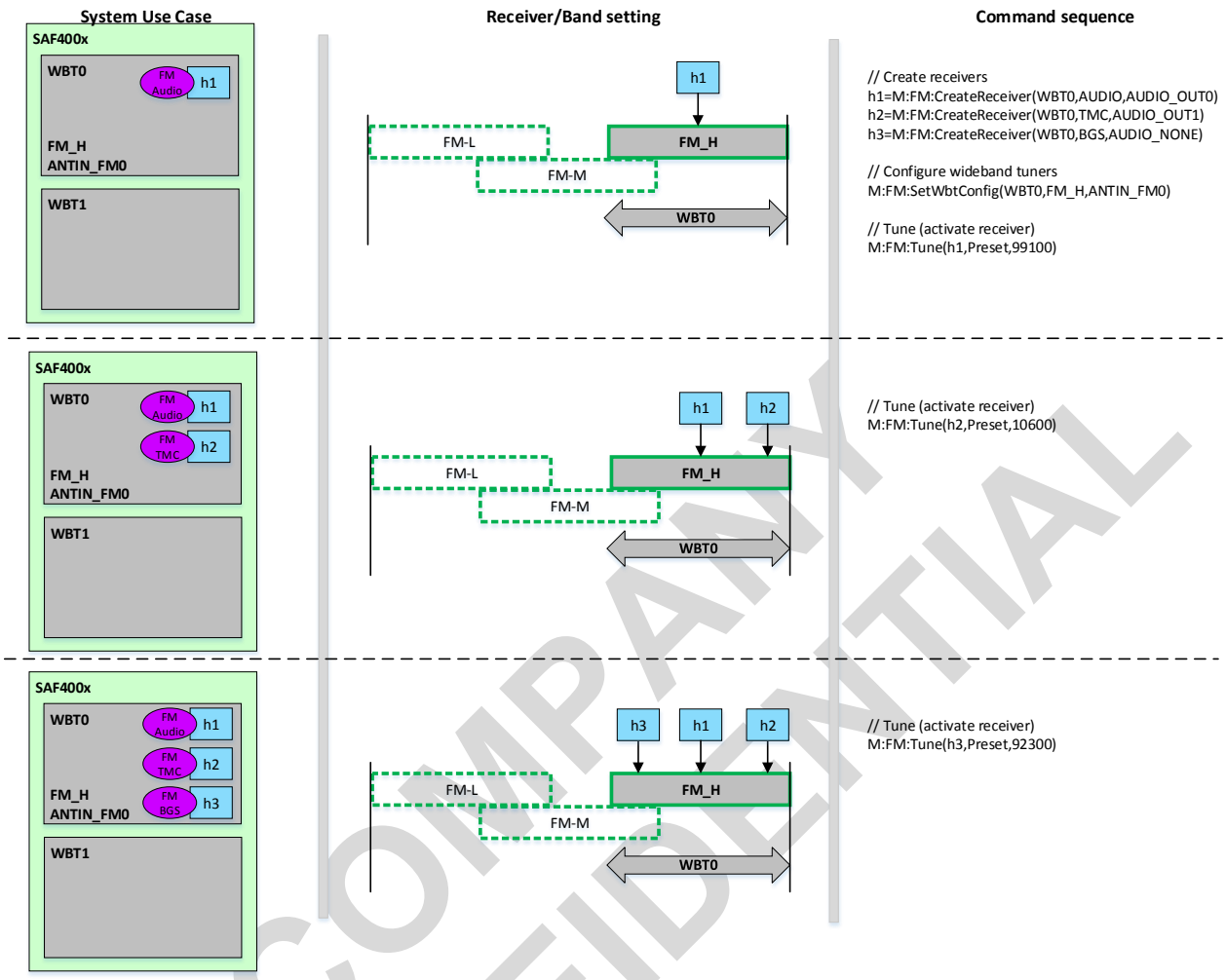

Fig 24. Setup of triple FM use case

In step 1, three FM receivers are created on WBT0: h1 for Audio, h2 for TMC and h3 for Background Scan. WBT0 is then configured to use antenna input FM0 and to band FM_H. And thirdly, receiver h1 (Audio) is tuned to preset 99.1MHz.

In step 2, receiver h2 (TMC) is tuned to preset 106.0MHz.

In step 3, receiver h3 (BGS) is tuned to preset 92.3MHz.

10.4.2 Switch from FM to AM

The following sequence shows an example of how to setup a switch between FM and AM.

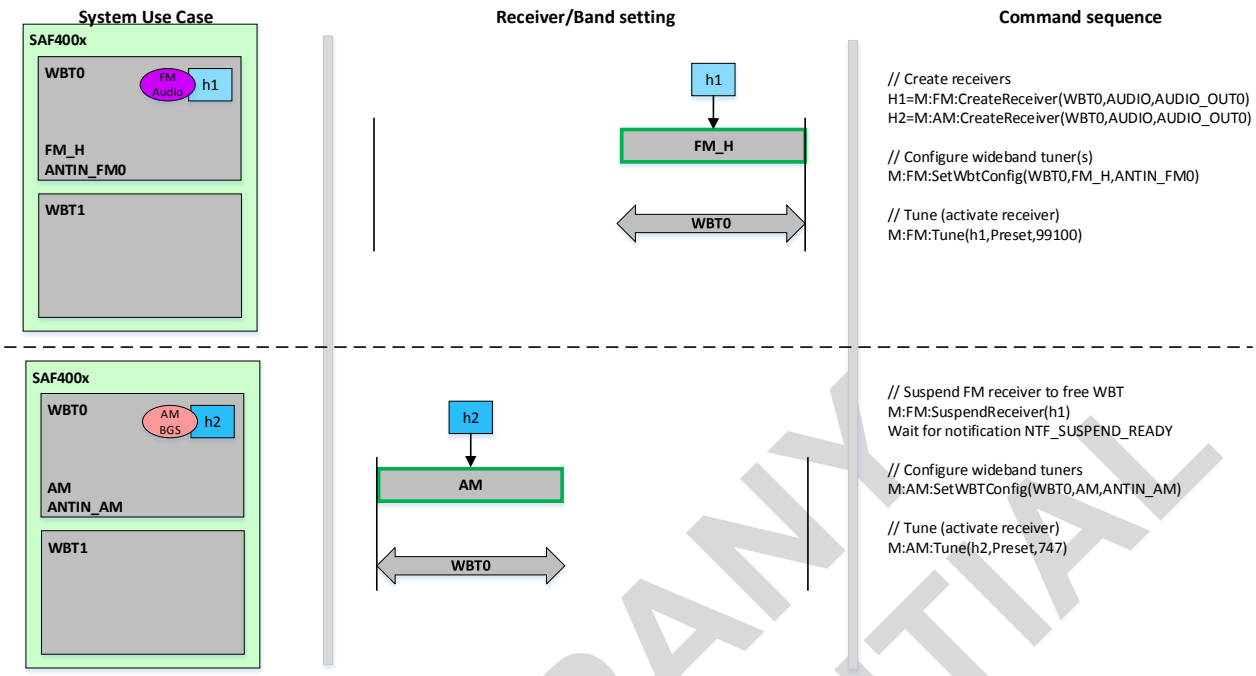

Fig 25. Switch from FM to AM use case

In step 1, one FM receiver and one AM receiver are created on WBT0: Both h1 and h2 are configured for Audio. WBT0 is then configured to use antenna input FM0 and to band FM_H. And thirdly, FM receiver h1 (Audio) is tuned to preset 99.1MHz.

In step 2, FM receiver h1 is suspended, and WBT0 is then configured to use antenna AM and to band AM. AM receiver h2 is then tuned to preset 747kHz.

10.4.3 Switch from FM to DAB

The following sequence shows an example of how to setup a switch between FM and DAB.

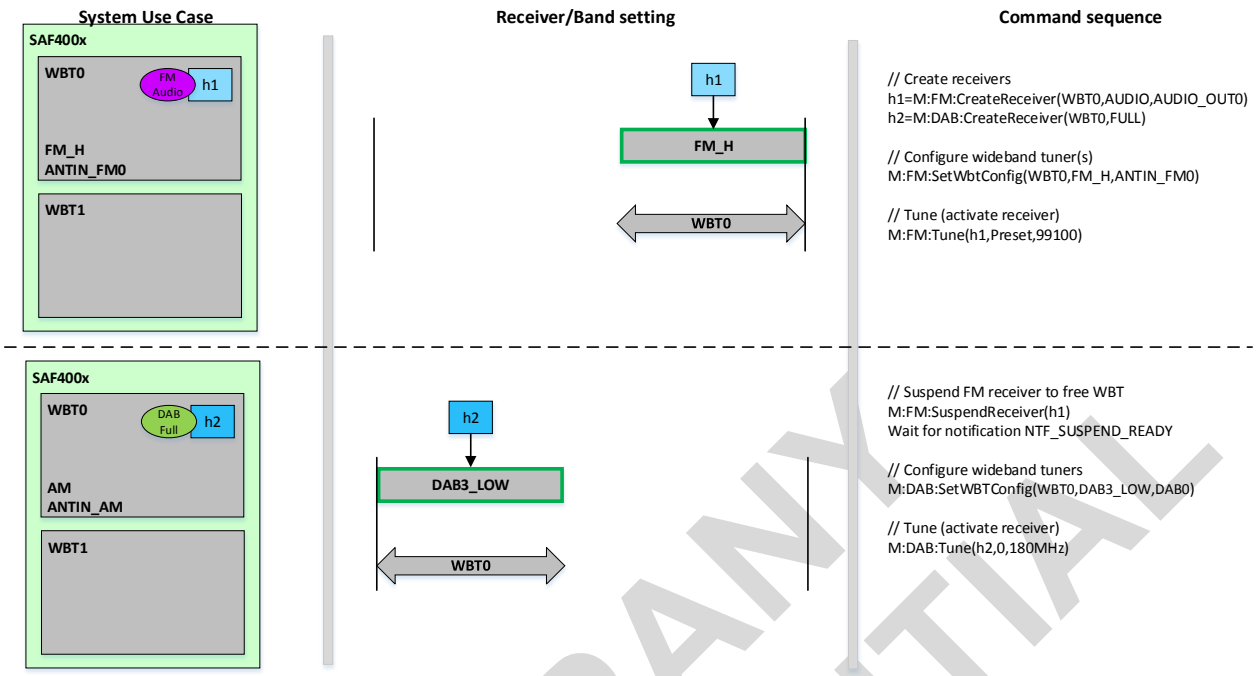

Fig 26. Switch from FM to DAB use case

In step 1, two receivers are created on WBT0: h1 is created for FM audio and h2 is configured for DAB full. WBT0 is then configured to use antenna input FM0 and to band FM_H. Thirdly, receiver h1 (Audio) is tuned to preset 99.1MHz.

In step 2, receiver h1 is suspended, and WBT0 is then configured to use antenna DAB0, and to band DAB3_LOW. DAB3 receiver h2 is then tuned to preset 180MHz.

10.4.4 FM Phase Diversity with internal tuners

The following sequence shows an example of how to setup FM Phase Diversity using the two internal WBTs.

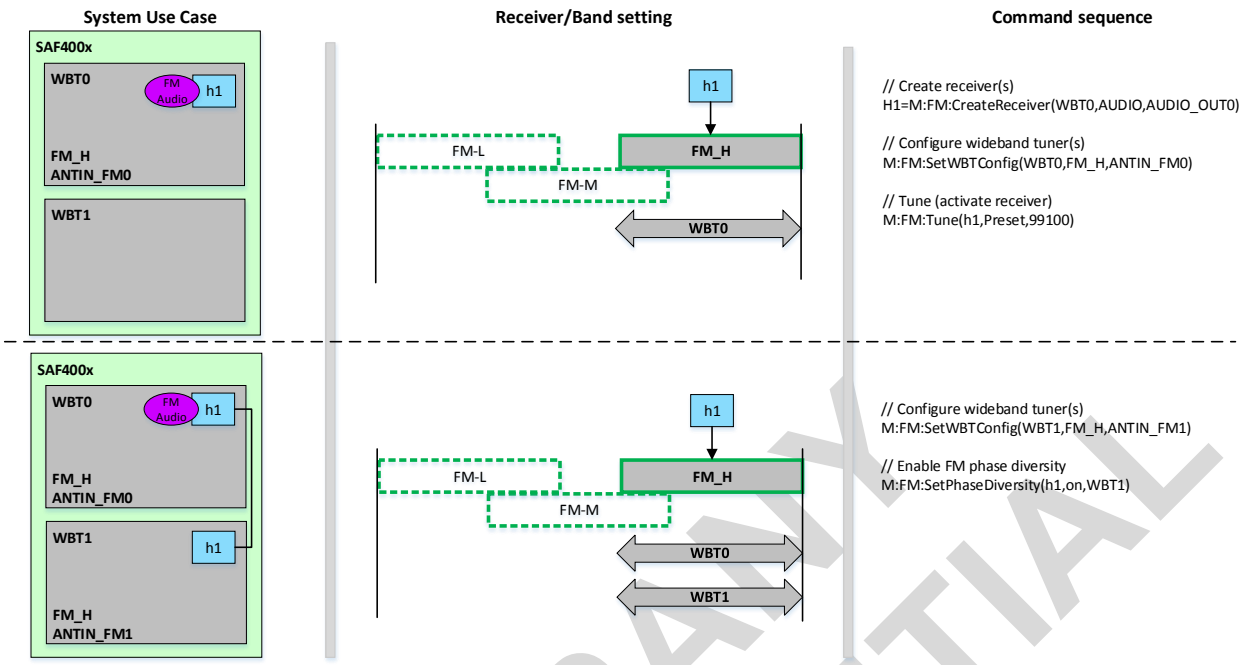

Fig 27. Setup of FM Phase Diversity using two internal tuners

In step 1, one FM receiver is created on WBT0 for Audio. WBT0 is then configured to use antenna input FM0 and to band FM_H and FM receiver h1 is tuned to preset 99.1MHz.

In step 2, WBT1 is configured to use antenna input FM1 and to band FM_H. Then FM receiver h1 is switched to FM phase diversity using the inputs from WBT0 and WBT1.

From this point on, when tuning to a new frequency in FM PD mode, a tune command to the SAF400x controls both the tuning on WBT0 and WBT1.

10.4.5 FM Phase Diversity with one internal tuner and an external tuner

The following sequence shows an example of how to setup FM Phase Diversity using an internal WBT and an external tuner. This is illustrated using a system combination of SAF400x and SAF777x.

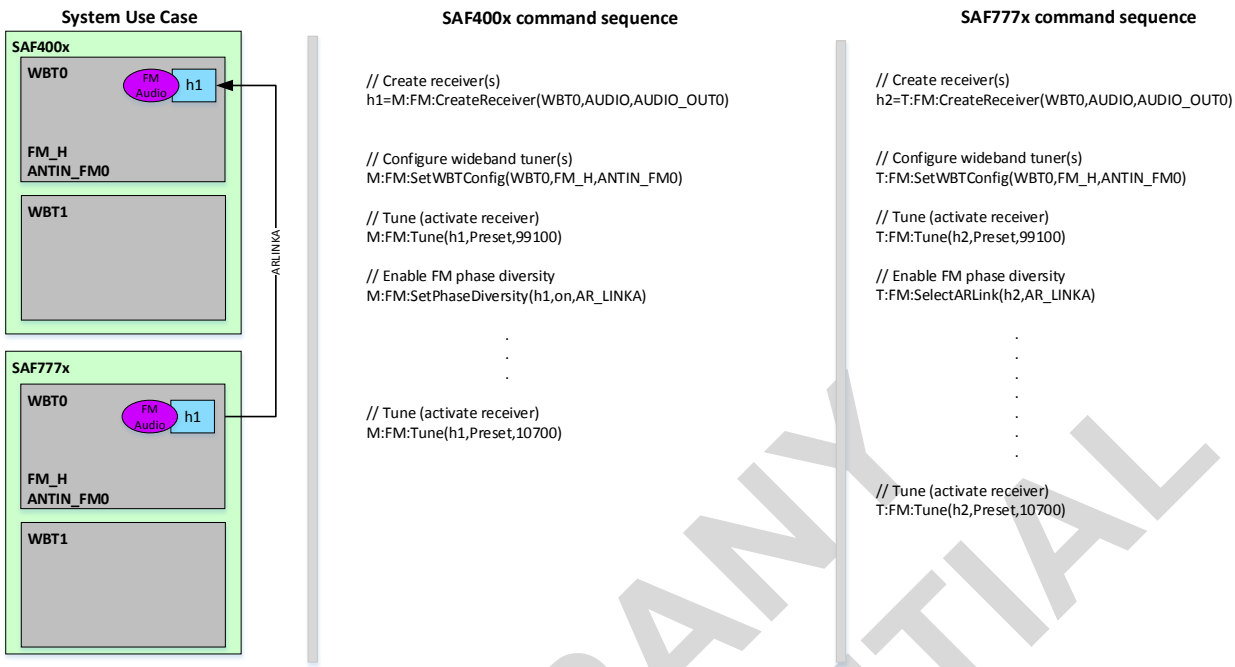

Fig 28. Setup of FM Phase Diversity using an internal tuner and an external tuner

The host must send commands to setup the reception on both devices.

On the SAF400x, one FM receiver is created on WBT0 for Audio. WBT0 is configured to use antenna input FM0 and to band FM_H. FM receiver h1 is tuned to preset 99.1MHz.

On the SAF777x, similarly, one FM receiver is created on WBT0 for Audio. WBT0 is configured to use antenna input FM0 and to band FM_H. FM receiver h2 is tuned to preset 99.1MHz.

On the SAF777x, the AR_LINKA output is connected to receiver h1 and on the SAF400x, the host switches FM receiver h2 to FM phase diversity using the internal WBT0 and the AR_LINKA input.

Note that after the AR link is setup, in order to tune to a new frequency in FM PD mode, the host must explicitly send the tune command to each of the two devices. The SAF400x will not independently tune the SAF777x (or other external tuner) device. Additionally, there is a constraint in the order in which the devices must be tuned. The device executing the Phase Diversity must be tuned first. In the case shown in Fig 28, the SAF400x must be tuned first to the new frequency before tuning the SAF777x.

10.4.6 DAB MRC with internal tuners

TBD

10.4.7 DAB MRC with one internal tuner and an external tuner

TBD

10.4.8 HD MRC with internal tuners

Refer to [SAF400x_HD_UM].

10.4.9 HD MRC with one internal tuner and an external tuner

Refer to [SAF400x_HD_UM].

10.4.10 General use case transitions

As shown in the previous sections, the use case setups and use case transitions can be de-composed into their respective sub-standards.

The particularities of these sub-standards with respect to special use cases, use case setup, and use case tear-down are described in the respective sub-system user manuals

AM: [AM_UM]

FM: [FM_UM]

WX: [WX_UM]

DAB [1]: [DAB_UM]

HD [1]: [HD_UM]

DRM [1]: [DRM_UM]

[1] If applicable for this system release. A system release will typically cover only one DR standard.

10.5 Simulcast

The Simulcast feature allows a system to switch between different reception sources (receiving the same radio station) in a seamless* way when the quality of one of the alternatives becomes better compared to the currently enabled.

To support seamless switching, the system supports the following functions:

- A circular buffer for different audio sources with an adjustable delay (currently not available)
- An alignment algorithm to determine time delay between audio sources (with optional level matching) (currently not available)
- An audio switch to switch between audio sources (with cross fade option)

In order to be able to use the Simulcast feature, the host must make sure that at least two reception sources are available with the same radio station. The host is responsible for searching for an alternative reception (analog or digital) of the same radio station as the selected radio station. The host is also responsible for monitoring the quality of both the main and the alternative reception and decide when to switch (seamlessly) to the alternative source.

A basic example scenario for using simulcast DAB-FM is as follows:

1. DAB reception must be activated and selected as the audio input source and the Host must retrieve the Service ID of a currently selected DAB service
2. The Simulcast feature on the system is reset via the SC_ResetBlend_cmd / SC_ResetBlend_repl
3. The Host scans the FM frequency band and checks the RDS info of the analog FM service to check if an analog service is available with the same Service ID as the selected DAB service.

   The audio for this service needs to be routed from the analog subsystem (in this example the FM) to the simulcast subsystem.

   This done by issuing the following Audio command:

```
APCSetASCMemory_cmd:
   1 /* CookieId */
   0x0010 /* OpCode */
   0 /* Rfa */
   1 /* DiscreteAddressesCount */
   0 /* DataType */
   0x42A /* DestinationAddress */
   0 /* Rfa */
   1 /* DataLength */
   0x000001 /* Data */
```

4. The Host instructs the Simulcast to perform correlation to determine the delay between the analog and the digital audio by issuing SC_xx_cmd / SC_xx_repl. (not yet available in the current firmware)
5. If a correct alignment has been found, the Host uses the indicated value to set the analog buffer delay. (in this example, it is assumed that analog source is timewise ahead of the digital source) (not yet available in the current firmware)
6. The host continuously monitors the digital signal quality (using e.g. RSSI, BER, SNR or Audio sync/mute states). If the quality of the analog source becomes better than that of the digital source, a seamless switch via SC_StartBlending_cmd / SC_StartBlending_repl to analog is performed. If the relation between the quality of the sources changes, a switch back to digital can need to be performed. Depending on the situation a number of steps can be repeated:
   a) When a new digital service is selected, step 1 to 6 should be repeated
   b) Steps 3 to 6 can be repeated to check for stronger FM alternatives
   c) Steps 4 to 6 can be repeated when a service is tuned to for a longer time (to make sure alignment is still OK). Note this has no impact on the performance of the rest of the system

* due to the unavailability of steps 4. and 5. in the current firmware release, the switch between sources is currently not seamless.

11. References

Table 2. References

Doc ID	Doc Title	Revision	Issue Date
[SAF400x_Boot]	SAF400x Boot User Manual	0.3	20161027
[SAF400x_Audio_UM]	SAF400x Audio User Manual	01.00	20170120
[SAF400x_Audio_Protocol]	SAF400x Audio Protocol		
[SAF400x_AM_Protocol]	SAF400x AM Protocol		
[SAF400x_FM_Protocol]	SAF400x FM Protocol		
[SAF400x_WX_Protocol]	SAF400x WX Protocol		
[SAF400x_DAB_UM]	SAF400x DAB User Manual	ER0.10.2	20170111
[SAF400x_DAB_Protocol]	SAF400x DAB Protocol		
[SAF400x_HD_UM]	SAF400x HD User Manual		
[SAF400x_HD_Protocol]	SAF400x HD Protocol		
[SAF400x_DevControl_Protocol]	SAF400x Device Control Protocol		
[SAF400x_AN]	SAF400x Application Note		
[SAF400x_DS]	SAF400x Datasheet		
[SAF400x_SoftConfig_UM]	SAF400x SoftConfig User Manual	1.0	20171101

12. Legal information

12.1 Definitions

Draft — The document is a draft version only. The content is still under internal review and subject to formal approval, which may result in modifications or additions. NXP Semiconductors does not give any representations or warranties as to the accuracy or completeness of information included herein and shall have no liability for the consequences of use of such information.

12.2 Disclaimers

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

In no event shall NXP Semiconductors be liable for any indirect, incidental, punitive, special or consequential damages (including - without limitation - lost profits, lost savings, business interruption, costs related to the removal or replacement of any products or rework charges) whether or not such damages are based on tort (including negligence), warranty, breach of contract or any other legal theory.

Notwithstanding any damages that customer might incur for any reason whatsoever, NXP Semiconductors' aggregate and cumulative liability towards customer for the products described herein shall be limited in accordance with the *Terms and conditions of commercial sale* of NXP Semiconductors.

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

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

Applications — Applications that are described herein for any of these products are for illustrative purposes only. NXP Semiconductors makes no representation or warranty that such applications will be suitable for the specified use without further testing or modification.

Customers are responsible for the design and operation of their applications and products using NXP Semiconductors products, and NXP Semiconductors accepts no liability for any assistance with applications or customer product design. It is customer's sole responsibility to determine whether the NXP Semiconductors product is suitable and fit for the customer's applications and products planned, as well as for the planned application and use of customer's third party customer(s). Customers should provide appropriate design and operating safeguards to minimize the risks associated with their applications and products.

NXP Semiconductors does not accept any liability related to any default, damage, costs or problem which is based on any weakness or default in the customer's applications or products, or the application or use by customer's third party customer(s). Customer is responsible for doing all necessary testing for the customer's applications and products using NXP Semiconductors products in order to avoid a default of the applications and the products or of the application or use by customer's third party customer(s). NXP does not accept any liability in this respect.

Export control — This document as well as the item(s) described herein may be subject to export control regulations. Export might require a prior authorization from competent authorities.

Translations — A non-English (translated) version of a document is for reference only. The English version shall prevail in case of any discrepancy between the translated and English versions.

Evaluation products — This product is provided on an "as is" and "with all faults" basis for evaluation purposes only. NXP Semiconductors, its affiliates and their suppliers expressly disclaim all warranties, whether express, implied or statutory, including but not limited to the implied warranties of non-infringement, merchantability and fitness for a particular purpose. The entire risk as to the quality, or arising out of the use or performance, of this product remains with customer.

In no event shall NXP Semiconductors, its affiliates or their suppliers be liable to customer for any special, indirect, consequential, punitive or incidental damages (including without limitation damages for loss of business, business interruption, loss of use, loss of data or information, and the like) arising out the use of or inability to use the product, whether or not based on tort (including negligence), strict liability, breach of contract, breach of warranty or any other theory, even if advised of the possibility of such damages.

Notwithstanding any damages that customer might incur for any reason whatsoever (including without limitation, all damages referenced above and all direct or general damages), the entire liability of NXP Semiconductors, its affiliates and their suppliers and customer's exclusive remedy for all of the foregoing shall be limited to actual damages incurred by customer based on reasonable reliance up to the greater of the amount actually paid by customer for the product or five dollars (US$5.00). The foregoing limitations, exclusions and disclaimers shall apply to the maximum extent permitted by applicable law, even if any remedy fails of its essential purpose.

12.3 Licenses

Purchase of NXP <xxx> components
<License statement text>

12.4 Patents

Notice is herewith given that the subject device uses one or more of the following patents and that each of these patents may have corresponding patents in other jurisdictions.

<Patent ID> — owned by <Company name>

12.5 Trademarks

Notice: All referenced brands, product names, service names and trademarks are property of their respective owners.

<Name> — is a trademark of NXP Semiconductors N.V.

13. List of figures

14. List of tables

15. Contents

Date of release: 9 November 2017

Document identifier: doc238252

Date of release: 9 November 2017

Document identifier: doc238252