Transceiver Facility Specification

“Where software defines the radio”

SDRF-08-S-0008-V1.0.0

Approved 28 January 2009

TERMS, CONDITIONS & NOTICES

This document has been prepared by the Transceiver Interface Task Group to assist The SDR Forum (or its successors or assigns). It may be amended or withdrawn at a later time and it is not binding on any member of The SDR Forum or on the SDRF Transceiver Interface Task Group.

Contributors to this document who have submitted copyrighted materials (the Submission) to The SDR Forum for use in this document retain copyright ownership of their original work, while at the same time granting The SDR Forum a non-exclusive, irrevocable, worldwide, perpetual, royalty-free license under the Submitter's copyrights in the Submission to reproduce, distribute, publish, display, perform, and create derivative works of the Submission based on that original work for the purpose of developing this document under the Forum's own copyright.

Permission is granted to SDR Forum participants to copy any portion of this document for legitimate purposes of the SDR Forum. Copying for monetary gain or for other non-SDR Forum related purposes is prohibited.

THIS DOCUMENT IS BEING OFFERED WITHOUT ANY WARRANTY WHATSOEVER, AND IN PARTICULAR, ANY WARRANTY OF NON-INFRINGEMENT IS EXPRESSLY DISCLAIMED. ANY USE OF THIS SPECIFICATION SHALL BE MADE ENTIRELY AT THE IMPLEMENTER'S OWN RISK, AND NEITHER THE FORUM, NOR ANY OF ITS MEMBERS OR SUBMITTERS, SHALL HAVE ANY LIABILITY WHATSOEVER TO ANY IMPLEMENTER OR THIRD PARTY FOR ANY DAMAGES OF ANY NATURE WHATSOEVER, DIRECTLY OR INDIRECTLY, ARISING FROM THE USE OF THIS DOCUMENT.

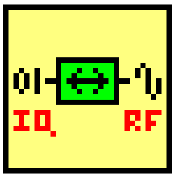
Recipients of this document are requested to submit, with their comments, notification of any relevant patent claims or other intellectual property rights of which they may be aware that might be infringed by any implementation of the specification set forth in this document, and to provide supporting documentation.

Executive Summary

This specification is the V.1.0.0 (“Increment 1”) of the Transceiver Facility Specification from the SDR Forum, referenced SDRF-08-S-0008.

For radio system integrators, waveform providers, SDR platform providers and radio head manufacturers who seek increased efficiency when integrating waveform applications with target platforms (including radio heads), and who seek increased portability for their waveform applications, the Transceiver Facility Specification V.1.0.0 is an open specification for transceiver subsystems that captures the information needed for interoperability between waveform applications and transceiver subsystems, expressed as generic and abstract requirements for properties and programming interfaces, including the associated real-time issues.

Unlike existing specifications such as OBSAI RP3, CPRI, Vita 49 or DigRF, which focus on interoperability between hardware subsystems, and which do not sufficiently address control and configuration mechanisms, this specification provides a software abstraction that decouples waveform software from the specifics of the transceiver subsystem implementation, while not preventing use of any of the aforementioned standards.



“Where software defines the radio”

Acknowledgment

This specification would not exist without the European Union funded project E3 (End-to-End Efficiency), which has enabled the essential of its content and supported the associated group work effort at SDR Forum level. The decisive contribution of the European Union and the E3 project as a whole to this work is hereby acknowledged.

Table of Contents

List of Figures

List of Tables

Document History

Version	Date	Editor	Description
V.0.1.0	15 April 2008	Eric Nicollet	Post-58th GM (Rome) release.
V.0.2.0	9 Sept 2008	Eric Nicollet	60th GM (Boston) release.
V.0.2.1	7 Oct 2008	Stéphane Pothin	Ctrb_TCF_01 merged on top of V.0.2.0
V.0.2.2	29 Oct 2008	Alejandro Sanchez	Contributions merged on top of V.0.2.1: Ctrb_TFS_TCF_02 Ctrb_TFS_TCF_03 (joint with Indra) Ctrb_TFS_TCF_04 Ctrb_TFS_TCF_05
V.0.3.0	20 Nov 2008	Eric Nicollet	Direct contribution from TCF on top of V.0.2.2. Contributors: Eric NICOLLET, Stéphane POTHIN, Alejandro SANCHEZ. **Release for Work Group ballot.**
V.0.5.0	23 Dec 2008	Eric Nicollet, Alejandro Sanchez	Update taking into account Work Group ballot comments. Editor's review taken into account. Clarifications and upgrades along the whole document, based on approval comments and a detailed Thales internal document review. No fund upgrade which would justify new Work Group ballot. **Release for Technical Committee ballot.**
V.0.6.0	26 Jan 2009	Eric Nicollet	Update taking into account Technical Committee ballot comments expressed by Uni of Karlsruhe. **Prepared to enter SDR Forum ballot.**
V.0.6.1	27 Jan 2009	Eric Nicollet	Few typos corrected. Couple of disambiguation sentences added in § 4.3.6. **Release for SDR Forum ballot.**
V1.0.0	5 Feb 2009	Allan Margulies	**Edit for public release.**

Table 1: Document History

Reference Documents

Identifier	Title	Author	Comments
[1]	Joint Digital IF Initial Submission	T. Demirbilek and E. Nicollet	OMG document, March 2005
[2]	The Transceiver Facility Platform Independent Model : Definition, Content and Usage	E. Nicollet	SDR Forum Technical Conference, November 2006, Orlando
[3]	Standardizing Transceiver APIs for Software Defined and Cognitive Radio	E. Nicollet and L. Pucker	Article from RF Design magazine, February 2008

Table 2: Reference Documents

Acronyms

Acronym	Signification
AGC	Automatic Gain Control
ALC	Automatic Level Control
API	Application Programming Interface
CPRI	Common Public Radio Interface
CPU	Central Processing Unit
dB	Decibel
dBFS	Decibel relative to the Full Scale
FS	Full Scale
HDL	Hardware Description Language
IF	Intermediate Frequency
IDL	Interface Description Language
JPEO	Joint Program Executive Office
JTRS	Joint Tactical Radio System
MAC	Medium Access Control
MDA	Model Driven Architecture
MDE	Model Driven Engineering
MHAL	Modem Hardware Abstraction Layer
OBSAI	Open Base Station Architecture Iniatitive
PA	Power Amplifier
PIM	Platform Independent Model
PSM	Platform Specific Model
RF	Radio Frequency
Rx	Reception / Receive
SCA	Software Communication Architecture
SDR	Software Defined Radio
Tx	Transmission / Transmit
UML	Unified Modeling Language
VHDL	Very high speed integrated circuit Hardware Description Language

Table 3: Acronyms

1 - Introduction

1.1 - Transceiver Subsystem

This chapter provides the reference definition for *Transceiver Subsystem*, as applicable within the *specification*.

The terminology *Transceiver* derives from the contraction of "<u>trans</u>mitter/re<u>ceiver</u>".

1.1.1 Definition

The *Transceiver Subsystem* is the part of a *radio chain* that transposes, for transmission, *baseband signal* into *radio signal*, and, for reception, *radio signal* into *baseband signal*.

The terminology *Transceiver* derives from the contraction of "<u>trans</u>mitter/re<u>ceiver</u>".

The transposition of *baseband signal* into *radio signal* is denoted as the *up-conversion*. The reciprocal is denoted as the *down-conversion*.

Tuning is characterizing the signal processing transformations realized by *up-* and *down-conversion*.

The *baseband signal* is a sampled complex signal, with (I,Q) values, denoted $\underline{s}_{BB}[n]$. The *radio signal* is is a continuous electric signal, denoted $s_{RF}(t)$.

The following figure illustrates the previous definitions:

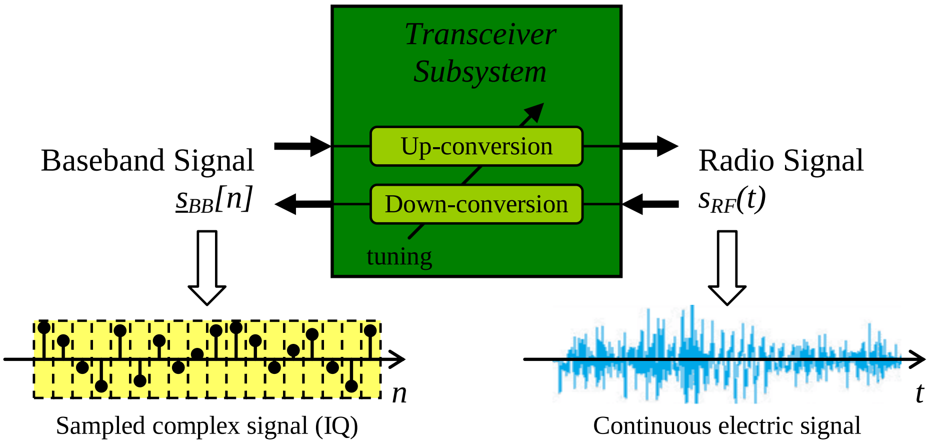


Figure 1: Definition of Transceiver Subsystem and key associated concepts

1.1.2 Positioning in Physical Layer

Inside a *radio chain*, the *Transceiver Subsystem* is comprised between the *Modem* and the *Antenna Subsystem*. It is exchanging the *baseband signal* with *Modem*, and *radio signal* with the *Antenna Subsystem*.

Modem, *Transceiver* and *Antenna* are generally considered as parts of the *Physical Layer*.

Modem

Modem is defined as the part of a *radio chain* that generates, in transmission, the *baseband signal* to be transmitted, and that exploits, in reception, the received *baseband signal*. Implementation of *Modem* relies on the modulation and demodulation techniques, which explains the selected name.

The *Modem* scope can not be strictly defined, since has an evident boundary towards the above layers of the *radio chain* which would be applicable in any case. Additionally, depending on the granularity to which a certain *radio chain* is decomposed, several functional modules can take part in the implementation of *Modem*.

Modem is thus used as the a role taken by the *functional modules* of the *radio chain* exchanging baseband signal with the *Transceiver Subsystem*.

Antenna Subsystem

Antenna Subsystem is defined as the part of a *radio chain* that transforms, for transmission, the *radio signal* into the transmitted *radio wave*, and, for reception, the impinging *radio wave* into the *radio signal*.

The signal transformation operated by the *Antenna Subsystem* is evidently conducted thanks to antennas, while the complete subsystem implementation may include additional hardware and control software (e.g. for mechanically directed antennas).

The following figure illustrates previous considerations:

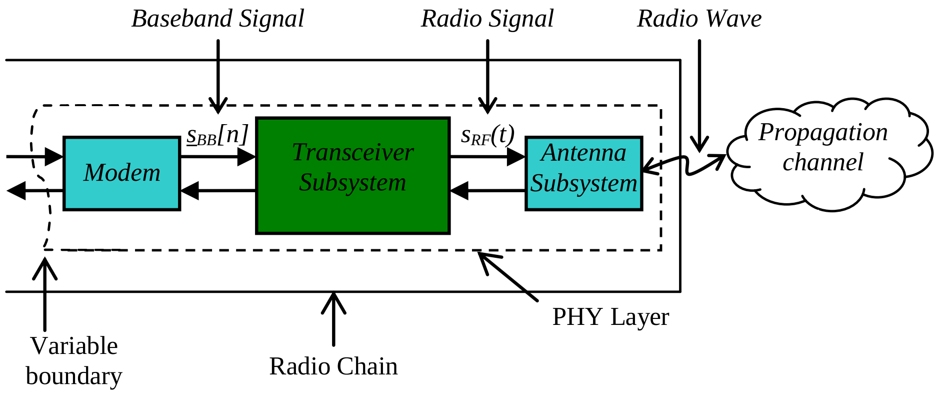


Figure 2: Transceiver Subsystem within the PHY Layer of a radio chain

1.1.3 Positioning within Radio Set Implementation

1.1.3.1 Implementation concepts

Reconfiguration Infrastructure

Reconfiguration Infrastructure is defined as the role taken by parts of the SDR Set which undertakes reconfigurations between the waveform capabilities supported by the Radio Set. It can be compliant with an open reference specification (e.g. SCA Core Framework solutions), or be a proprietary solution.

Waveform Application

The *Waveform Application* is composed of software modules which provide the essential software-defined dimension of the radio capability implementation. The software architectures are often targeted to achieve a high degree of portability for those software modules.

Within the *Waveform Application*, the *Waveform Functional Application* denotes the software modules which integrally implement (i.e. without intervention of a particular hardware) functionalities of the considered *Waveform Capability*. The emergence of high processing power programming capabilities is making implementation of such functionalities in highly portable software modules a tangible reality, even for the most constrained parts of the physical layer processing.

Within the *Waveform Application*, the *Configuration Agent* represents the role taken by a software module that accesses the other constituents of the radio capability implementations, in order to dynamically modify or read some configuration properties of those constituents. It typically relays orders provided by a user interface or commands retrieved from a remote source. In contrast to the Waveform Functional Application, this software has a strong dependency on the way commands impacting the *Transceiver Subsystem* are issued, and is not subject to be easily portable across different *Radio Sets*.

Platform Functional Support

The *Platform Functional Support* denotes the set of *Radio Subsystems*, such as *Transceiver Subsystem*, which are provided by the *SDR Platform* to complement the *Waveform Application* in order to dispose of a complete *Waveform Capability* implementation.

The Transceiver Subsystem and the Antenna Subsystem are parts of the Platform Functional Support.

1.1.3.2 Illustration

The following figure illustrates the *Transceiver Subsystem* positioning within a typical radio implementation inside an SDR set:

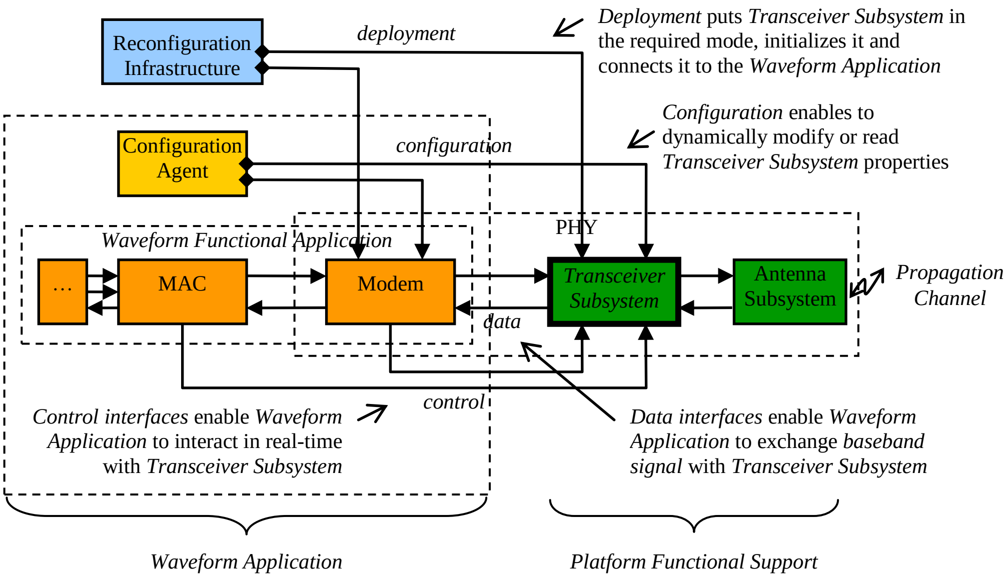


Figure 3: Transceiver Subsystem and related capabilities

The previous figure is an example that illustrates a typical set of modules and relationships, corresponding to an example of wavefom capability implementation. It is recalled that the terminology “waveform” is used by reference to radio communications, but covers a potentially wider scope of radio applications than radio communications only (e.g. spectrum sensing, direction finding, identification friend-foe, …).

The control interaction between *MAC* and *Transceiver Subsystem* is underlining that in some cases the control interactions from the *Waveform Functional Application* to the *Transceiver Subsystem* are not strictly limited to Physical layer internal interactions.

For simplification purpose, some deployment and configuration interactions between *Reconfiguration Infrastructure* and *Waveform Application* components are not represented.

The roles of *Reconfiguration Infrastructure* and *Configuration Agent* are presented in conformance with previous definitions. The *Transceiver Subsystem* and *Antenna Subsystem* are apperarting as parts of the *Platform Function Support*.

The presented *Waveform Application* is composed of one *Modem* software component, with one *MAC* software component, the “…” module representing other components of higher layers which are not related to the *Transceiver Subsystem*.

1.1.4 Internal architecture

There are as many different *Transceiver Subsystem* implementations as there are radio equipments. The variety in *Transceiver Subsystem* implementations, architectures, designs and technology is huge, and out of the scope of the *specification*, which specifically abstracts the implementation choices.

There are nevertheless internal architecture invariants which enable better depictions of *Transceiver Subsystem*s.

A *Transceiver Subsystem* implementation is composed of a digital segment and an analogue segment separated by a conversion stage. The breakdown between the digital and analogue segment is one essential architectural choice of a transceiver design.

The conversion stage is in charge of transforming the digital signal into analogue signals and vice-versa. It can be positioned at any level between the following extremes: (i) immediately at base-band, for each real component of the complex baseband signal, (ii) directly at the antenna foot.

The digital segment, denoted as *Digital Transceiver*, is in charge to interact with the *Waveform Application*, in compliance with the facility programming interfaces. It can undertake a significant part of the signal transformation, when conversion is not happening at baseband level. This part represents the whole lot of the digital segment in the case of antenna foot conversion, often depicted as “genuine” software radio case.

The analogue segment, denoted as *Analogue Transceiver*, is in charge of exchanging the RF signal with the antenna Subsystem. It undertakes all necessary transformations between the conversion stage and the antenna interface. In most situations the analogue stage will comprise a power amplification stage.

1.1.5 Standard-based implementation solutions

Using standard implementation solutions to provide a complete Transceiver Subsystem can be a relevant option in many industrial cases.

Standards having been identified as typical cases are comprised of OBSAI (or CPRI) in base-stations manufacturing, DigRF in cellular handsets manufacturing, or JTRS JPEO MHAL RF Chain Coordinator in military radios manufacturing.

For illustrative purposes, Annexe 2 graphically presents a certain number of such situations.

1.2 - Rationale for a standard specification

The ongoing development of software-defined radio (SDR) technologies in advanced wireless systems, allowing some or all of the radio implementation to be realized in software and/or HDL code, is creating a need for a further level of standardization between the *Waveform Application* and the *Transceiver Subsystem*.

This standard specification shall enable interoperability between *Waveform Applications* and *Transceiver Subsystems*.

Expected benefits of such interoperability are (i) increased integration efficiency between a compliant *Transceiver Subsystem* segment and the *Waveform Application* parts, (ii) increased portability of a *Waveform Applications* through different radio platforms with compliant *Transceiver Subsystems*, (iii) increased openness of *Transceiver Subsystems*, capable of meeting the needs of a large panel of *Waveform Applications* with reduced costs.

As a consequence, the standard specification shall specify the *Transceiver Subsystem* thanks to a generic and abstract approach. Generic signifies that the specification shall be capable to fulfill the needs of the widest possible range of waveform capability, ideally any of them. Abstract signifies that the specification shall propose a characterization of the *Transceiver Subsystem* which does not make *any* implementation specific assumptions.

The following figure is summarizing those expectations:

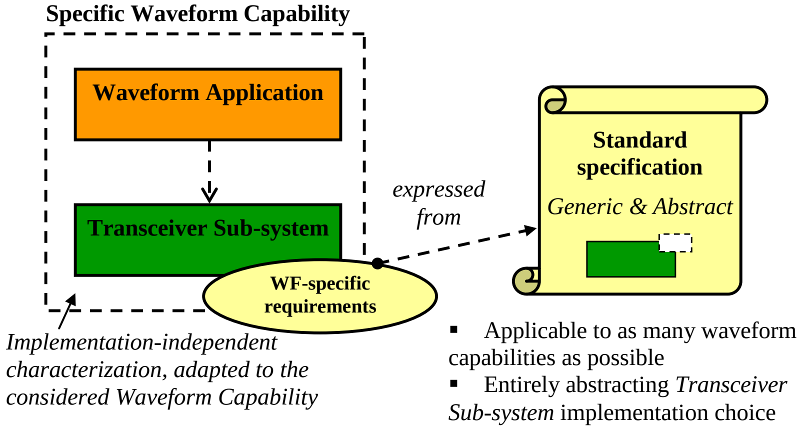


Figure 4: Key expectations towards Transceiver Subsystem Facility

Meeting the difficult compromise of the often contradictory ambitions of genericity and abstraction requires capturing the fundamental domain concepts which govern radio engineering expertise. As such the Facility elaboration presents similitudes with elaboration of a domain specific language.

1.3 - The Transceiver Facility

1.3.1 Overview

The *Transceiver Facility* is the solution proposed by the SDR Forum for definition of a generic and abstract *Transceiver Subsystem* specification conformed to the expectations exposed in the previous rationale.

It is a general-purpose specification applicable to particular development cases. It aims at becoming a largely referenced and implemented standard within the SDR industry.

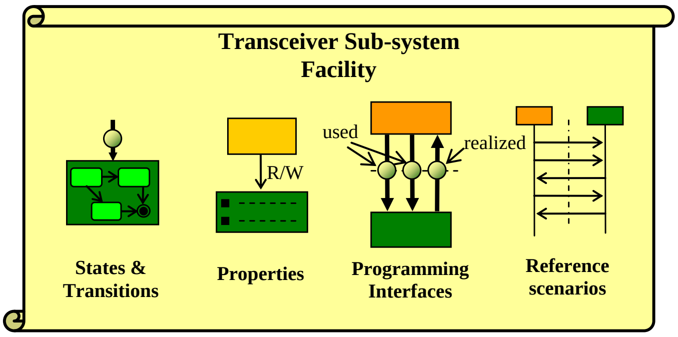
Its definition is an openly available SDR Forum Specification and follows an open elaboration process conducted by the SDR Forum.

Specification topics

The requirements expressed in the Facility are covering the following topics:

- Structural Concepts
- States & Transitions
- Properties
- Programming Interfaces (*used* and *realized* interfaces)
- Real-time requirements
- Reference scenarios

The following figure summarizes what the Transceiver Subsystem Facility is:



- Generic & Abstract specification
- Publicly available
- Openly defined by SDR Forum

Figure 5: Overview of the Transceiver Subsystem Facility

1.3.2 Specification levels

The *analysis requirements* of the Facility (i.e the requirements issued from the analysis process) impact the complete development chain, from the early design phase to the final integration stages. They are valid whatever the implementation choices will be.

To support usage of model-driven engineering and developments, UML concepts are formally required in direct derivation from the *analysis requirements*.

Reference source code applicable for programming of *Waveform Application* and *Transceiver Subsystems* interactions, in a number of popular programming languages such as C, C++, VHDL, etc., are as well specified.

Finally, requirements are expressed to characterize how *Transceiver Subsystems* compliant with the Facility shall be implemented following the specific constraints of reference SDR standards. The SCA (Software Communication Architecture) is addressed from this perspective.

1.3.3 Features and Common Concepts

The Facility is organized through Features and Common Concepts.

Features

A *Feature* is an **elementary capability** of a Transceiver Subsystem for which the Facility is providing a set reference *specification items* in a dedicated chapter.

Two categories of features are identified:

- **Core features:** Specifying basic and mandatory functionalities of a transceiver, typically Transmit/Receive capability
- **Optional features:** Specifying functionalities which are dependent upon the transceiver type, for example Half/Full Duplex features

This separation of capabilities in Features provides the *Transceiver* Facility with modularity, extensibility and scalability.

Core and Optional Features include a set of specification items (APIs, attributes, behaviors, etc.) which enable either the waveform or the platform developers to manage the various transceiver interfaces and characterize their performance.

Example

> *The Feature "Transmit Channel" provides a reference function used by the Waveform Application to notify the Transceiver that a packet of new baseband samples is ready, called pushBBSamplesTx(). Properties attached to the baseband flow exchanged between Waveform Application and Transceiver are the sample rate and the number of coding bits for the stream signal.*

Common Concepts

A *Common Concept* is defined as a set of *specification items* collectively shared between different *Features*, and captured independently from the Features themselves.

The introduction of Common Concepts avoids duplication of information throughout the *specification*.

Example

> *The concept of baseband signal power level is defined in the Common Concept that is used by both the "Receive Channel" and "Transmit Channel" features.*

2 - Specification Assumptions

This chapter defines the specification assumptions as applied by the *Facility* to express formal requirements.

2.1 - Waveform engineering

The *abstract* and *generic* characterization of the *Transceiver Subsystem* which is realized by the *Facility* is made through the *External Interfaces* and *Internal Structure* of the subsystem, as depicted in following chapters.

These assumptions are intended for engineering of each specific *Waveform Capability*. They are not addressing matters associated to *Transceiver Subsystem* implementations, and the issues related to simultaneous support of different waveform capabilities by a given multi-radio *Transceiver Subsystem*.

Only mono-antenna transceiver capabilities are considered in this chapter, meaning that MIMO or Smart Antenna techniques are not supported by the *current specification*.

2.1.1 External interfaces

The following figure depicts the external interfaces of a *Transceiver Subsystem*:

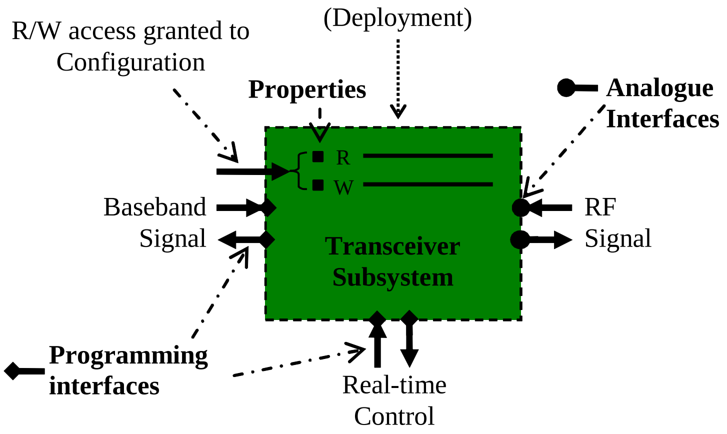


Figure 6: External interfaces of a Transceiver Subsystem

Programming Interfaces

The *Baseband Signal* interfaces convey the baseband signal from the *Waveform Application* (typically its Modem part) to the *Transceiver Subsystem*. It represents the transceiver input data for transmission and the output data for reception. Typically baseband I&Q samples from a particular waveform travel through this interface.

The *Real-time Control* interfaces provide data flow control and real-time configuration. Typically the facility provides an API with generic waveform-independent methods configuring transceiver performance and air-interface RF access.

The *Baseband Signal* interfaces and the *Real-time Control* interfaces compose the *Programming Interfaces* attached to the *Transceiver Subsystem*.

Analogue Interfaces

The *RF Signal* interfaces convey the radio frequency signal from the *Transceiver Subsystem* to the *Antenna Subsystem*. They are the only externally visible *Analogue Interfaces* presented by the *Transceiver Subsystem*.

Properties

The *Transceiver Subsystem* properties are not as such interfaces, but they do correspond to data representations of internal notions of the *Transceiver Subsystem* which enable the *Waveform Configuration* to *read* current values taken by the considered notion or to *write* desired values to be implemented by the considered notion.

2.1.2 Internal structure

The following figure depicts the internal structure of a *Transceiver Subsystem*:

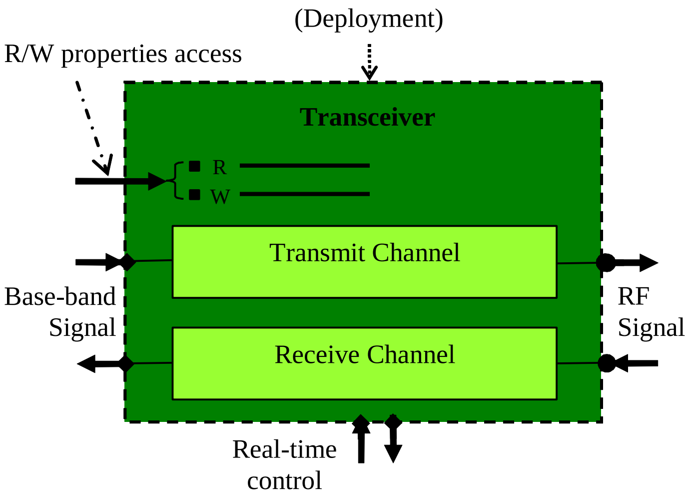


Figure 7: Internal structure of a Transceiver

Transmit and *Receive Channels* are the *core features* associated with the *Transceiver Subsystem*. For a given *Waveform Capability*, the way they are arranged characterizes three primary possibilities:

- "True" Transceiver: *Receive* and *Transmit Channels* core features are both needed
- Receiver ("Receive-only Transceiver"): only *Receive Channel* core feature is needed
- Transmitter ("Transmit-only Transceiver"): only *Transmit Channel* core feature is needed

As far as "True" Transceivers are concerned, the following possibilities can then be distinguished:

- Full-duplex Transceiver: *Receive* and *Transmit Channels* can be simultaneously activated
- Half-duplex Transceiver: *Receive* and *Transmit Channels* shall be exclusively activated

The Facility is based on a baseline specification of the identified core features, complemented by Transceiver-level notions enabling the ability to characterize how they are arranged. Corresponding specifications are provided in §3.

2.1.3 Deployment

Definition of "Deployment"

Deployment corresponds to the implementation-specific mechanisms used to dispose of a *Transceiver Subsystem* operating in compliance with the needs of a particular radio capability.

Beyond the *Transceiver Subsystem*, *deployment* concerns the complete implementation of the considered radio capability. It thus covers the deployment of the different software components of the *Waveform Application*, and the way they are connected together and with the *Transceiver Subsystem* itself.

Deployment is primarily introduced to depict the reconfiguration mechanisms of SDR equipment, which triggers *deployments* in order to evolve from one radio configuration to another.

Deployment typically covers the following steps:

- **Creation:** the expected *Waveform Application* components are created and the necessary support subsystems (incl. *Transceiver Subsystem*) are attached to the waveform implementation
- **Connection:** the connections between *Waveform Application* components and support subsystems (incl. *Transceiver Subsystem*) are established
- **Initial configuration:** initial values of properties of *Waveform Application* components and support subsystems (incl. *Transceiver Subsystem*) are set

Deployment can be based on a proprietary solution or can use a standard solution such as SCA Core Frameworks.

The previous definition encompasses non-SDR equipments, for which radio capabilities are implemented using a static deployment mechanism, which deploys once-for-all the associated software, but does not allow reconfiguration.

Deployment Abstraction

The Facility assumes a total abstraction of deployment mechanisms, making no assumption on the *deployment* solution, which is considered as strictly dependent on *Transceiver Subsystem* implementation choices.

This means in general that SDR or non-SDR implementations are addressed by the Facility.

Furthermore, in the mainstream case of SDR implementations, this abstraction enables to undistinctly address standard-compliant (e.g. SCA) or proprietary solutions.

2.2 - Requirements

This chapter explains the different nature of requirements expressed by the Facility.

2.2.1 Abstraction and Genericity

The following chapters detail generally applicable constraints that have to be met by all the requirements expressed in the *Facility*.

Abstraction

Abstraction (of the *Transceiver Subsystem* implementation) signifies that requirements of the *Facility* shall be totally independent of any implementation choice of the *Transceiver Subsystem*.

This applies as well for implementation requirements, meaning that beyond the implementation specific assumptions made (e.g. choice of a particular programming language), no further assumptions dependent on the internal implementation choices may be made.

Genericity

Genericity (in front of as many radio capabilities as possible) signifies that requirements of the *Facility* shall be applicable to the widest possible extent of waveform capabilities.

No additional requirement should be added for the sake of supporting a particular waveform needs, unless the existing requirements would prove not usable by the considered waveform capability.

2.2.2 Functional requirements

2.2.2.1 Requirements Analysis

***Analysis requirements* are the very core of the Facility specification.**

They are expressed without any explicit assumption concerning the implementation choices of the *Waveform Application* and the *Transceiver-Subsystem*.

Analysis requirements are applicable to any step of *engineering activities* related to the *Transceiver Subsystem* and associated *Waveform Applications* (specification, preliminary design, detailed design, coding, testing, integration, …).

Analysis requirements are expressed in common technical English language, with eventual usage of non-normative technical illustrations to support explanation of the concepts. The principle objective is to achieve as general as possible requirements.

Analysis requirements appear within the core text of the Facility.

2.2.2.2 Modelling requirements

Modelling requirements provide normative UML modelling artifacts to be used in Facility-compliant UML models. Requirements specifically identify the associated modelling artifacts.

Modelling requirements are expressed in UML 2.0. They appear:

- Within the core text of the Facility, distributed in complementary sections next to the analysis requirements to which they correspond
- As a reference model (the *Transceiver Facility Reference UML Model*) attached to the *specification*, provided as XMI source file

Some specific UML conventions applied to build the UML artifacts associated with modelling requirements are explained in the rest of the document using specific sections denoted “Modelling Support”. Applicable stereotypes are a typical example of what is captured in such chapters.

Structural stereotypes

Analysis notions and concepts are developed in Features and Common Concepts specifications that are modelled in UML as classes. The following stereotypes are applicable for modelling classes associated with Transceiver structural concepts:

- <<core feature>> applicable to classes modelling core features
- <<feature concept>> applicable to sub-classes aggregated by features
- <<common concept>> applicable to common concepts

2.2.3 Programming language requirements

Programming language requirements provide the normative programming language source code to be used in Facility-compliant software developments (*Waveform Application* components and *Transceiver Subsystem*).

Current specification addresses the following programming languages:

- C++
- VHDL
- IDL

Programming language requirements appear:

- Throughout the core document, next to the corresponding Analysis requirements, organized into language-specific sections
- As language-specific annexes to the document core, regrouping all the source code associated to the considered programming languages
- As reference source code, provided as source files for the different programming languages

2.2.3.1 C++ requirements

C++ requirements address the C++ language.

Conformance to C++ reference standard, ISO/IEC 14882:2003, is assumed.

2.2.3.2 VHDL requirements

The version of the VHDL language used in this specification is VHDL 93.

The non-exhaustive list of naming conventions is presented in the following table:

VHDL constructs	Naming conventions
generic	G_ prefix with name in uppercase and underscore between words
constant	C_ prefix with name in uppercase and underscore between words
active low signal	_n suffix

Table 4: VHDL naming conventions

The mapping rules between UML, IDL and VHDL constructs are described below:

UML constructs	IDL constructs	VHDL constructs
	boolean	Std_logic
	short	Std_logic_vector (15 downto 0)
	unsigned short	Std_logic_vector (15 downto 0)
	long	Std_logic_vector (31 downto 0)
	unsigned long	Std_logic_vector (31 downto 0)
integer of nbBits		Std_logic_vector (nbBits-1 downto 0)
Package	module M { ... };	package P is ... end package P;
Class	interface I { ... };	Entity E is ... end entity E;

Table 5: UML, IDL and VHDL constructs mapping

Each Transceiver Feature is described by a VHDL entity, which has to include clock and asynchronous active low reset signals. Other optional signals can be added to the entity by the designer if needed, such as clocks for the following Features or access to FPGA pins (e.g. for LEDs control).

There are three possible implementations of a feature property:

1. It can be implemented as a CONSTANT if its value can not be modified once synthesized (compilation time). The CONSTANT can be defined:
    a. inside a *Transceiver* PACKAGE if it is desirable to make it accessible for every Feature, or
    b. inside a feature specific PACKAGE if it is only needed by this feature,
    c. a constant may be used to configure VHDL generics on the feature entity interface during component instantiation.
2. It can be implemented as a register if its value may be modified during run time. The register can be defined inside of a clocked process inside the architecture of the Feature.
3. It can be data/control signals
    a. which are declared in the corresponding Feature entity, and
    b. whose protocol is defined inside the Feature architecture.

Generally speaking, the transfer of information from and to the Transceiver features is based on two phases

1. Up stream or request flow consisting of
   a. Useful data,
   b. Control flow signals in the data direction with a mandatory 'write' signal, which means that a valid data is ready to be sent,
   c. Other optional signals to include user-defined extensions
2. Down stream or response flow consisting of
   a. Control flow signals in the opposite data direction with a mandatory 'ready' signal, which means that the next valid data to be sent will be accepted
   b. Other optional signals to include user-defined extensions

2.2.3.3 IDL requirements

To be completed.

2.2.4 Implementation architecture requirements

Implementation architecture requirements provide normative complementary requirements which are applicable in case the *Waveform Application* software and the *Transceiver Subsystem* are developed in compliance with particular architecture assumptions.

The only case considered in the *current specification* is SCA compliancy. Others such as Native Simulation are considered for future evolutions.

2.2.4.1 SCA

To be completed.

2.2.5 Requirements Identification and Numbering

This section summarizes the conventions used within the document for requirements specification.

Requirements identification

The requirements are identified by a unique identifier, composed of the following field:

- Requirement family: General → "GEN_", Feature → "FEAT_", Common Concept → "CC_"
- Mnemonic: "<mnemonic>_"
- Requirement number

A mnemonic is a short capital letters string of characters attached to a certain chapter/section of the specification. The identifiers of the requirements attached to the chapter/section will be using the chapter/section mnemonic.

The requirement numbering is a three decimal digits number.

Requirement tags

The requirements are tagged by a few words capturing the nature of each requirement. These tags are intended to facilitate specification ease-of-read and are not normative concepts.

Delimitation of requirements

The beginning of a requirements expression is delimited by an introductive line, containing the identifier of the requirements and its tag, in conformance with the following arrangement convention:

<IDENTIFIER> & <SPACE> & "[" & <5 x SPACE> <TAG>

The format of the entry delimitation is in bold red text.

Example: **FEAT_TXCHAN_010 [Example Req**.

The end of a requirement expression is delimited by a conclusive line, containing the identifier of the requirement, formatted as:

“] END_”& <IDENTIFIER>

The format of the end delimitation is in bold green text.

Example: **] END_FEAT_TXCHAN_010**.

Verification of requirements

Description of the verification principles for each identified requirement is not in the scope of the *current specification*.

2.3 - Detailed specification conventions

Following chapters explain, for different sorts of concepts introduced within the *specification*, general specification conventions applicable to the concerned concepts.

2.3.1 Classification of Transceiver Characteristics

2.3.1.1 Introduction

The *Facility* introduces many characteristics associated with the *Transceiver Subsystem*. “Characteristic” refers to an information set, simple or structured, quantified or qualitative, which corresponds to a particular notion of the *Transceiver Subsystem*.

Characteristics are used during radio capability engineering to define how the *Transceiver Subsystem* shall operate in conjunction with the *Waveform Application* in order to implement the *radio capability* of interest. The notions described thanks to characteristics can indistinctively be constant or transient.

Most characteristics of the *Transceiver Subsystem* are formally defined by the Facility, through specific requirements. The purpose of this section is to define the classification used by the Facility in order to differentiate those characteristics.

The categories of *Notions* and *Constraints* are the first level of differentiation used for discrimination of characteristics.

This Facility defines as *Notions* characteristics of the Transceiver Subsystem which are *observable*. This criteria (the possibility to be *observed* or not) is the fundamental frontier which distinguishes the concept of *Notion* from the concept of *Constraint*.

“Observable” indicates that by using inspection, analysis and/or measurement means, values of the considered characteristics of a specific implementation of a *Transceiver Subsystem* can be inferred. This surely includes externally observable characteristics specified in implementation independent manner, but as well, implementation specific aspects of the considered *Transceiver Subsystem*.

The *Constraints* are defined as non-observable notions, which are typically the design constraints captured in the requirement specifications, which are satisfied by a given *Transceiver Subsystem* implementation, but which are not explicitly defined by measurement. For instance all boundary values of *Notions* are typical of constraints, which the observed notion is complying with, but being by definition impossible to measure by themselves.

2.3.1.2 Notions

Notions are *Transceiver Subsystem* observable characteristics, sorted by the *Facility* among three sub-categories:

- *Internal* notions
- *Implicit* notions
- *Explicit* notions

Internal Notions

Internal Notions are *Notions* answering to the following criterion:

- **Implementation dependency:** one *Internal Notion* of a *Transceiver Subsystem* is **dependent** on some implementation assumptions of the *Transceiver Subsystem*, no matter how minor such assumptions may be

As a consequence, since the *Facility* is providing abstraction from the implementation choices, no information relative to a particular value of an internal notion may be considered at any stage of the radio capability engineering, from the design phases to the run-time software exchanges between the *Waveform Application* and the *Transceiver Subsystem*.

Internal Notions are nevertheless essential to consider for *Waveform Application* design, and boundary *Constraints* are handled in *Waveform Engineering* to make appropriate design. *Internal Notions* are typically real-time or signal processing characteristics.

<u>Examples</u>

> *Up-conversion Latency is an "Internal Notion" which defines the time baseband samples need to travel across the Up-conversion chain prior to becoming the radiated RF signal. This value is bounded by the Engineering constraint "Max Up-conversion Latency"and its actual value depends on the particular Transceiver implementation.*
>
> *ConsumptionStartTime is an "Internal Notion" derived from TransmitStartTime as the instant at which baseband samples should be consumed by the "Up-conversion chain" in order for the RF Transmission signal to be available at the Transceiver output at instant TransmitStartTime.*

Implicit Notions

Implicit Notions answer to the following criteria:

- **Implementation independency:** one *Implicit Notion* of a *Transceiver Subsystem* is **independent** from any implementation assumptions of the *Transceiver Subsystem*, no matter how minor such assumptions may be
- **Not exchanged:** no information relative to an *Implicit Notion* is exchanged at run-time between the *Waveform Application* and the *Transceiver Subsystem*, through programming interfaces or usage of *Configuration*

Implicit Notions are thus completely transparent from the *Waveform Application* point of view, but are essential to be given a well defined value during Waveform Engineering, which will both influence the implementation of the *Waveform Application* and the *Transceiver Subsystem*.

Such notions generally remain constant once deployment is completed. The *Deployment* is in charge to bring the *Transceiver Subsystem* to support the specified values, as needed by the considered *radio capability*. The mechanism of *presets* enables to switch during run-time among a predefined constant set of implicit notions.

<u>Examples</u>

> *The SamplingFrequency is an example of Implicit Notion. Its frequency in hertz is set once during Transceiver deployment. No waveform is supposed to change it. It remains constant except if a reconfiguration occurs.*

Explicit Notions

Explicit Notions answer to the following criteria:

- **Implementation independency (as for Implicit Notions):** one *Explicit Notion* of a *Transceiver Subsystem* is **independent** from any implementation assumptions of the *Transceiver Subsystem*, no matter how minor such assumption may be
- **Exchanged:** information relative to an *Explicit Notion* can be exchanged at run-time between the *Waveform Application* and the *Transceiver Subsystem*, through programming interfaces or usage of *Configuration*

Explicit Notions encompass both static concepts, changed only by means of the configuration functionalities of the waveform application, and very transient concepts characterizing accurate real-time behaviour.

Explicit Notions belong to at least one of the two following sub-categories:

- Configurable
- Programmable

The *Facility* explicitly identifies which explicit notions are only *Configurable*, only *Programmable*, or potentially accessed with both possibilities.

Explicit Notions potentially accessed by both possibilities are in principle accessed using only one of the two possibilities once the engineering choices of the associated radio capability are done.

Configurable notions are accessed by the *Configuration* part of the *Waveform Application* (if existing for the considered radio capability), which can read or write associated values through usage of the associated *Property*. The *Facility* only identifies the nature of the *Property*, leaving to implementation choices the way it is implemented by the *Transceiver Subsystem* and handled by *Configuration*.

Configurable notions are modified by external agents of the radio capability, typically the end user or a remote configuration manager. The associated configuration mechanism is not strictly bounded from a real-time perspective, what matters is to be reactive in front of the configuration agent needs.

Programmable notions are accessed by the *Waveform Functional Application* through usage of particular operations of the *Programming Interfaces* specified by the *Facility*. Specifically related arguments of the *Programming Interfaces* are introduced by the *Facility*.

Programmable notions are typically very transient values automatically accessed in run-time by the *Waveform Application* software, thus generally answering to specific real-time constaints.

Example

The CarrierFrequency is an Explicit Notion as far as it has a dedicated argument in some waveform programming interface operations i.e CreateTransmitChannel. It may be a static property whose value is set only-once during configuration as in fixed frequencies radios (no need to use the dedicated argument in this case) or a very transient property varying in run-time, typically frequency hopping radios (argument will be use to convey information during run-time in this case).

2.3.1.3 Constraints

Per the definition provided beforehand, constraints are *non observable* characteristics of the *Transceiver Subsystem*.

Constraints are assigned values during *Waveform Engineering*, and implementation phases ensure compliancy with the assigned values. They lead to primary requirements within this specification.

Characteristics inferred from the previous definition are:

- Implementation independency
- Not exchanged

Example

> *Up-Conversion Latency which describes the elapsed time from the instant baseband samples arrive to the Up-Conversion chain and the instant an RF signal is available at transceiver output is an Internal Notion. This time is dependent on implementation (like any Internal Notion).“MaxUp-Conversion Latency” is the Constraint which limits the value the Notion is allowed to take.*

2.3.1.4 Summary table

The following table summarizes the essential criteria answered or not by the categories and sub-categories indentified in the previous chapters:

Category	Sub-Category	Observable	Implementation Dependency	Waveform/Transceiver exchange
Notion	Internal	Yes	Yes	No
	Implicit	Yes	No	No
	Explicit	Yes	No	Yes
Constraint	N/A	No	No	No

Table 6: Differences between Characteristics categories

2.3.1.5 Modelling support

The following stereotypes are defined for modelling support:

- Stereotype <<Constraint>> for *constraints*
- Stereotype <<Explicit>> for *explicit notions*
- Stereotype <<Implicit>> for *implicit notions*

2.3.2 Operations and Programming Interfaces

All information exchanges between the implementations of a *Transceiver Subsystem* and a *Waveform Application* are realized thanks to *operations*. *Operations* are regrouped into *Programming Interfaces*.

An *operation* of the *Transceiver Subsystem* facility characterizes one possible information exchange mechanism between the *Waveform Application* and the *Transceiver Subsystem*.

Operations and *Programming Interfaces* are primarily defined at analysis level in a strictly implementation-independent approach. The requirements specific to implementation programming languages are then provided as supplements, as detailed in chapter “Programming Languages Requirements”.

This section strictly focuses on the principles applied throughout the *Facility* for analysis level specification of *Operations* and *Programming Interfaces*.

2.3.2.1 Static assumptions

Defining operations

Defining operations consists of providing a complete set of information, as detailed hereinafter.

Each *operation* is given a specific operation *name*.

An operation enables information exchanges in a specific *direction*: from *Waveform Application* to *Transceiver Subsystem*, or vice-versa.

Exchanges based on a given *operation* happen through circulation of elementary *messages*, each *message* carrying a structured set of information characterized by the list of *arguments* of the operation.

The typing of operation *arguments* is compliant with *arguments typing conventions* detailed in a later chapter of the *specification*.

The originator of the information exchanged through an *operation* has a *use* relationship with the *operation*, while the addressee has a *realize* relationship with the *operation*.

Bi-directional Programming Interfaces

The *Facility* considers bi-directional interactions, which means that the *Transceiver Subsystem* is by principle subject to invoke operations realised by the *Waveform Application*, additionally to the intuitive case for which the *Waveform Application* invokes operations realized by the *Transceiver Subsystem*.

This approach, as suprising as it may appear at first sight, has been identified as the only solution that can characterize the waveform application remaining in a truly implementation abstract mode. For instance, a callback mechanism for reception is specifically software-based, and is at run-time equivalent to a call to a registered method. The Facility is simply specifying the name and prototype of the call back, the registration being part of life cyle designs which are not at analysis level.

Furthermore, this is compliant with the component-based software design paradigm which predominates for standard-based SDR Waveform Application designs.

Example of Realized interfaces

The flagship example of Used interface if the operation pushBBSamplesRx(), which is enabling a Receive Channel to notify the Waveform Application of the availability of a new packet of baseband samples.

This means that a waveform analysis based in the specification shall consider the Transceiver as the caller of this operation, which corresponds to the natural flow of information. This is nevertheless sufficiently disruptive in front of most frequent design approaches where the Transceiver Subsystem is considered as a "slave" system, with the Waveform Application performing all the calls towards the Transceiver, with in particular usage for reception of getSamples() or callbacks registration approaches.

Simplifying assumptions

A certain number of simplifying assumptions are used for specification of *Operations*. Those simplifications focus on what is strictly necessary as far as engineering related to *Transceiver Subsystem* is concerned, in order to facilitate as efficient as possible implementations of the *Facility* concepts.

Those simplifying assumptions are:

- No return value
- No exceptions
- Types subset

No return value: the operations are not be expected by the originator to bring return values after completion.

No exceptions: the operations are not expected to generate any sort of software exceptions.

Types subset: the types applicable to operation arguments are limited to a certain sub-set in front of the types set classically supported in software engineering. Refer to the chapter on typing conventions for details.

Operations arguments naming convention

The arguments of the interface operations are all preceded by the keyword *requested* in order to make clear distinction against the characteristics (explicit notions) they refer to.

Specification approach summary

At analysis level, operations are statically specified providing name, direction, and list of typed arguments. The dynamics are specified through identification of what orginator and addressee shall do before, during and after operation invocation.

2.3.2.2 Dynamic assumptions

Operations invocation

Invocation of an *operation* occurs when originator of the *operation* triggers the information exchange. In doing so the originator provides the values to be carried by the generated message for each of the operations arguments. Some arguments can remain undefined.

Invocation of an operation occurs at the certain *invocation time*.

Message handling carries the *message* to the *addressee*, which is triggered to perform *operation execution*, which corresponds to the exploitation of the message.

Until the *message handling* is over, the originator is on hold. The instant when message handling is over is defined as the *completion time*.

Interaction models

Different strategies can be associated with *message handling*, and are denoted interaction models.

Interaction models are primarily differentiated by considering the independence of the *originator* and *addressee* of the message.

The *originator* and *addressee* can be executed on the *same thread of control*, in which case *invocation handling* triggers the addressee for operation execution, while the originator is maintained on hold, waits for the operation execution to be over, and only then releases the originator.

The originator and addressee can be executed on *different threads of control* (i.e. independently), in which case invocation handling triggers the addressee for operation execution, but does not wait for operation execution to start before releasing the originator.

2.3.2.3 Real-time Constraints

Maximum Invocation Duration

Max Invocation Duration is a constraint which can be attached to each *interface* specified in the *Facility*. It characterizes the amount of time taken within the caller execution time when an operation is invoked.

Each occurrence of an *invocation* by the *Caller* of the operation happens at an instant denoted *Invocation Time*. The *Caller* is then on hold until the operation invocation mechanism *returns*, giving back the hand to let *Caller* proceed. The instant when the call *returns* is denoted *Return Time*. The time elapsed between *Invocation Time* and *Return Time* is represented by the *Internal Notion Invocation Duration*.

Each occurrence of an operation call can have a different *Invocation Duration*. Whatever the occurrence of an operation invocation, the constraint *Max Invocation Duration* specifies the maximum value possibly taken by *Invocation Duration.*

A value for this constraint is defined when accurate real-time engineering is necessary for the considered *waveform capability*.

A constraint *Max Invocation Duration*, of domain type *Latency*, is specified by the Facility for each defined operation, as depicted in following figure:

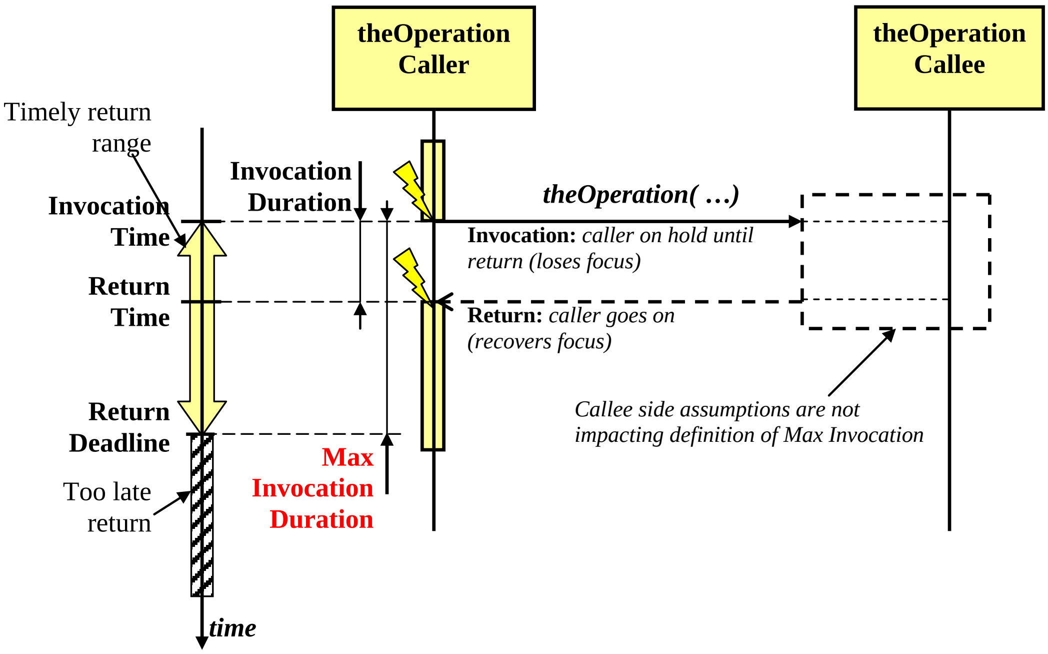


Figure 8: Notion of Max Invocation Duration

2.3.2.4 Modelling support

UML *interfaces* are used to capture the programming interfaces.

UML *operations* are used to capture the analysis *operations* contained in programming interfaces.

2.3.3 Types setting conventions

This chapter gives the dispositions applicable, throughout the *specification*, for definition of (i) *Notions* and *Constraints* types and (ii) *Arguments* of programming interfaces operations.

2.3.3.1 Analysis types

Base types

Allowed basic types used within the spec are::

Base Type	Correspondance
Boolean	True / False
Short	Signed integer, 16 bits min
Long	Signed integer, 32 bits min
UShort	Unsigned integer, 16 bits min
Ulong	Unsigned integer, 32 bits min

Table 7: Base types

The specified set of Base types has been taken as a sub-set of CORBA types.

Notice that those types represent an arbitrary design choice. The table above shows a minimum size for the data types. These sizes are common to most of the embedded processors currently implemented in radio equipement but can vary. The minimum has been selected in order to ensure functionality. Nevertheless, depending upon the hardware Short, Long, Ushort and Ulong might be of a larger size and more accurate.

Composed types

The *Composed types* possibly used within the specification are limited to the following possibilities:

- Tables of base types
- Structures of base types

Domain types

Domain types are associated with domain-specific types, applicable for *Notions* and *Constraints*. They are generally attached to physical quantities, and specify the base unit associated.

Types setting makes the association between a domain type and a base type. This shall be used when a numerical representation of a *Notion* or *Constraint* is necessary, or to use an operation attached to the domain type.

A default setting is systematically defined for each domain type. Supplementary settings may be defined.

Unless otherwise specified, any implementation of a given domain type shall be based on the default setting.

The domain types are defined within *Common Concepts*, with the exception of *Timing Types*, presented later in this chapter.

Types settings for Notions and Constraints

Notions and *Constraints* of the Transceiver are generally first defined using an abstract type, representing the physical value attached to the concept. Then, for the implemented ones, a data type is associated to the concept type.

Types settings for Arguments

Arguments of programming interfaces operation require a functional definition subject to be translated into programming language data type, since *Waveform Application* software will interact with the *Transceiver Subsystem* according to information carried within the arguments.

2.3.3.2 Implementation specific types

For each considered implementation language, the association of base type definitions to types available within the implementation language is defined.

2.3.4 States and Transitions

All transitions identified between states are instantaneous.

They are given a specific formal name.

A certain occurrence time can be attached to each transition. Occurrence times of a transition are identified by the formal identifier of the transition it relates to, with “Time” appended. A specific occurrence is identified by its count index.

2.3.5 Events Handling

This chapter exposes the concepts used in the *Facility* to specify requirements involving events handling.

2.3.5.1 Events Timely Control

This chapter exposes how the *Facility* introduces *timely control operations*, through which the *Waveform Application* can require internal events of the *Transceiver Subsystem* to occur at explicitly specified instants.

The *timely control operation* is invoked at *Invocation Time*, the *Caller* being the *Waveform Application*, the *Callee* being the *Transceiver Subsystem*.

Target Event denotes the event attached to such a *timely control operation*. *Target Event Time* denotes the effective occurrence time of *Target Event*, which shall be close to *requestedEventTime*.

The following concepts are attached to characterization of *timely control operations*:

- Argument *requestedEventTime*,
- Constraint *Event Accuracy*,
- Constraint *Min Anticipation*.

The argument *requestedEventTime*, of generic domain type Time, is used by *timely control operations* to specify at which time *Target Event* shall happen. Put simply, it is the time at which the Transceiver must fulfill an operation, typically a transmission start/stop or a reception start/stop. Refer to $4.2 Time handling Domain Types for generic domain type Time description.

The constraint *Event Accuracy*, of domain type *Latency*, is specified by the *Facility* to capture the time accuracy to be fulfilled by *Target Event Time*, which shall belong to the time interval defined by *requestedEventTime* plus or minus *Event Accuracy* value.

The constraint *Min Anticipation*, of domain type *Latency*, characterizes the anticipation needed for *Invocation Time* of *timely control operations* in front of the specified *Target Event Time*. *Invocation Time* shall occur before *requestedEventTime* minus *Min Anticipation*.

The following figure illustrates the concepts introduced:

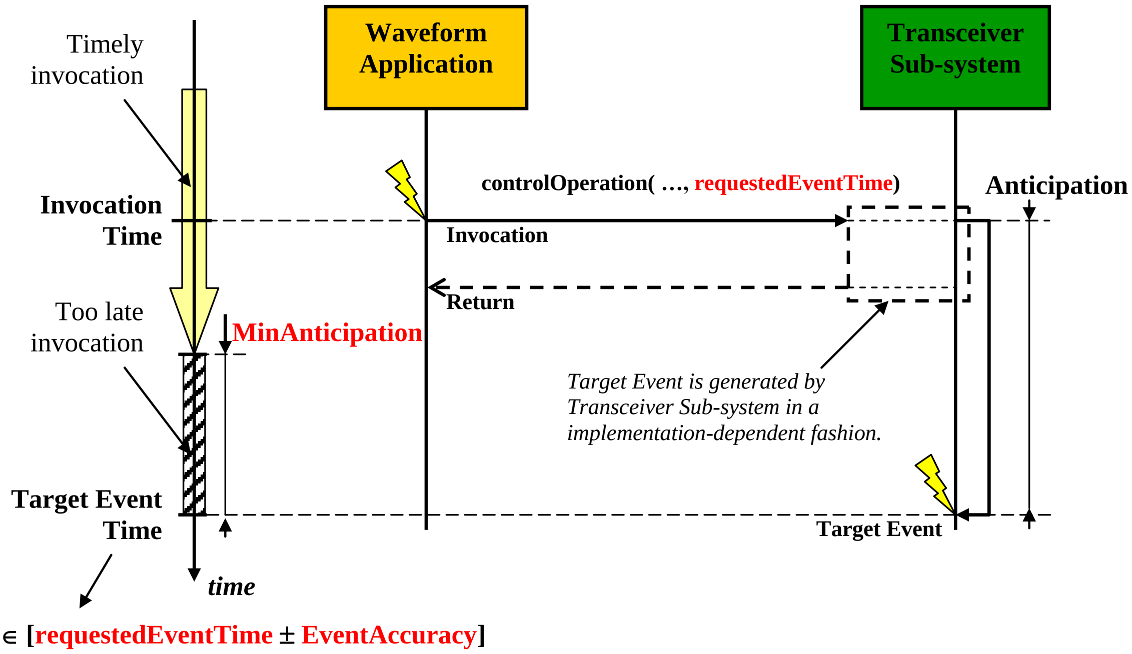


Figure 9: Expression of Timely Events

2.3.5.2 Referencing Event Sources

This chapter exposes how reference *event sources* are identified.

Referencing such *event sources* is a possibility for *timely control operations* to request event times, as exposed in *common concepts*.

Principle

Event sources correspond to events of the *Transceiver Subsystem* which are defined by the *specification* and thus useable for the engineering of *Waveform Capabilities*.

The *Waveform Application* can reference, in control messages, the *Event Sources* of the *Transceiver Subsystem*. It may as well reference *Event Sources* belonging to other radio Subsystems of the *Platform Support* attached to the *Waveform Application*.

Platform Support signifies the complete set of *Radio Subsystems* with which the *Waveform Application* is interacting in order to implement a certain *Waveform Capability*. The *Transceiver Subsystem* is one *Radio Subsystem* of the *Platform Support*.

<u>Examples of Event Sources</u>

> ***RF Transmit Start*** *and* ***RF Transmit Stop*** *are event sources attached to a* ***Transmit Channel****, while* ***RF Receive Start*** *and* ***RF Receive Stop*** *are attached to a* ***Receive Channel****.*

Event Sources Public Identifiers

A unique *Public Identifier* is assigned to each *Event Source* of the *Platform Support*. This assignment is specific to each *Waveform Capability*, and can be different from one *Waveform Cabability* to another. The *deployment* of the *Platform Support* is in charge to assign the desired *Public Identifiers* values.

The *Public Identifier* values are used by arguments of the control messages sent by the *Waveform Application* to the *Transceiver Subsystem*. This enables the *Waveform Application* to specify accurate real-time behaviors of the *Transceiver Subsystem* using any available *Event Source* of the *Platform Support*. The usable *Event Sources* may therefore not be limited to those of the *Transceiver Subsystem*.

The following figure summarizes the principle of Event Sources Public Identifiers:

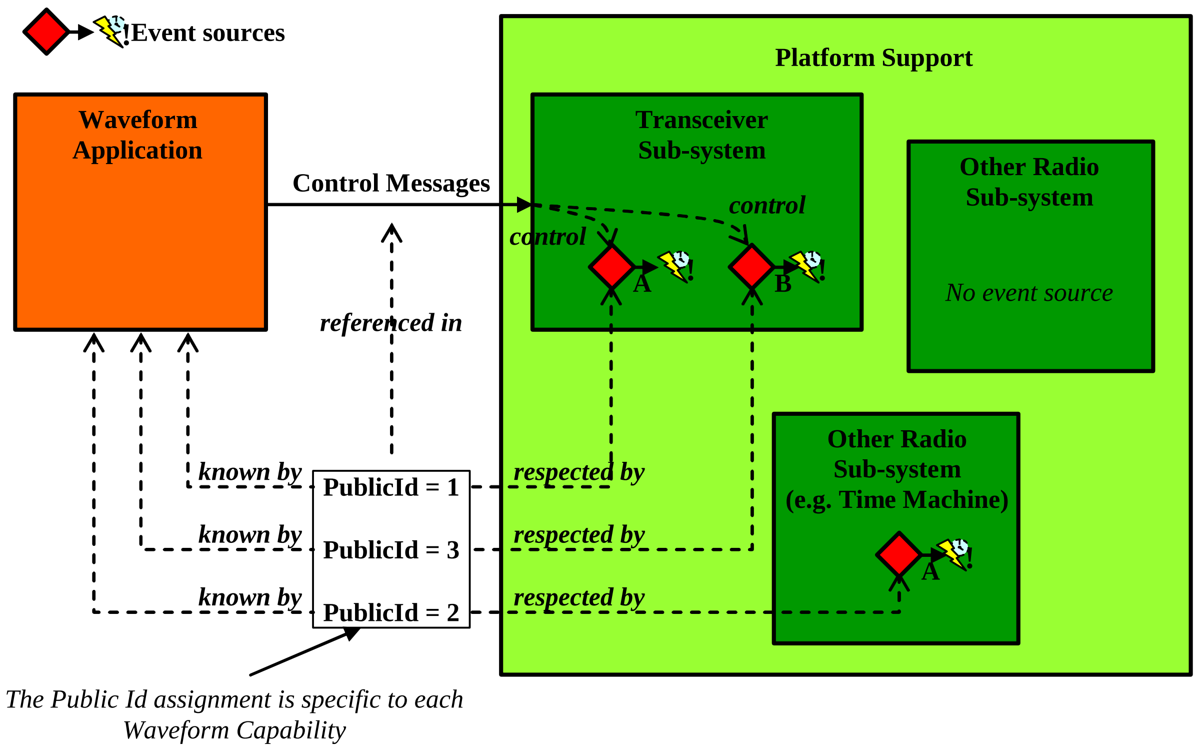


Figure 10: Event Sources Public Identifiers

Event Occurences

Event Sources generate *Event Occurrences*.

Each *Event Source* counts the generated *Event Occurences* using an *unsigned 32 bit* value. This enables count values ranging from 1 to 4,294,967,295. the value 0 remaining unused.

Event Occurences count values can be used as arguments in control messages sent by the *Waveform Application* to specify when particular events shall happen in the *Transceiver Subsystem*.

2.3.5.3 Last and Next Events

The notion of *last* and *next* event of a certain *event source* is defined relative to the instant when the *control operation* referring to such events is invoked. *Invocation Time* and *Event Time* denote the time of corresponding events.

To enable unambiguous identification of a *previous* event, the *Invocation Time* referencing the event shall happen after a margin of time, denoted *Real-time Accuracy*. Otherwise, guaranteeing that existence of the previous event is taken into account for correct realisation of the *control operation* is not possible.

Similarly, unambiguous identification of the *next* event requires *Invocation Time* to happen with at least *Real-time Accuracy* before the target *Event Time*. A *constraint* of category *Min Event Proximity*, of domain type *Latency*, is used to capture the time margin needed between the *Invocation Time* of any invoked operation and a *previous* (resp. *next*) *Target Event*, where the *Invocation Time* shall happen at least *Min Event Proximity* before (resp. *after*) *Event Time*.

The formal requirements using *Min Event Proximity* are specified by the *Facility* for each concerned *timely control operation* for a given referenced event source.

The real-time engineering of the *Waveform Application* shall ensure that the *Invocation Time* of the *timely control operation* never falls in the ambiguity period characterized by *Min Event Proximity*.

3 - Features Specification

3.1 - Transceiver composition

The *Transceiver Subsystem* can be composed of the following *Core Features*, as explained in chapter *Internal Structure*:

- Transmit Channel,
- Receive Channel.

The following implicit notions directly attached to the *Transceiver Subsystem* are characterizing what is specifically expected in terms of channels composition.

As the *Transceiver Subsystem* considered by the *Facility* is attached to a specific radio capability, in total abstraction of the implementation assumption, the aspects covering multi-channel *Transceiver Subsystems* are not addressed.

FEAT_GEN__010 [Used "Transmit Channel"

The implicit notion *UsedTransmitChannel*, of type *Boolean*, **shall** be used to specify if the considered radio capability is requesting usage of a *Transmit Channel* core feature.

Value *True* in case a *Transmit Channel* is needed, *False* otherwise, may not be left undefined.

] END_FEAT_GEN_010

FEAT_GEN _020 [Used "Receive Channel"

The implicit notion *UsedReceiveChannel*, of type *Boolean*, **shall** be used to specify if the considered radio capability is requesting usage of a *Receive Channel* core feature.

Value *True* in the case where a *Transmit Channel* is needed, *False* otherwise, may not be left undefined.

] END_FEAT_GEN_020

FEAT_GEN_ 030 [Half-duplex Transceiver

The implicit notion *HalfDuplexTransceiver*, of type *Boolean*, **shall** be used to specify if the considered radio capability is requesting usage of a half-duplex *Transceiver*.

Value *True* in the case where the *Transceiver* is half-duplex, *False* otherwise (full-duplex). Can remain undefined only if a simplex *Transceiver* is required (i.e. *UsedReceiveChannel* or *UsedTransmitChannel* set to *False*).

] END_FEAT_GEN_030

Note: It is worth noting that the Transmit Channel and Receive Channel are the elementary features of any Transceiver hence their classification as core features. Half-Duplex is an optional feature and it is not elaborated in the current version of the Facility. Other features such as MIMO capability might be also considered as optional and be introduced in future versions of the document. Distinguishing between Core and Optional featues, as mentioned in previous chapters, is intended for specification extensibility.

3.2 - Core Feature "Transmit Channel"

3.2.1 Principle

For radio transmission, a *Transmit Channel* of a *Transceiver Subsystem* is up-converting bursts of input *Baseband Signal* into bursts of *RF Signal*. A *Transmit Cycle* denotes the phase corresponding to the up-conversion of a particular signal burst.

A *Transmit Channel* implementation is namely composed of the following specific sub-items:

- A *Baseband FIFO*
- An *Up-conversion Chain*

The *Up-conversion Chain* is the signal processing chain which performs the up-conversion and most probably the upsampling and filtering of the burst of *Baseband Signal* contained in *Baseband FIFO*, outputting it as a burst of *RF Signal*.

The programming interface *TransmitDataPush* enables the samples packets to be pushed by the *Waveform Application* towards the *Transmit Channel*. The *Transmit Channel* takes in charge the packets pushed by *Waveform Application*, stores them into *Baseband FIFO*, where *Up-conversion Chain* consumes them in real-time to upconvert them and generate the *RF Signal*. The samples need to be available on time in *Baseband FIFO* for the up-conversion process to start, and need to be continuously stored in *Baseband FIFO* in a timely manner that prevents signal interruption occurrence.

The programming interface *TransmitControl* enables the *Waveform Application* to manage when and how *Transmit Cycles* shall occur. For each *Transmit Cycle*, this control is based on specification of when the concerned burst shall start and stop, and through characterization of the applicable *Tuning Profile*, which characterizes the exact signal processing transformation to be applied to *Baseband Signal* by *Up-conversion Chain* on the concerned burst in order to generate the *RF Signal*.

The following figure summarizes the previous concepts:

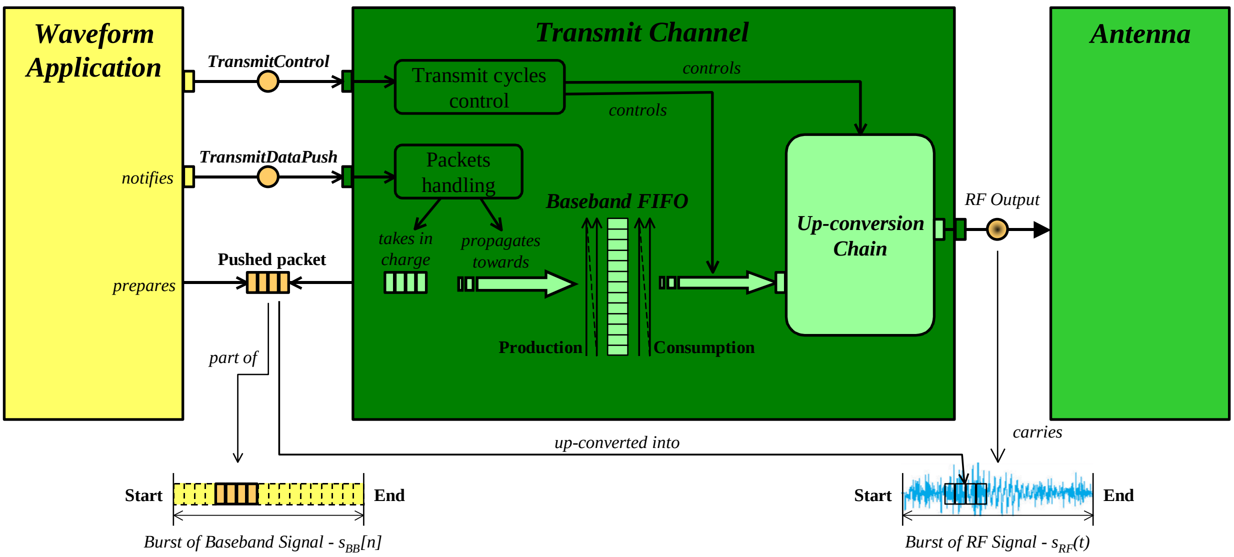


Figure 11: Overview of a Transmit Channel

3.2.2 Overview

3.2.2.1 Programming interfaces overview

The following table provides an overview of the programming interface *TransmitControl*:

Signature summary (pseudo-code)	Used by	Realized by	Description

`createTransmitCycleProfile(`
`Time requestedTransmitStartTime,`
`Time requestedTransmitStopTime,`
`UShort requestedPresetId,`
`Frequency requestedCarrierFrequency,`
`AnaloguePower requestedNominalRFPower)`
	Waveform Application	Transceiver Subsystem	Creation of a Transmit Cycle Profile.

`configureTransmitCycle(`
`Ulong targetCycleId,`
`Time requestedTransmitStartTime,`
`Time requestedTransmitStopTime,`
`Frequency requestedCarrierFrequency,`
`AnaloguePower requestedNominalRFPower)`
	Waveform Application	Transceiver Subsystem	Configuration of an existing Transmit Cycle Profile.

`setTransmitStopTime(`
`Ulong targetCycleId,`
`Time requestedTransmitStopTime)`
	Waveform Application	Transceiver Subsystem	Specification of the end time of a Transmit Cycle.

Table 8: Overview of programming interface TransmitControl

The following table provides an overview of the programming interface *TransmitDataPush*:

Signature summary (pseudo-code)	Used by	Realized by	Description

`pushBBSamplesTx(`
`BBPacket thePushedPacket,`
`Boolean endOfBurst)`
	Waveform Application	Transceiver Subsystem	Notifies availability of a baseband samples packet.

Table 9: Overview of programming interface TransmitDataPush

3.2.2.2 Characteristics overview

Explicit Notions

Explicit Notions defined in §2.3 are *observable*, *implementation independent* and *exchanged* characteristics of the *Transceiver Subsystem*. The following table summarizes the *Explicit Notions* related to *Transmit Channel*:

Explicit Notion Name	Defined in	Nature	Exchange Mechanism	Associated Argument	Associated Property
`TransmitCycle`	Core Feature	Logical	Programmable	`targetCycleId`	n.a.
`TransmitStartTime`	Core Feature	Real-Time	Programmable	`requestedTransmitStartTime`	n.a.
`TransmitStopTime`	Core Feature	Real-Time	Programmable	`requestedTransmitStopTime`	n.a.
`TuningPreset`	Core Feature	Logical	Programmable	`requestedPresetId`	n.a.
`CarrierFrequency`	Common Concept	Signal Processing	Programmable & Configurable	`requestedCarrierFrequency`	`CarrierFrequency`
`NominalRFPower`	Common Concept	Signal Processing	Programmable & Configurable	`requestedNominalRFPower`	`NominalRFPower`

Table 10: Explicit Notions related to Transmit Channel

Implicit Notions

Implicit Notions defined in §2.3 are *observable*, *implementation independent* and *not exchanged* characteristics of the *Transceiver Subsystem*. The following table summarizes the *Implicit Notions* related to *Transmit Channel*:

Implicit Notion Name	Defined in	Nature
`BasebandFIFOSize`	Core Feature	Logical
`MaxPushedPacketSize`	Core Feature	Logical
`OverflowMitigation`	Core Feature	Logical
`TuningStartThreshold`	Core Feature	Logical

`BasebandSignal:…`
`BasebandSamplingFrequency`
`BasebandCodingBits`
`BasebandNominalPower`
	Common Concept	Signal Processing

Table 11: Implicit Notions related to Transmit Channel

Internal Notions

Internal Notions defined in §2.3 are *observable* and *implementation dependent* characteristics of the *Transceiver Subsystem*. The following table summarizes the *Internal Notions* related to *Transmit Channel*:

Name	Defined in	Nature	Transceiver Design Formulas
`TuningDuration`	Core Feature	Real-time	`< MaxTuningDuration`
`TuningStartTime`	Core Feature	Real-time	`= ConsumptionStartTime – TuningDuration` `∈ [ConsumptionStartTime - MaxTuningDuration; ConsumptionStartTime]`
`UpconversionLatency`	Core Feature	Real-time	`< MaxUpConversionLatency`
`ConsumptionStartTime`	Core Feature	Real-time	`= TransmitStartTime - UpConversionLatency` `∈ [TransmitStartTime - MaxUpConversionLatency; TransmitStartTime]`
`ConsumptionStopTime`	Core Feature	Real-time	`= TransmitStopTime – UpConversionLatency` `∈ [TransmitStopTime - MaxUpConversionLatency; TransmitStopTime]`
`ReactivationTime`	Core Feature	Real-Time	`= TransmitStartTime(n) – TransmitStopTime(n-1)` `> MinReactivationTime`
`TxChannelTransferFunction`	Common Concept	Signal Processing	Fits in mask defined by `ChannelMask`

Table 12: Internal Notions related to Transmit Channel

Constraints

Constraints defined in §2.3 are ***non**-observable* characteristics which constrain the *Transceiver Subsystem* and/or *Waveform Application* implementation. The following table summarizes the *Constraints* related to *Transmit Channel*:

Name of the Constraint	Nature
`MaxTuningDuration`	Real-time
`MaxUpconversionLatency`	Real-time
`MinTransmitStartProximity`	Real-time
`MinTransmitStartAnticipation`	Real-time
`TransmitTimeProfileAccuracy`	Real-time
`MinTransmitStopAnticipation`	Real-time
`MinTransmitStartProximity`	Real-time
`MaxTransmitDataPushInvocationDuration`	Real-time
`MinPacketStorageAnticipation`	Real-time
`MinReactivationTime`	Real-time
`ChannelMask:`…	Signal Processing
`SpectrumMask:`…	Signal Processing
`GroupDelayMask:`…	Signal Processing
`MaxCycleId`	Logical
`MaxTxCycleProfiles`	Logical

Table 13: Constraints related to Transmit Channel

3.2.3 Analysis Requirements

The mnemonic used for identification of requirements of Transmit Channel feature is: “TX_CHAN”.

3.2.3.1 Transmit Channel

FEAT_TX_CHAN_ANA_010 [Definition of a “Transmit Channel”

The core feature *Transmit Channel* **shall** be used to refer to the capabilities provided by the *Transceiver Subsystem* component that performs up-conversion of a succession of input *Baseband Signal* bursts into output analogue *RF Signal* bursts.

] END_FEAT_TX_CHAN_ANA_010

FEAT_TX_CHAN_ANA_020 [Creation of a “Transmit Channel”

The *deployment* of a *Transmit Channel* and all its constituents **shall** be undertaken by *Transceiver Subsystem deployment*, following implementation specific choices.

The state of a *Transmit Channel* just after its creation and any of its constituent **shall** be state *Deployed*.

The initialization of the *Transmit Channel* shall be conducted by *deployment*.

] END_FEAT_TX_CHAN_ANA_020

The steps setting one *Up-conversion Chain* into state *Deployed* are realized by (re)configuration mechanisms proper to the *Transceiver Subsystem* implementation, which are out of the scope of this *specification*.

FEAT_TX_CHAN_ANA_030 [Composition of a “Transmit Channel”

The following concepts **shall** be integrated as constituents of a given *Transmit Channel*:

- Concept *Up-conversion Chain*,
- Concept *Transmit Baseband Signal FIFO*.

] END_FEAT_TX_CHAN_ANA_030

They are defined in their respective definition requirements as specified in the following paragraphs.

3.2.3.2 Transmit Baseband Signal

The *Transmit Baseband Signal* is the useful signal provided by the *Waveform Application* to the *Transmit Channel* in order to be transmitted over the air.

Terms and definitions used in this chapter are based on those presented within Common Concepts *Baseband Interface 4.2.4 section*.

FEAT_TX_CHAN_ANA_040 [Definition of “Transmit Baseband Signal”

The concept *Transmit Baseband Signal* **shall** describe the useful baseband signal sent by the *Waveform Application* to a *Transmit Channel* for up-conversion.

] END_FEAT_TX_CHAN_ANA_040

Useful baseband signal corresponds to the signal useful to be transmitted on the antenna, and received by the receiver.

Useful baseband signal can include the following possible information:

- Power rising pattern
- Power falling pattern
- ALC and/or AGC patterns
- Data and eventually WF specifics pattern (sequences for synchronization)

FEAT_TX_CHAN_ANA_050 [Implicit Notions of "Transmit Baseband Signal"

A *Transmit Baseband Signal* **shall** be characterized using the following notions, defined by common concept *Baseband Signal*.

- Implicit Notion *Baseband Coding Bits*
- Implicit Notion*Baseband Nominal Power*

] END_FEAT_TX_CHAN_ANA_050

FEAT_TX_CHAN_ANA_060 [Transmit Baseband Signal configuration

The properties of *Transmit Baseband Signal* **shall** be set to values appropriate for the considered waveform during Transmit Channel deployment.

] END_FEAT_TX_CHAN_ANA_060

3.2.3.3 Transmit Baseband FIFO

The *Transmit Baseband FIFO* contains baseband signal samples provided by the *Waveform Application* in order for them to be consumed by the *Up-conversion Chain*.

Consumption of baseband samples in the *Transmit Baseband FIFO* is realized in a packet mode.

The FIFO monitors samples availability in order to raise error notification in case the samples needed by the Up-conversion chain would not be available (underflow situation).

FEAT_TX_CHAN_ANA_070 [Definition of "Transmit Baseband FIFO"

The terminology *Transmit Baseband FIFO* **shall** be associated with the part of the sub-part of the *Transmit Channel* where the baseband samples of the bursts pushed by the *Waveform Application* are stored, waiting to be consumed by the *Up-conversion Chain*.

] END_FEAT_TX_CHAN_ANA_070

FEAT_TX_CHAN_ANA_080 [Implicit Notion "BasebandFIFOSize"

The Implicit Notion *BasebandFIFOSize* **shall** be the *Ulong* value which captures the size of *Transmit Baseband FIFO*, capturing the maximum number of baseband samples possibly stored inside the *Baseband FIFO*.

] END_FEAT_TX_CHAN_ANA_080

FEAT_TX_CHAN_ANA_090 [Implicit Notion "TuningStartThreshold"

The Implicit Notion *TuningStartThreshold* **shall** be the *Ulong* value which defines the threshold applicable to the number of samples received in the *Transmit Baseband FIFO* within a transmitted baseband samples burst before *Tuning* transition is activated.

] END_FEAT_TX_CHAN_ANA_090

Remark: This active phase triggering mode is useful for waveforms having no specific requirement concerning the instant when *RFTransmitStart* shall happen.

FEAT_TX_CHAN_ANA_100 [Default value of “TuningStartThreshold”

The Implicit Notion *TuningStartThreshold* **shall** have a default value of 0, meaning the Waveform does not use this threshold to set up when *RFTransmitStart* shall happen.

] END_FEAT_TX_CHAN_ANA_100

Keeping the default value means that *Tuning* activity starts immediately after *first sample* has been received, which means further samples must arrive sufficiently fast from the Waveform Application.

FEAT_TX_CHAN_ANA_110 [Configuration of “Transmit Baseband FIFO”

The properties of *Transmit Baseband FIFO* **shall** be set to the values required by the considered *Waveform Application* during the *Transmit Channel* deployment.

] END_FEAT_TX_CHAN_ANA_110

Availability deadline and Underflow situations

Any sample of the burst shall be present within baseband FIFO when the up-conversion needs to consume them. An *availability deadline* is therefore attached to each sample to be transmitted within a cycle.

The *availability deadline* of the *first transmitted sample* is equal to *ConsumptionStartTime* which is derived from the *TransmitStartTime* and *UpConversionLatency*. The availability deadline of each next sample is an incrementof the baseband signal period per new sample, namely the time between two consecutive samples.

In case any sample to be transmitted would not be available within the FIFO before its *availability deadline*, the Transmit Channel goes into a *SignalUnderflow* situation. This is an error situation.

FEAT_TX_CHAN_ANA_120 [Error “FIFOUnderflow”

During the *Active Phase* of a *Transmit Cycle*, an error *ERR_FIFOUnderflow* **shall** be generated if the FIFO runs into underflow situtation.

] END_FEAT_TX_CHAN_ANA_120

FEAT_TX_CHAN_ANA_130 [FIFO Underflow TransmitStop

During the *active* state of a *Transmit Cycle*, if *Baseband FIFO* runs into underflow situation, the transmission **shall** be stopped by initiating a *TransmitStop* transition, immediately after the last sample pushed into *baseband FIFO* has been consumed by *Up-conversion Chain*.

] END_FEAT_TX_CHAN_ANA_130

3.2.3.4 Up-conversion Chain

FEAT_TX_CHAN_ANA_140[Definition of the “Up-conversion Chain”

Concept *Up-conversion Chain* **shall** denote the signal processing chain in the Transmit Channel which undertakes, during periods of time denoted as *Active Transmit Cycles*, continuous transformation of one input baseband signal *burst* into the corresponding RF signal *burst*.

] END_FEAT_TX_CHAN_ANA_140

FEAT_TX_CHAN_ANA_150 [Constraint “MaxUpconversionLatency”

The constraint *MaxUpconversionLatency* of type *Latency* **shall** be used to characterize the maximum allowed elapsed time between the instant a sample is consumed by the Up-conversion Chain from the Baseband FIFO and the instant this sample is on the RF output.

] END_FEAT_TX_CHAN_ANA_150

This value has to be taken into consideration by the waveform application engineering process. The term *Up-conversionLatency* is used in the following sections as an Internal Notion which is implementation related and does not exceed the above constraint.

States and Transitions of Up-conversion Chain

The following figure summarizes the concepts addressed in the following paragraphs:

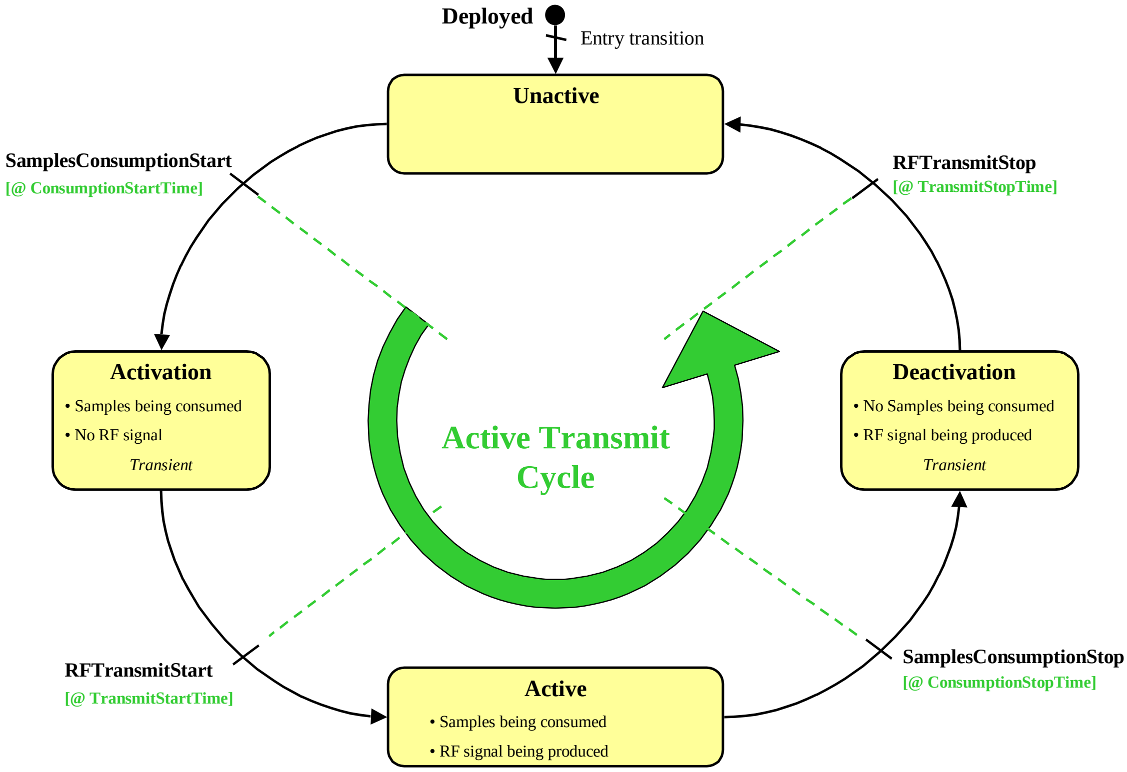


Figure 12: States and transitions of the Up-conversion Chain

FEAT_TX_CHAN_ANA_160 [States of Up-conversion Chain

The following states **shall** be implemented by an *Up-conversion Chain*:

- *Deployed*: Initial state of the chain once *deployed* according to considered Waveform requirements.
- *Unactive*: The state during which the *Up-conversion Chain* is not consuming samples within *baseband FIFO*, and is not outputing any signal at RF level. Automatically reached once *Up-conversion Chain* is *Deployed*.
- *Activation*: The transient state during which the *Up-conversion Chain* has started to consume input baseband signal, while it is not yet outputing the corresponding useful signal at RF level. Arrival of the useful baseband signal at RF level corresponds to the end of the state.
- *Active*: The state during which the input baseband signal is continuously consumed by the *Up-conversion Chain*, and transformed into the associated RF signal which is being produced by the chain at its output.
- *Deactivation*: The transient state during which Tx baseband signal is not consumed anymore by the *Up-conversion Chain*, while the previously consumed signal is still being processed by the Up-conversion with associated signal produced. End of this useful signal processing corresponds to the end of the state.

] END_FEAT_TX_CHAN_ANA_160

Activation and *Deactivation* are transient states which have a duration equal to the *Up-conversionLatency*, which is the time elapsed between (i) the instant when consumption of a given sample occurs from the *Baseband FIFO* and (ii) the instant when the corresponding RF signal is transmitted. A *Tuning* activity, not depicted in the preceding figure, is the step during which the Waveform Application shall configure a *Transmit Cycle* for the transmission of the next burst.

FEAT_TX_CHAN_ANA_170 [Transitions between “Up-conversion Chain” states

The instantaneous transitions between states of an Up-conversion Chain **shall** respect the following list:

- *SamplesConsumptionStart*: transition between states *Unactive* and *Activation*
- *RFTransmitStart***:** transition between states *Activation* and *Active*
- *SamplesConsumptionStop***:** transition between states *Active* and *Deactivation*
- *RFTransmitStop*: transition between states *Deactivation* and *Unactive*

] END_FEAT_TX_CHAN_ANA_170

FEAT_TX_CHAN_ANA_180 [Timings associated to Up-conversion Chain transitions

The following timings **shall** be associated with instants when instantaneous transitions of Up-conversion Chain are happening:

- *ConsumptionStartTime*: associated to transition *SamplesConsumptionStart*
- *TransmitStartTime*: associated to transition *RFTransmitStart*
- *ConsumptionStopTime*: associated to transition *SamplesConsumptionStop*
- *TransmitStopTime*: associated to transition *RFTransmitStop*.

] END_FEAT_TX_CHAN_ANA_180

FEAT_TX_CHAN_ANA_190 [Tuning activity

The *Tuning* activity **shall** be used to configure the signal processing requirements captured in *Tuning Profile*. This activity is characterized by the time it starts, *TuningStartTime*, and its duration, *TuningDuration*.

] FEAT_TX_CHAN_ANA_190

FEAT_TX_CHAN_ANA_200 [Constraint "MaxTuningDuration"

The Constraint *MaxTuningDuration* **shall** be used to characterize the maximum allowable constant duration of the *Tuning* activity.

] END_FEAT_TX_CHAN_ANA_200

It is the boundary for internal notion *TuningDuration*

FEAT_TX_CHAN_ANA_210 [Internal Notion "TuningStartTime"

The internal notion *TuningStartTime* **shall** be used to specify the beginning of the *Tuning* activity. Nominal use of *TuningStartTime* is described by the relationships below:

$$TuningStartTime[n+1] = ConsumptionStartTime[n+1] - TuningDuration$$

$$TuningStartTime[n+1] \geq TransmitStopTime[n]$$

In rare cases the TuningStartTime[n+1] would be allowed to start before the TransmitStopTime[n], for example in case of duplicated channels for very fast frequency hopping applications.

] FEAT_TX_CHAN_ANA_210

The following figure summarizes the definitions introduced in this section:

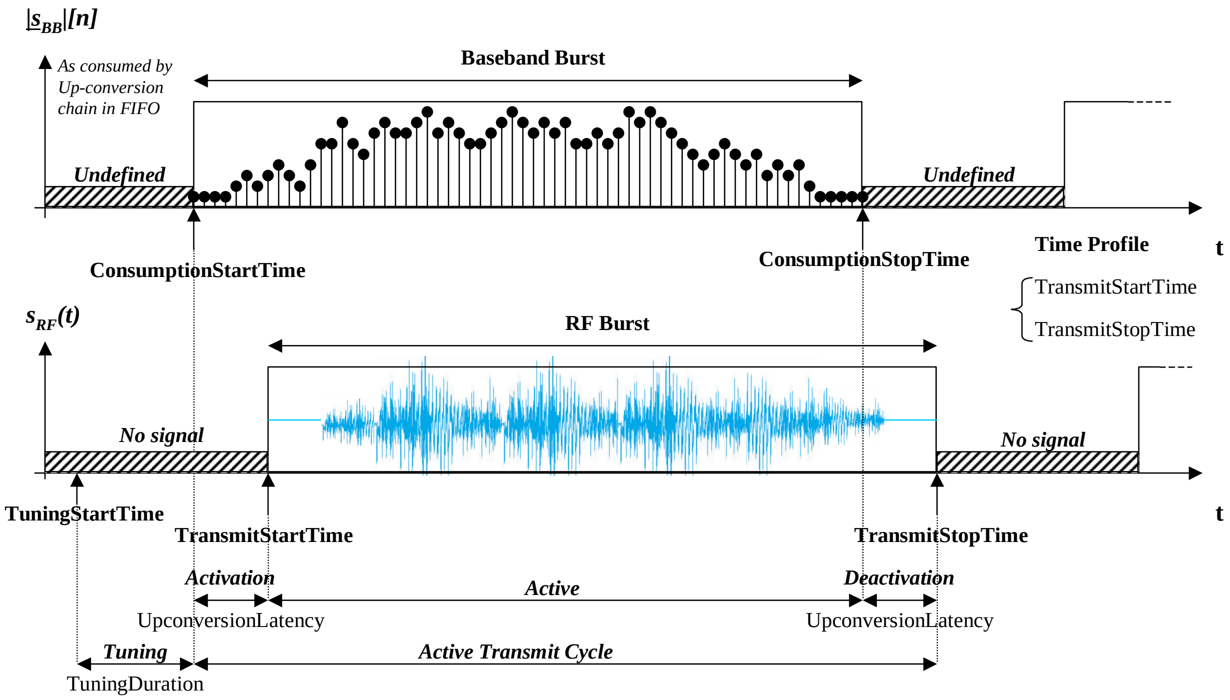


Figure 13: Time profile of Transmit Cycle

FEAT_TX_CHAN_ANA_220 [Constraint "MinReactivationTime"

The Constraint *MinReactivationTime* **shall** be used to quantify the minimum time elapsed between the TransmitStopTime of one given Transmit Cycle and the TransmitStartTime of the next one.

] END_FEAT_TX_CHAN_ANA_220

Identification of Current Cycle

Default event-based time requests are to be based on the notion of "Current" Transmit Cycle.

FEAT_TX_CHAN_ANA_230 [Definition of the "Current" Transmit Cycle

The notion of *current Transmit Cycle* **shall** be applied, for operations invoked by the *Waveform Application* making event-based time requests, as a function of the instant when the considered operation is invoked by *Waveform Application*.

] END_FEAT_TX_CHAN_ANA_230

FEAT_TX_CHAN_ANA_240 [Rule for "Current" Transmit Cycle identification

The *current Transmit Cycle* **shall** be the *Active Transmit Cycle*, if the operation is invoked when a Transmit Cycle is active, or the last *Transmit Cycle* that finished before invocation.

] END_FEAT_TX_CHAN_ANA_240

Uncertainty of identification of the *current Transmit Cycle* thus exists when invocation of *createTransmitCycle()* happens close to the *ConsumptionStartTime* of a certain cycle. It is up to *Waveform Application* design to take appropriate assumptions to avoid calls happening within this time zone.

3.2.3.5 Transmit Cycle

A compliant execution by the *Up-conversion Chain* of a given *Transmit Cycle* relies upon availability of two sorts of information:

- Control data, captured into the *Transmit Cycle Profile*
- Input data, denoted as the *Transmit Cycle Input Burst*

The *Transmit Cycle Input Burst* is the slice of baseband signal provided by the *Waveform Application* that the *Up-conversion Chain* will transform into RF signal during the considered *Transmit Cycle*, in compliance with the signal processing requirements captured in *Tuning Profile* and the real-time requirements captured in the *Time Profile*.

Transmit Cycle Input Burst

FEAT_TX_CHAN_ANA_250 [Definition of "Transmit Cycle Input Burst"

The *Transmit Cycle Input Burst* **shall** denote the un-interrupted slice of baseband signal provided by the *Waveform Application* for transmission during the considered *Transmit Cycle*.

] END_FEAT_TX_CHAN_ANA_250

FEAT_TX_CHAN_ANA_260 [Definition of "Transmit Cycle First Sample"

The *Transmit Cycle First Sample* **shall** denote the first sample of the considered input burst.

] END_FEAT_TX_CHAN_ANA_260

FEAT_TX_CHAN_ANA_270 [Definition of "Transmit Cycle Last Sample"

The *Transmit Cycle Last Sample* **shall** denote the last sample of the considered input burst.

] END_FEAT_TX_CHAN_ANA_270

The mechanisms provided to the *Waveform Application* for provision of the input bursts unambiguously identify the burst first and last samples. This enables synchronization of the *Up-conversion Chain* operation to meet accurate transmission start and end time requirements.

Transmit Cycle Profile

FEAT_TX_CHAN_ANA_280 [Definition of "Transmit Cycle Profile"

The concept of *Transmit Cycle Profile* **shall** be used to control the *Transmit Cycles* implemented during the life-time of one *Up-conversion Chain*. One specific *Transmit Cycle Profile* instance is created for each specific *Transmit Cycle* occurrence.

] END_FEAT_TX_CHAN_ANA_280

Several *Transmit Cycles Profiles* may be simultaneously available. The profile corresponding to an *Active Phase* of the *Up-conversion Chain* is the only profile active at this time.

FEAT_TX_CHAN_ANA_290 [Composition of "Transmit Cycle Profile"

A *Transmit Cycle Profile* **shall** be composed of a *Cycle Identifier*, a *Time Profile* and a *Tuning Profile*.

] END_FEAT_TX_CHAN_ANA_290

Cycle identifier

FEAT_TX_CHAN_ANA_300 [Explicit Notion "TransmitCycle"

Each created cycle has a unique integer identifier *TransmitCycle* which **shall** be set up during creation to a value incremented by one for each newly created Transmit Cycle, starting at 0 (ZERO) for the first created cycle and reaching the value of the constraint *MaxCycleId* before restarting counter to 0.

] END_FEAT_TX_CHAN_ANA_300

FEAT_TX_CHAN_ANA_310 [Constraint "MaxTxCycleProfiles"

The constraint *MaxTxCycleProfiles* **shall** refer to the maximum number of *Transmit Cycles* simultaneously existing at any time within the Transmit Channel.

] END_FEAT_TX_CHAN_ANA_310

Time Profile

The *Time Profile* is set by the *Waveform Application* to define time positioning of the *Transmit Cycle*. It is composed of *Explicit Notions TransmitStartTime* and *TransmitStopTime*.

They correspond to the only externally measurable instants attached to a *Transmit Cycle* which are not dependent on the *Up-conversion Chain* implementation.

FEAT_TX_CHAN_ANA_320 [Definition of "Time Profile"

A *Time Profile* **shall** be composed of *explicit notions TransmitStartTime* and *TransmitStopTime*.

] END_FEAT_TX_CHAN_ANA_320

FEAT_TX_CHAN_ANA_330 [Constraint "TransmitTimeProfileAccuracy"

The constraint "TransmitTimeProfileAccuracy" **shall** refer to the *Event Accuracy* associated with the *Time Profile* explicit notions, namely *TransmitStartTime* and *TransmitStopTime*.

] END_FEAT_TX_CHAN_ANA_330

FEAT_TX_CHAN_ANA_340 [Explicit Notion "TransmitStartTime"

The *Explicit NotionTransmitStartTime* **shall** be used within *Time Profile* to contain the *Transmit Start Time* of the corresponding *Transmit Cycle*.

] END_FEAT_TX_CHAN_ANA_340034

FEAT_TX_CHAN_ANA_350 [Referenced event source “TransmitStart”

The “TransmitStart” event, associated with the *Transmit Start Time* explicit notion, **shall** be the unique *Referenced Event Source* of the Transmit Channel.

] END_FEAT_TX_CHAN_ANA_350

FEAT_TX_CHAN_ANA_360 [Explicit Notion “TransmitStopTime”

The *Explicit Notion TransmitStopTime* **shall** be used within *Time Profile* to contain the *Transmit Stop Time* of the corresponding *Transmit Cycle*.

] END_FEAT_TX_CHAN_ANA_360

Explicit Notions *TransmitStartTime* and *TransmitStopTime* obey accuracy constraints directly imposed by radio link interoperability conventions between transmitting and receiving radio nodes. Low accuracy on those values is requested for waveforms such as fixed frequency AM/FM speech waveforms, while high accuracy is needed for frequency hopping digitally modulated waveforms.

TransmitStartTime and *TransmitStopTime* may be explicitly set by the *Waveform Application* by means of operations arguments. They belong to the Programmable subtype.

Their values can be measured thanks to external measurements conducted on a transmitting radio set without any knowledge of the *Transceiver Sub-system* internal implementation.

Implementation-dependent time Internal Notions

Implementation-dependent time Internal Notions necessary for analysis may be derived from the *Explicit Notions* introduced beforehand, using implementation-dependent durations or latencies:

- *ConsumptionStartTime* equals to *TransmitStartTime* less *UpconversionLatency*
- *TuningStartTime* equals *ConsumptionStartTime* less *TuningDuration*
- *ConsumptionStopTime* equals *TransmitStopTime* less *UpconversionLatency*

Refer to Figure 13: Time profile of Transmit Cycle for a graphical representation of above concepts

Tuning Profile

FEAT_TX_CHAN_ANA_370 [Definition of “Transmit Tuning Profile”

A *Transmit Tuning Profile* **shall** be used to characterize which signal processing explicit notions values shall be applied by the *Up-conversion Chain* when the *Transmit Cycle* is activated.

] END_FEAT_TX_CHAN_ANA_370

These explicit notions will be fixed to the requested values and applicable for the complete duration of the *Transmit Cycle*.

FEAT_TX_CHAN_ANA_380 [Contents of “Tuning Profile”

The following *explicit notions* are contained within the *Tuning Profile*:

- CarrierFrequency
- Nominal RF Power
- Tuning Preset

] END_FEAT_TX_CHAN_ANA_380

From one *Transmit Cycle* to the next, depending on the deployed waveform needs, the *Tuning Profile* may be totally different, partially different, or identical.

Examples

> *In fixed frequency TDD mode with unique channelization, the Tuning Profile is identical from one transmit cycle to another.*

> *In advanced frequency hopping waveforms, the CarrierFrequency or the ChannelMask can be changed from one Transmit Cycle to another, causing different Tuning Profiles.*

States and Transitions of Transmit Cycle Profile

The following figure shows how, for *Active Transmit Cycle* number *n* of a given *Up-conversion Chain*, active states of the corresponding *Transmit Channel Profile* are defined:

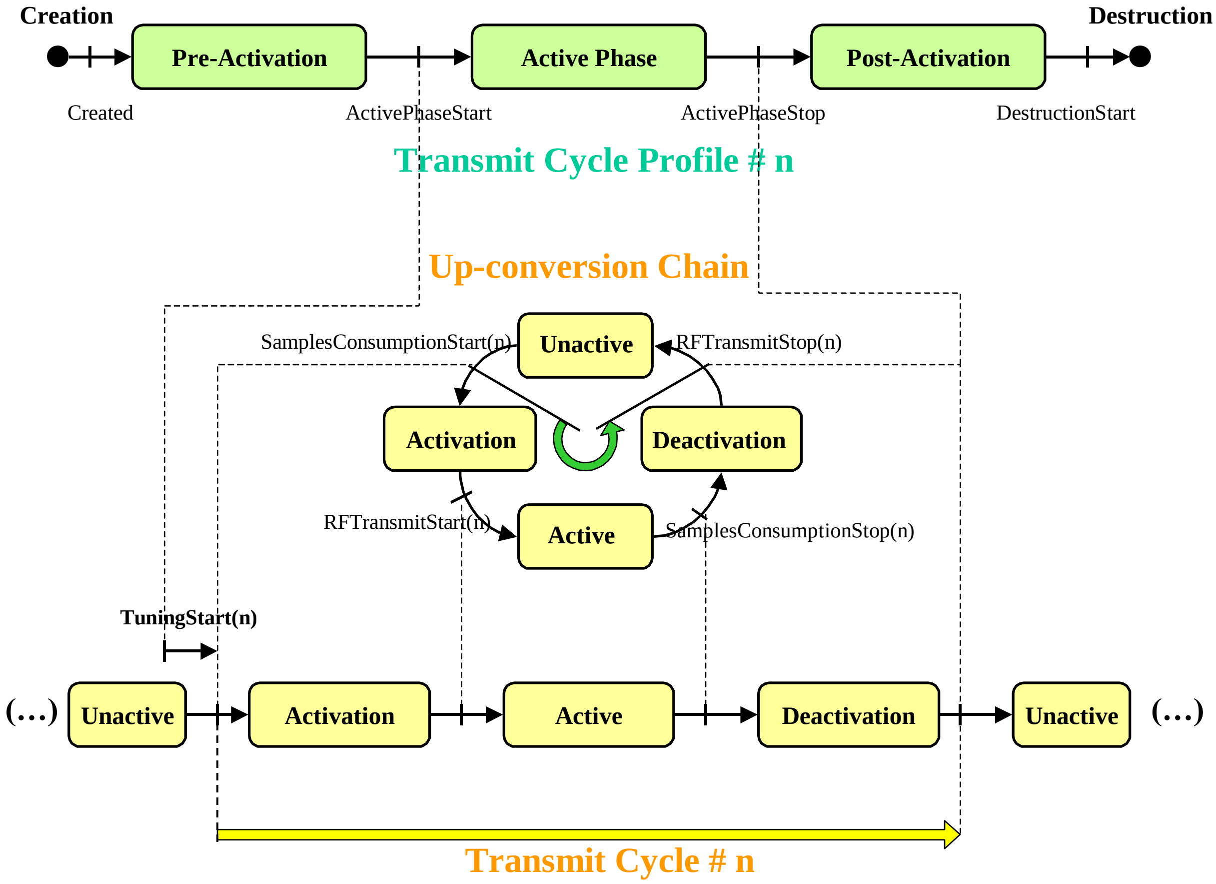


Figure 14: States and Transitions of a Transmit Cycle Profile

FEAT_TX_CHAN_ANA_390 [States of a "Transmit Cycle Profile"

The following states **shall** be implemented by a *Transmit Cycle Profile*:

- *Creation*: The *Transmit Cycle Profile* is created by *Transmit Channel.*
- *Pre-Activation*: The *Transmit Cycle Profile* is kept dormant with the possibility for the Waveform Application to update the profile (*Time Profile* or *Up-conversion Profile*).
- *Active Phase*: The profile values are taken into account by the *Up-conversion Chain* to launch the *Transmit Cycle* of interest, with undefined values completed according to *Tuning*, and updated to reflect what is effectively implemented during the *Transmit Cycle* in case they change because of *Active Tuning.*
- *Post-Activation*: The *Transmit Cycle* specified by the *Transmit Cycle Profile* is over, but profile is kept available to let the *Tuning* of the next *Transmit Cycle* to re-use previous values
- *Destruction*: The *Transmit Cycle Profile* is destroyed by the *Transmit Channel.*

] END_FEAT_TX_CHAN_ANA_390

FEAT_TX_CHAN_ANA_400 [Transitions of "Transmit Cycle Profile"

The instantaneous transitions between states of a *Transmit Cycle Profile* **shall** respect the following list:

- *Created*: transition between states *Creation* and *Pre-Activation*
- *ActivePhaseStart*: transition between states *Pre-Activation* and *Active Phase*
- *ActivePhaseStop*: transition between states *Active Phase* and *Post-Activation*
- *DestructionStart*: transition between states *Post-Activation* and *Desctruction*

] END_FEAT_TX_CHAN_ANA_400

FEAT_TX_CHAN_ANA_410 [Transitions correspondance

The transitions of the *Up-conversion Chain* for the *Active Transmit Cycle # N* **shall** follow the following relationships with the transitions of the corresponding instance of *Transmit Cycle Profile*:

- *ActivePhaseStart* and *TuningStart(n)* simultaneous
- *ActivePhaseStop* and *RFTransmitStop(n)* simultaneous
- *DestructionStart* after than *TransmitStartTime(n+1)*

] END_FEAT_TX_CHAN_ANA_410

3.2.3.6 Active Phase

Triggering conditions

The process leading to the activation of the *Transmit Channel Profile* starts when the *Up-conversion Chain* starts the *Tuning* activity, immediately before state *Activation* starts.

The triggering condition to start *Tuning* activity is based on the definition status of *TransmitStartTime*:

- If *TransmitStartTime* is defined, *TuningStart* will be realized so as to have the resulting *RFTransmitStart* happen at *TransmitStartTime.*
- If *TransmitStartTime* is not defined, *TuningStart* will be realized as soon as the number of baseband samples of the current burst accumulated in *Transmit Baseband Signal FIFO* has reached *TuningStartThreshold*.

If explicit notion *TransmitStopTime* is set to a defined value when *TuningStart* is happening, the *Transmit Cycle* is activated for a pre-defined duration. Otherwise, the cycle ending conditions remain to be defined after the cycle has started.

FEAT_TX_CHAN_ANA_420 [Time-defined transition "StartTuning"

If the *Explicit Notion TransmitStartTime* of the *TimeProfile* is defined, the Transmit Channel **shall** initiate a *TuningStart* transition so as to have the *RFTransmitStart* happen at the instant specified by value of property *TransmitStartTime*.

] END_FEAT_TX_CHAN_ANA_420

FEAT_TX_CHAN_ANA_430 [Signal-defined transition "StartTuning"

If the *Explicit Notion TransmitStartTime* of the *TimeProfile* is not defined, the Transmit Channel **shall** initiate a *TuningStart* transition as soon as the number of baseband samples in *Transmit Baseband Signal FIFO* for the current burst has reached a value superior or equal to the value of *Implicit Notion TuningStartThreshold*.

] END_FEAT_TX_CHAN_ANA_430

FEAT_TX_CHAN_ANA_440 [Error "ERR_TooLateRequest"

In case the cycle specified by argument *targetCycleId* has already reached the state *Tuning*, or went further in the life sequence of the *Transmit Cycle Profile*, the operation can not be taken into account and an error *ERR_TooLateRequest* **shall** be generated.

] END_FEAT_TX_CHAN_ANA_440

Setting applicable profile

FEAT_TX_CHAN_ANA_450 [Precedence rules for Tuning explicit notions definition

For each up-conversion processing explicit notion of any given *Transmit Cycle*, except the first one, the values applicable by the Up-conversion shall be determined during state *Tuning* in compliance with the following precedence rules, sorted in decreasing order:

- Last value set by a call during the state *Idle*
- Value set during the call to *createTransmitCycle()*
- Value applied for the considered property on the previous *Transmit Cycle*, if defined

] END_FEAT_TX_CHAN_ANA_450

3.2.3.7 Post-Activation

Triggering conditions

State *Post-Activation* of the *Transmit Channel Profile* corresponds to when the *Up-conversion Chain* has finished its state *Deactivation* and is back to state *Unactive*.

Entering into state *Post-Activation* is entirely based on the triggering conditions which push the *Up-conversion Chain* into state *Deactivation*.

- If *TransmitStopTime* is defined, *SignalConsumptionStop* will be realized so as to have the resulting *RFTransmitStop* happen at *TransmitStopTime*.
- If *TransmitStopTime* is not defined, availability of the *last burst sample* will trigger transition *SignalConsumptionStop*.

The usage of operation *setTransmitStopTime()* with an undefined *requestedTransmitStopTime* value is another possibility for *SignalConsumptionStop* trigger introduced previously.

FEAT_TX_CHAN_ANA_460 [Time-defined transition "SignalConsumptionStop"

If the *explicit notion TransmitStopTime* of the *TimeProfile* is defined, the *Transmit Channel* **shall** initiate a transition *SignalConsumptionStop* so as to have the *RFTransmitStop* happen at the instant specified by value of *TransmitStopTime*.

] END_FEAT_TX_CHAN_ANA_460

FEAT_TX_CHAN_ANA_470 [Signal-defined transition "SignalConsumptionStop"

If the *explicit notion TransmitStopTime* of the *TimeProfile* is not defined, the *Transmit Channel* **shall** initiate a *SignalConsumptionStop* transition as soon as the last sample of the burst is available within *Tx Baseband Signal FIFO*.

] END_FEAT_TX_CHAN_ANA_470

FEAT_TX_CHAN_ANA_480 [Useless samples discarding

The *Tx Baseband Signal FIFO* **shall** discard the samples transmitted by the *Waveform Application* within the current burst which would exceed the last sample effectively consumed by the *Up-conversion Chain* in respect of the specified *TransmitStopTime*.

] END_FEAT_TX_CHAN_ANA_480

FEAT_TX_CHAN_ANA_490 [Destruction start time

The destruction of a given *Transmit Cycle Profile* **shall** not be realized by the Transmit Channel until the next *Transmit Cycle* has reached its transition *RFTransmitStart*.

] END_FEAT_TX_CHAN_ANA_490

This requirement enables the *Tuning* of the next cycle to be able to re-use the previous *Tuning Profile* for undefined properties.

FEAT_TX_CHAN_ANA_500 [Error "Too Short Burst"

When the last sample of a burst is consumed and consumption shall continue because *ConsumptionStopTime* is not yet reached, the *Transceiver Subsystem* shall generate an error *Too Short Burst*.

] END_FEAT_TX_CHAN_ANA_500

3.2.3.8 Transmit Control

`createTransmitCycleProfile()`

Purpose

The operation *createTransmitCycleProfile()* enables to request the creation by the *Up-conversion Chain* of a specific *Transmit Cycle Profile*. It aims at performing a systematic call for any necessary *Transmit Cycle*.

Syntax

FEAT_TX_CHAN_ANA_510 [Operation "createTransmitCycleProfile()"

The operation *createTransmitCycleProfile()* **shall** be implemented by the Transmit Channel and be used by the Waveform Application in order to request the creation of a particular *Transmit Cycle Profile*.

] END_FEAT_TX_CHAN_ANA_510

FEAT_TX_CHAN_ANA_520 [Argument "requestedTransmitStartTime"

The argument *requestedTransmitStartTime* of type *TimeRequest* **shall** be used to request an initial value for the explicit notion *TransmitStartTime*.

] END_FEAT_TX_CHAN_ANA_520

FEAT_TX_CHAN_ANA_530 [Constraint "MinTransmitStartProximity"

If the argument *requestedTransmitStartTime* is an Event-based Time, it shall respect the *Min Event Proximity* constraint specified by "MinTransmitStartProximity".

] END_FEAT_TX_CHAN_ANA_530

FEAT_TX_CHAN_ANA_540 [Constraint "MinTransmitStartAnticipation"

The constraint *MinTransmitStartAnticipation* of type *Latency* **shall** be used to characterize the minimum allowed elapsed time between the instant the *createTransmitCycleProfile()* operation is invoked and the target Transmission start time.

] END_FEAT_TX_CHAN_ANA_540

FEAT_TX_CHAN_ANA_550 [Argument "requestedTransmitStopTime"

The argument *requestedTransmitStopTime* of type *TimeRequest* **shall** be used to request an initial value for the explicit notion *TransmitStopTime*.

] END_FEAT_TX_CHAN_ANA_550

FEAT_TX_CHAN_ANA_560 [Argument “requestedTuningPresetId”

The argument *requestedTuningPresetId* of type *UShort* **shall** be used in order to indicate the *PresetId* value of the *Tuning Preset* to be applied on the considered burst.

] END_FEAT_TX_CHAN_ANA_560

The requirements associated with *Tuning Presets* are provided in *Common Concepts*.

FEAT_TX_CHAN_ANA_570 [Argument “requestedCarrierFrequency”

The argument *requestedCarrierFrequency* of type *Frequency* **shall** be used to request a value for tuning explicit notion *Carrier Frequency*.

] END_FEAT_TX_CHAN_ANA_570

FEAT_TX_CHAN_ANA_580 [Argument “requestedNominalRFPower”

The argument *requestedNominalRFPower* of type *Frequency* **shall** be used to request a value for tuning explicit notion *Nominal RF Power*.

] END_FEAT_TX_CHAN_ANA_580

Semantics

FEAT_TX_CHAN_ANA_590 [TransmitCycleProfile initialization

The Transmit Channel **shall** set the initial values of the corresponding *explicit notions* in *TransmitCycleProfile* to the values specified by passed arguments.

] END_FEAT_TX_CHAN_ANA_590

If argument *requestedTransmitStartTime* is defined by the Waveform Application, the Transmit Channel will start the Active Phase according to the speficied time. If the initial value of *TransmitStartTime* was left undefined, the Active Phase may not start until the value is further set.

FEAT_TX_CHAN_ANA_600 [Constraint “MinTransmitStartAnticipation”

If argument *requestedTransmitStartTime* is set to a defined value by the Waveform Application, the operation invocation which corresponds with internal notion TransmitStartAnticipation **shall** happen before the corresponding *TuningStartTime* respecting the MinTransmitStartAnticipation real-time constraint.

] END_FEAT_TX_CHAN_ANA_600

FEAT_TX_CHAN_ANA_610 [Error “ERR_TooManyCreatedTxProfiles”

The Transmit Channel **shall** generate an error *ERR_TooManyCreatedTxProfiles* if creation of the *Transmit Cycle Profile* would cause a number of created profiles exceeding the value of constraint *MaxTxCycleProfiles*.

] END_FEAT_TX_CHAN_ANA_610

FEAT_TX_CHAN_ANA_620 [“requestedTransmitStartTime” using Absolute Time Request format

Usage of domain type *Absolute Time Request format* **shall** be available as a type possibility for argument *requestedTransmitStartTime*.

] END_FEAT_TX_CHAN_ANA_620

FEAT_TX_CHAN_ANA_630 [“requestedTransmitStartTime” using event-based format

Usage of domain type “Event-based Time Request”**shall** be available as a type possibility for argument *requestedTransmitStartTime*.

] END_FEAT_TX_CHAN_ANA_630

FEAT_TX_CHAN_ANA_640 [Default event source for event-based time requests

The default *Event Source* for event-derived expression of *requestedTransmitStartTime* **shall** be the *TransmitStartTime* of the current Transmit Cycle.

] END_FEAT_TX_CHAN_ANA_640

For the first *Transmit Cycle* of the *Transmit Channel* lifetime, since there is no available pre-existing *Transmit Cycle*, the *current* Transmit Cycle is not defined so event-based approach can not be used.

FEAT_TX_CHAN_ANA_650 [Default event occurrence for event-derived time requests

The default *event occurrence* for event-derived expression of *requestedTransmitStartTime* **shall** be the *last occurence*.

] END_FEAT_TX_CHAN_ANA_650

configureTransmitCycle()

Purpose

This operation is the way a Waveform Application can set the Up-conversion profile during its state *Pre-activation*.

Syntax

FEAT_TX_CHAN_ANA_660 [Operation “configureTransmitCycle()”

The operation *configureTransmitCycle()* **shall** be implemented by the Transmit Channel and be used by the Waveform Application in order to set the value of the properties of a previously created *Transmit Cycle*.

] END_FEAT_TX_CHAN_ANA_660

FEAT_TX_CHAN_ANA_670 [Target cycle identification in “configureTransmitCycle()”

The *Transmit Cycle* for which the requested properties values are applicable **shall** be identified by the argument *targetCycleId* of type *ULong*, which specifies the *Cycle Identifier* of the target Transmit Cycle.

] END_FEAT_TX_CHAN_ANA_670

FEAT_TX_CHAN_ANA_680 [Argument “requestedTransmitStartTime”

The argument *requestedTransmitStartTime* of type *TimeRequest* **shall** be used to request an initial value for the *explicit notion TransmitStartTime*.

] END_FEAT_TX_CHAN_ANA_680

FEAT_TX_CHAN_ANA_690 [Constraint “MinTransmitStartAnticipation”

The constraint *MinTransmitStartAnticipation* of type *Latency* **shall** be used to characterize the minimum allowed elapsed time between the instant *configureTransmitCycle()* operation are invoked and the target transmission start time, when the argument *requestedTransmitStartTime* is specified.

] END_FEAT_TX_CHAN_ANA_690

FEAT_TX_CHAN_ANA_700 [Argument “requestedTransmitStopTime”

The argument *requestedTransmitStopTime* of type *TimeRequest* **shall** be used to request an initial value for the explicit notion *TransmitStopTime*.

] END_FEAT_TX_CHAN_ANA_700

FEAT_TX_CHAN_ANA_710 [Argument “requestedCarrierFrequency”

The argument *requestedCarrierFrequency* of type *Frequency* **shall** be used to request a value for tuning explicit notion *Carrier Frequency*.

] END_FEAT_TX_CHAN_ANA_710

FEAT_TX_CHAN_ANA_720 [Argument “requestedNominalRFPower”

The argument *requestedNominalRFPower* of type *Frequency* **shall** be used to request a value for tuning explicit notion *Nominal RF Power*.

] END_FEAT_TX_CHAN_ANA_720

Errors

FEAT_TX_CHAN_ANA_730 [Error “Unkown Target Cycle”

In case the cycle specified by argument *targetCycleId* is not created, the operation can not be taken into account and an error *Unknown Target Cycle* **shall** be generated.

] END_FEAT_TX_CHAN_ANA_730

`setTransmitStopTime()`

Purpose

The operation *setTransmitStopTime()* is intended to allow setting the *explicit notion TransmitStopTime value* for a *Transmit Cycle.*

FEAT_TX_CHAN_ANA_740 [Operation “setTransmitStopTime()”

The operation *setTransmitStopTime()* **shall** be implemented by the *Transmit Channel* and be used by the Waveform Application in order to set the *TransmitStopTime* of the considered *Transmit Cycle*.

] END_FEAT_TX_CHAN_ANA_740

FEAT_TX_CHAN_ANA_750 [Argument “requestedTransmitStopTime”

The argument *requestedTransmitStopTime* **shall** be available in operation *setTransmitStopTime()* in order to enable the Waveform Application to specify when the transition *RFTransmitStop* shall happen.

] END_FEAT_TX_CHAN_ANA_750

FEAT_TX_CHAN_ANA_760 [Defined “requestedTransmitStopTime”

If argument *requestedTransmitStartTime* is set to a defined value by Waveform Application, the Transmit Channel **shall** deactivate the *Transmit Cycle* in respect of the specified value.

] END_FEAT_TX_CHAN_ANA_760

FEAT_TX_CHAN_ANA_770 [Undefined “requestedTransmitStopTime” (instant Deactivation)

If argument *requestedTransmitStartTime* is set to *undefined* by Waveform Application, the Transmit Channel **shall** immediately turn the current *Transmit Cycle* into *Deactivation* state, discarding the unused baseband samples eventually stored into *Tx Baseband Signal FIFO* for the aborted burst.

] END_FEAT_TX_CHAN_ANA_770

FEAT_TX_CHAN_ANA_780 [Constraint “MinTransmitStopAnticipation”

The operation *setTransmitStopTime()* **shall** be called with an anticipation relatively to the *requestedTransmitStopTime* at least equal to value of constraint *MinTransmitStopAnticipation*, of type *Latency*.

] END_FEAT_TX_CHAN_ANA_780

The operation *setTransmitStopTime()* may be used to modify as often as necessary the *TransmitStopTime* of a *Transmit Cycle*, provided it always respects *TransmitStopAnticipation* invocation margin. Its actual invocation time corresponds with internal notion *TransmitStartAnticipation.*

Errors

FEAT_TX_CHAN_ANA_790 [Error “ERR_TooLateRequest”

If the operation *setTransmitStopTime()* is invoked after *TransmitStopTime – MinTransmitStopAnticipation*, the Transmit Channel is not able to guarantee correct application, and an error *ERR_TooLateRequest* **shall** be generated.

] END_FEAT_TX_CHAN_ANA_790

Interface real-time constraints

FEAT_TX_CHAN_ANA_800 [Constraint “MaxTransmitControlInvocationDuration”

Operations of the TransmitControl interface **shall** respect the *MaxInvocationDuration* constraint specified by *MaxTransmitControlInvocationDuration*.

] END_FEAT_TX_CHAN_ANA_800

3.2.3.9 Transmit Data Push

Overview

The programming interface *Transmit Data Push* enables the *Waveform Application* to push packets of baseband samples towards the *Transmit Channel*.

The programming interface relies on: (i) a single operation, *pushBBSamplesTx()*, (ii) the data pushed by the *Waveform Application*, denoted as *The Pushed Packet*.

pushBBSamplesTx() is used by the *Waveform Application* to: (i) notify the *Transmit Channel* of availability of a new packet of samples, (ii) indicate if the pushed packet is the last packet of the transmitted burst.

The Pushed Packet structure is defined by common concept *Baseband Packet*. It must be prepared by *Waveform Application* prior to the *pushBBSamplesTx()* notification. The samples are stored by the *Waveform Application* in conformance with specific *packets contents requirements*.

Once *pushBBSamplesTx()* is called, the *Transmit Channel* undertakes *packets handling* activities. First, it has to *take in charge* the *pushed packet*, copying its content into an internal memory zone. Second, it has to *store* the corresponding data into the *Baseband FIFO*.

The way a *Transmit Channel* implementation takes in charge the *pushed packet* and stores it into the *Baseband FIFO* is implementation dependent. Most reactive implementations will take in charge and store in baseband FIFO the pushed packet into one single data copy. More complex designs may exist, especially when the digital part of the *Transceiver Subsystem* is distributed over different digital signal processing units.

The Pushed Packet

Structural requirements

FEAT_TX_CHAN_ANA_810 [Definition of “The Pushed Packet”

A data constuct denoted *The Pushed Packet*, of type *Baseband Packet*, **shall** be used to contain the Baseband Signal samples to be pushed.

] END_FEAT_TX_CHAN_ANA_810

FEAT_TX_CHAN_ANA_820 [Creation of “The Pushed Packet”

The Pushed Packet **shall** be created by the *Waveform Application*.

] END_FEAT_TX_CHAN_ANA_820

FEAT_TX_CHAN_ANA_830[Visibility on “The Pushed Packet”

The Pushed Packet **shall** be visible by the *Waveform Application* with *write* rights and be visible by the *Transmit Channel* with *read* rights.

] END_FEAT_TX_CHAN_ANA_830

FEAT_TX_CHAN_ANA_840[Destruction of “The Pushed Packet”

The Pushed Packet **shall** be destroyed by the *Waveform Application*.

] END_FEAT_TX_CHAN_ANA_840

Packets ordering requirements

This section specifies a certain number of requirements applicable to packets content.

FEAT_TX_CHAN_ANA_850 [Packets burst alignment

The *pushed* baseband samples **shall** be aligned with the *Transmit Cycles Input Bursts*, which means no packet contains samples belonging to two different input bursts.

] END_FEAT_TX_CHAN_ANA_850

FEAT_TX_CHAN_ANA_860 [Sequential bursts transmission

The *pushed* baseband samples packets **shall** be transmitted one burst after the other.

] END_FEAT_TX_CHAN_ANA_860

FEAT_TX_CHAN_ANA_870 [Ordered packets pushes

The *pushed* baseband samples packets **shall** not be interverted within a given burst.

] END_FEAT_TX_CHAN_ANA_870

Packets identification

The *first packet* of a *burst* is defined as the packet where *first sample* is the *first transmitted sample* of the *burst*.

The last packet of a *burst* is defined as the packet where *last sample* is the *last transmitted sample* of the *burst*.

A packet containing **all** the samples of a given burst is at the same time the *first* and *last* packet of the burst.

The *last packet* of a burst is therefore systematically followed by the *first packet* of the next burst.

`pushBBSamplesTx()`

Syntax

FEAT_TX_CHAN_ANA_880 [Operation "pushBBSamplesTx()"

Operation *pushBBSamplesTx()* **shall** be used by the *Waveform Application* and realized by the *Transmit Channel*, in order to: (i) notify the *Transmit Channel* of availability of a new packet of samples, (ii) indicate if the pushed packet is the last packet of the transmitted burst.

] END_FEAT_TX_CHAN_ANA_880

FEAT_TX_CHAN_ANA_890 [Argument "thePushedPacket"

The argument *thePushedPacket*, of type *BBSamplesPacket*, **shall** contain information enabling to access to the pushed packet.

] END_FEAT_TX_CHAN_ANA_890

The pushed packet content shall have been prepared by the Waveform Application to contain correct signal before invocation. The size of the pushed packet is determined by the value of argument *requestedPacketSize* set during the Transmit Cycle creation or profile configuration.

FEAT_TX_CHAN_ANA_900 [Argument "endOfBurst"

The argument *endOfBusrt*, of type *Boolean*, **shall** be used to indicate that the pushed packet is the last packet of the current *Transmit Cycle Input Burst*.

Value *True* if the packet is the last packet, *False* or *undefined* otherwise.

] END_FEAT_TX_CHAN_ANA_900

Semantics

FEAT_TX_CHAN_ANA_910 [pushBBSamplesTx() Post-Invocation

When *return* of *pushBBSamplesTx()* returns, the samples values contained in the pushed packet **shall** have been taken into account by the *Transceiver Subsystem*.

] END_FEAT_TX_CHAN_ANA_910

The *Waveform Application* is then free to handle in any way the *pushed packet*, since return from invocation means the useful data hase been taken into account by *Transmit Channel*. Among possibilities, one can quote that the same *pushed packet* can be re-used for the next push, that it can be destroyed with another one being created for the next push, that a pair of packets may be alternately used in flip-flop.

FEAT_TX_CHAN_ANA_920 [pushBBSamplesTx() Post-Invocation for defined *TransmitStopTime*

When the *TransmitStopTime* of a burst is explicitly set, the *Transmit Channel* **shall** discard any pushed samples in excess from the Baseband Burst limit set by *TransmitStopTime* value.

] END_FEAT_TX_CHAN_ANA_920

Realt-time Constraints

FEAT_TX_CHAN_ANA_930 [Constraint "MaxTransmitDataPushInvocationDuration"

The constraint *"MaxTransmitDataPushInvocationDuration",* of type *Latency*, **shall** be used to identify the *Max Invocation Duration* to be met by *pushBBSamplesTx* implementations.

] END_FEAT_TX_CHAN_ANA_930

As noted in §2.3,the *Max Invocation Duration* of an operation corresponds to the time difference between:

- The instant when an operation *invocation* occurs
- The instant when an operation *return* occurs

The constraint may remain *undefined* if the real-time constraints of the radio capability do not justify it.

Max pushBBSamplesTx Invocation Duration is generally assigned a unique value, taking into consideration the maximum packet size used by the waveform application.

FEAT_TX_CHAN_ANA_940 [Constraint "Min Packet Storage Anticipation"

The constraint *Min Packet Storage Anticipation*, of type *Latency*, **shall** be used to identify the *Minimum Anticipation* to be guaranteed in order to have a packet stored in *Baseband FIFO* before the target time.

] END_FEAT_TX_CHAN_ANA_940

The target time to be considered takes into account the consumption start time of the considered burst and the relative time shift between the first sample of the burst and the samples in the pushed packet.

Arguments Boundaries

FEAT_TX_CHAN_ANA_950 [Implicit Notion "MaxPushedPacketSize"

The implicit notion *MaxPushedPacketSize*, of type *Ulong*, **shall** be used to set the maximum packet size sent through the operation *pushBBSamplesPacket()*.

] END_FEAT_TX_CHAN_ANA_950

FEAT_TX_CHAN_ANA_960 [Error "ERR_Oversized Pushed Packet"

In case the size of a *pushed packet* exceeds the property *MaxPushedPacket Size*, the Transmit Channel **shall** generate the error *ERR_Oversized Pushed Packet*.

] END_FEAT_TX_CHAN_ANA_960

Overflow mitigation

Two overflow mitigation options can be selected:

- Caller Blocked: The *Transceiver Subsystem* will block the waveform application until there is sufficient space in the FIFO to copy the pushed packet.
- No Overflow: Reaching overflow is corresponded to an error, so the *Transceiver Subsystem* is requested to generate an error.

FEAT_TX_CHAN_ANA_970 [Implicit notion "OverflowMitigation"

The enumerated implicit notion *Overflow Mitigation* **shall** be used to set the applicable overflow mitigation option.

The possible values are: (i)*Caller Blocked* and (ii) *No Overflow*. Undefined corresponds to *Caller Blocked*.

] END_FEAT_TX_CHAN_ANA_970

FEAT_TX_CHAN_ANA_980 [Caller Blocked Overflow Mitigation

When implicit notion *Overflow Mitigation* is set to *CallerBlocked*, the Transceiver Subsystem **shall** block the *Waveform Application* execution until sufficient room is available within *Baseband FIFO* to complete copy of the *pushed packet*. The constraint *Max Invocation Duration* is not applicable in this scenario.

] END_FEAT_TX_CHAN_ANA_980

This is the default mode since it does not require any supplementary dependency between the Waveform Application and the platform. This occurs at the expense of handling trivial synchronization mechanisms. It is therefore believed to be the optimal solution.

FEAT_TX_CHAN_ANA_990 [No Overflow Overflow Mitigation

When implicit notion *Overflow Mitigation* is set to *No Overflow*, the Transceiver Subsystem **shall** raise an error in case an overflow of the FIFO occurs. The samples in excess are not copied into *baseband FIFO*.

] END_FEAT_TX_CHAN_ANA_990

This mode can be used in cases where waveform design is considering overflow case shall not occur. Two mainstream design paradigms can justify usage of such an option:

- Synchronization events provide the *Waveform Application* with information which shall avoid overflow situation.
- Flow control mechanisms enable the *Transmit Channel* to stop the *Waveform Application* when the *Tx Baseband Signal FIFO* fill level becomes critical, and to restart it once room is available again. This corresponds to feature *AsyncControlMechanisms* of [Digital IF Initial Submission].

3.2.3.10 Analysis Requirements Summary

The following table provides an overview of the Analysis requirements of *Transmit Channel*:

Identifier	Tag	Section
FEAT_TX_CHAN_ANA_010	Definition of a “Transmit Channel”	Transmit Channel
FEAT_TX_CHAN_ANA_020	Creation of a “Transmit Channel”	Transmit Channel
FEAT_TX_CHAN_ANA_030	Composition of a “Transmit Channel”	Transmit Channel
FEAT_TX_CHAN_ANA_040	Definition of “Transmit Baseband Signal”	Transmit Baseband Signal
FEAT_TX_CHAN_ANA_050	Implicit Notions of “Transmit Baseband Signal”	Transmit Baseband Signal
FEAT_TX_CHAN_ANA_060	Transmit Baseband Signal configuration	Transmit Baseband Signal
FEAT_TX_CHAN_ANA_070	Definition of “Transmit Baseband FIFO”	Transmit Baseband FIFO
FEAT_TX_CHAN_ANA_080	Implicit Notion “BasebandFIFOSize”	Transmit Baseband FIFO
FEAT_TX_CHAN_ANA_090	Implicit Notion “TuningStartThreshold”	Transmit Baseband FIFO
FEAT_TX_CHAN_ANA_100	Default value of “TuningStartThreshold”	Transmit Baseband FIFO
FEAT_TX_CHAN_ANA_110	Configuration of “Transmit Baseband FIFO”	Transmit Baseband FIFO
FEAT_TX_CHAN_ANA_120	Error “FIFOUnderflow”	Transmit Baseband FIFO
FEAT_TX_CHAN_ANA_130	FIFO Underflow TransmitStop	Transmit Baseband FIFO
FEAT_TX_CHAN_ANA_140	Definition of the “Up-conversion Chain”	Up-conversion Chain
FEAT_TX_CHAN_ANA_150	Constraint “MaxUpconversionLatency”	Up-conversion Chain
FEAT_TX_CHAN_ANA_160	States of Up-convestion Chain	Up-conversion Chain
FEAT_TX_CHAN_ANA_170	Transitions between “Up-conversion Chain” states	Up-conversion Chain
FEAT_TX_CHAN_ANA_180	Timings associated to Up-convestion Chain transitions	Up-conversion Chain
FEAT_TX_CHAN_ANA_190	Tuning activity	Up-conversion Chain
FEAT_TX_CHAN_ANA_200	Constraint “MaxTuningDuration”	Up-conversion Chain
FEAT_TX_CHAN_ANA_210	Internal Notion “TuningStartTime”	Up-conversion Chain
FEAT_TX_CHAN_ANA_220	Constraint “MinReactivationTime”	Up-conversion Chain
FEAT_TX_CHAN_ANA_230	Definition of the “Current” Transmit Cycle	Up-conversion Chain
FEAT_TX_CHAN_ANA_240	Rule for “Current” Transmit Cycle identification	Up-conversion Chain
FEAT_TX_CHAN_ANA_250	Definition of “Transmit Cycle Input Burst”	Transmit Cycle
FEAT_TX_CHAN_ANA_260	Definition of “Transmit Cycle First Sample”	Transmit Cycle
FEAT_TX_CHAN_ANA_270	Definition of “Transmit Cycle Last Sample”	Transmit Cycle
FEAT_TX_CHAN_ANA_280	Definition of “Transmit Cycle Profile”	Transmit Cycle
FEAT_TX_CHAN_ANA_290	Composition of “Transmit Cycle Profile”	Transmit Cycle
FEAT_TX_CHAN_ANA_300	Explicit Notion “TransmitCycle”	Transmit Cycle
FEAT_TX_CHAN_ANA_310	Constraint “MaxTxCycleProfiles”	Transmit Cycle
FEAT_TX_CHAN_ANA_320	Definition of “Time Profile”	Transmit Cycle
FEAT_TX_CHAN_ANA_330	Constraint « TransmitTimeProfileAccuracy »	Transmit Cycle
FEAT_TX_CHAN_ANA_340	Explicit Notion “TransmitStartTime”	Transmit Cycle
FEAT_TX_CHAN_ANA_350	Referenced event source « TransmitStart »	Transmit Cycle
FEAT_TX_CHAN_ANA_360	Explicit Notion “TransmitStopTime”	Transmit Cycle
FEAT_TX_CHAN_ANA_370	Definition of “Transmit Tuning Profile”	Transmit Cycle
FEAT_TX_CHAN_ANA_380	Contents of “Tuning Profile”	Transmit Cycle
FEAT_TX_CHAN_ANA_390	States of “Transmit Cycle Profile”	Transmit Cycle
FEAT_TX_CHAN_ANA_400	Transitions of “Transmit Cycle Profile”	Transmit Cycle
FEAT_TX_CHAN_ANA_410	Transitions correspondence	Transmit Cycle
FEAT_TX_CHAN_ANA_420	Time-defined transition “StartTuning”	Active Phase
FEAT_TX_CHAN_ANA_430	Signal-defined transition “StartTuning”	Active Phase
FEAT_TX_CHAN_ANA_440	Error “ERR_TooLateRequest”	Active Phase
FEAT_TX_CHAN_ANA_450	Precedence rules for Tuning explicit notions definition	Active Phase
FEAT_TX_CHAN_ANA_460	Time-defined transition “SignalConsumptionStop”	Post-Activation
FEAT_TX_CHAN_ANA_470	Signal-defined transition “SignalConsumptionStop”	Post-Activation
FEAT_TX_CHAN_ANA_480	Useless samples discarding	Post-Activation
FEAT_TX_CHAN_ANA_490	Desctruction start time	Post-Activation
FEAT_TX_CHAN_ANA_500	Error TooShortBurst	Post-Activation
FEAT_TX_CHAN_ANA_510	Operation “createTransmitCycleProfile()”	Transmit Control
FEAT_TX_CHAN_ANA_520	Argument “requestedTransmitStartTime”	Transmit Control

FEAT_TX_CHAN_ANA_530	Constraint “MinTransmitStartProximity”	Transmit Control
FEAT_TX_CHAN_ANA_540	Constraint “MinTransmitStartAnticipation”	Transmit Control
FEAT_TX_CHAN_ANA_550	Argument “requestedTransmitStopTime”	Transmit Control
FEAT_TX_CHAN_ANA_560	Argument “requestedTuningPresetId”	Transmit Control
FEAT_TX_CHAN_ANA_570	Argument “requestedCarrierFrequency”	Transmit Control
FEAT_TX_CHAN_ANA_580	Argument “requestedNominalRFPower”	Transmit Control
FEAT_TX_CHAN_ANA_590	TransmitCycleProfile initialization	Transmit Control
FEAT_TX_CHAN_ANA_600	Constraint “MinTransmitStartAnticipation”	Transmit Control
FEAT_TX_CHAN_ANA_610	Error “ERR_TooManyCreatedTxProfiles”	Transmit Control
FEAT_TX_CHAN_ANA_620	“requestedTransmitStartTime” using Absolute TimeRequest format	Transmit Control
FEAT_TX_CHAN_ANA_630	“requestedTransmitStartTime” using event-based format	Transmit Control
FEAT_TX_CHAN_ANA_640	Default event source for event-derived time requests	Transmit Control
FEAT_TX_CHAN_ANA_650	Default event occurrence for event-derived time requests	Transmit Control
FEAT_TX_CHAN_ANA_660	Operation “configureTransmitCycle()”	Transmit Control
FEAT_TX_CHAN_ANA_670	Target cycle identification in “configureTransmitCycle()”	Transmit Control
FEAT_TX_CHAN_ANA_680	Argument “requestedTransmitStartTime”	Transmit Control
FEAT_TX_CHAN_ANA_690	Constraint “MinTransmitStartAnticipation”	Transmit Control
FEAT_TX_CHAN_ANA_700	Argument “requestedTransmitStopTime”	Transmit Control
FEAT_TX_CHAN_ANA_710	Argument “requestedCarrierFrequency”	Transmit Control
FEAT_TX_CHAN_ANA_720	Argument “requestedNominalRFPower”	Transmit Control
FEAT_TX_CHAN_ANA_730	Error “Unkown Target Cycle”	Transmit Control
FEAT_TX_CHAN_ANA_740	Operation “setTransmitStopTime()”	Transmit Control
FEAT_TX_CHAN_ANA_750	Argument “requestedTransmitStopTime”	Transmit Control
FEAT_TX_CHAN_ANA_760	Defined “requestedTransmitStopTime”	Transmit Control
FEAT_TX_CHAN_ANA_770	Undefined “requestedTransmitStopTime” (instant Deactivation)	Transmit Control
FEAT_TX_CHAN_ANA_780	Constraint “TransmitStopAnticipation”	Transmit Control
FEAT_TX_CHAN_ANA_790	Error “ERR_TooLateRequest”	Transmit Control
FEAT_TX_CHAN_ANA_800	Constraint “MaxTransmitControlInvocationDuration”	Transmit Control
FEAT_TX_CHAN_ANA_810	Definition of “The Pushed Packet”	Transmit Data Push
FEAT_TX_CHAN_ANA_820	Creation of “The Pushed Packet”	Transmit Data Push
FEAT_TX_CHAN_ANA_830	Visibility on “The Pushed Packet”	Transmit Data Push
FEAT_TX_CHAN_ANA_840	Desctruction of “The Pushed Packet”	Transmit Data Push
FEAT_TX_CHAN_ANA_850	Packets burst alignement	Transmit Data Push
FEAT_TX_CHAN_ANA_860	Sequential bursts transmission	Transmit Data Push
FEAT_TX_CHAN_ANA_870	Ordered packets pushes	Transmit Data Push
FEAT_TX_CHAN_ANA_880	Operation “pushBBSamplesTx()”	Transmit Data Push
FEAT_TX_CHAN_ANA_890	Argument “thePushedPacket”	Transmit Data Push
FEAT_TX_CHAN_ANA_900	Argument “endOfBurst”	Transmit Data Push
FEAT_TX_CHAN_ANA_910	pushBBSamplesTx() Post-Invocation	Transmit Data Push
FEAT_TX_CHAN_ANA_920	pushBBSamplesTx() Post-Invocation for defined *TransmitStopTime*	Transmit Data Push
FEAT_TX_CHAN_ANA_930	Constraint “MaxTransmitDataPushInvocationDuration”	Transmit Data Push
FEAT_TX_CHAN_ANA_940	Constraint “MinPacketStorageAnticipation”	Transmit Data Push
FEAT_TX_CHAN_ANA_950	Implicit Notion “MaxPushedPacketSize”	Transmit Data Push
FEAT_TX_CHAN_ANA_960	Error “ERR_Oversized Pushed Packet”	Transmit Data Push
FEAT_TX_CHAN_ANA_970	Implicit notion “Overflow Mitigation”	Transmit Data Push
FEAT_TX_CHAN_ANA_980	Caller Blocked Overflow Mitigation	Transmit Data Push
FEAT_TX_CHAN_ANA_990	No Overflow Overflow Mitigation	Transmit Data Push

Table 14: Overview of Transmit Channel Analysis requirements

3.2.4 Modelling Requirements

3.2.4.1 Transmit Channel interfaces

The following class diagram involves the interfaces realized by the Transmit Channel, between the Waveform Application and the Transceiver Subsystem:

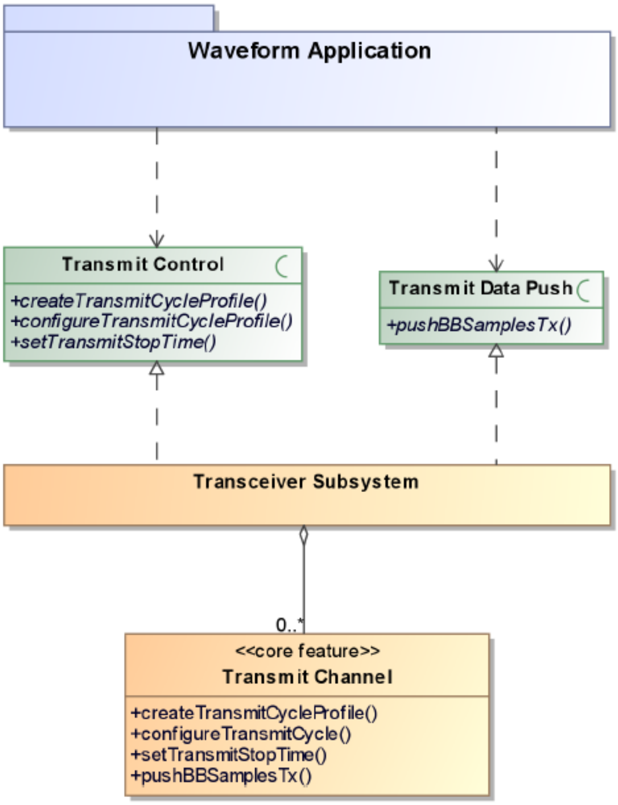


Figure 15: Transmit Channel involved interfaces

FEAT_TX_CHAN_MOD_010 [Transmit Control interface

Transmit Control is the interface used to control the Transmit Channel behaviors according to the Waveform needs. This interface **shall** include the following operations:

- createTransmitCycleProfile()
- configureTransmitCycle()
- setTransmitStopTime()

Definitions of operations and behaviors of their realizations by the Transmit Channel core feature are specified in the baseline requirements of the Transmit Channel.

] END_FEAT_TX_CHAN_MOD_010

FEAT_TX_CHAN_MOD_020 [Transmit Data Push interface

Transmit Data Push is the interface used for transferring baseband samples from the Waveform Application to the Transceiver Subsystem. This interface **shall** include the following operation:

- pushBBSamplesTx()

Definition of this operation and associated behaviour of its realization by the Transmit Channel core feature are specified in the baseline requirements of the Transmit Channel.

] END_FEAT_TX_CHAN_MOD_020

Note: Signatures of the operations are not provided for better visibility. Specification of the operations is depicted in paragraph Baseline Requirements of this chapter.

3.2.4.2 Transmit Channel composition

FEAT_TX_CHAN_MOD_030 [Transmit Channel modelling elements 1

Transceiver Subsystem model involving feature *Transmit Channel* **shall** use the modelling elements depicted in the following class diagram:

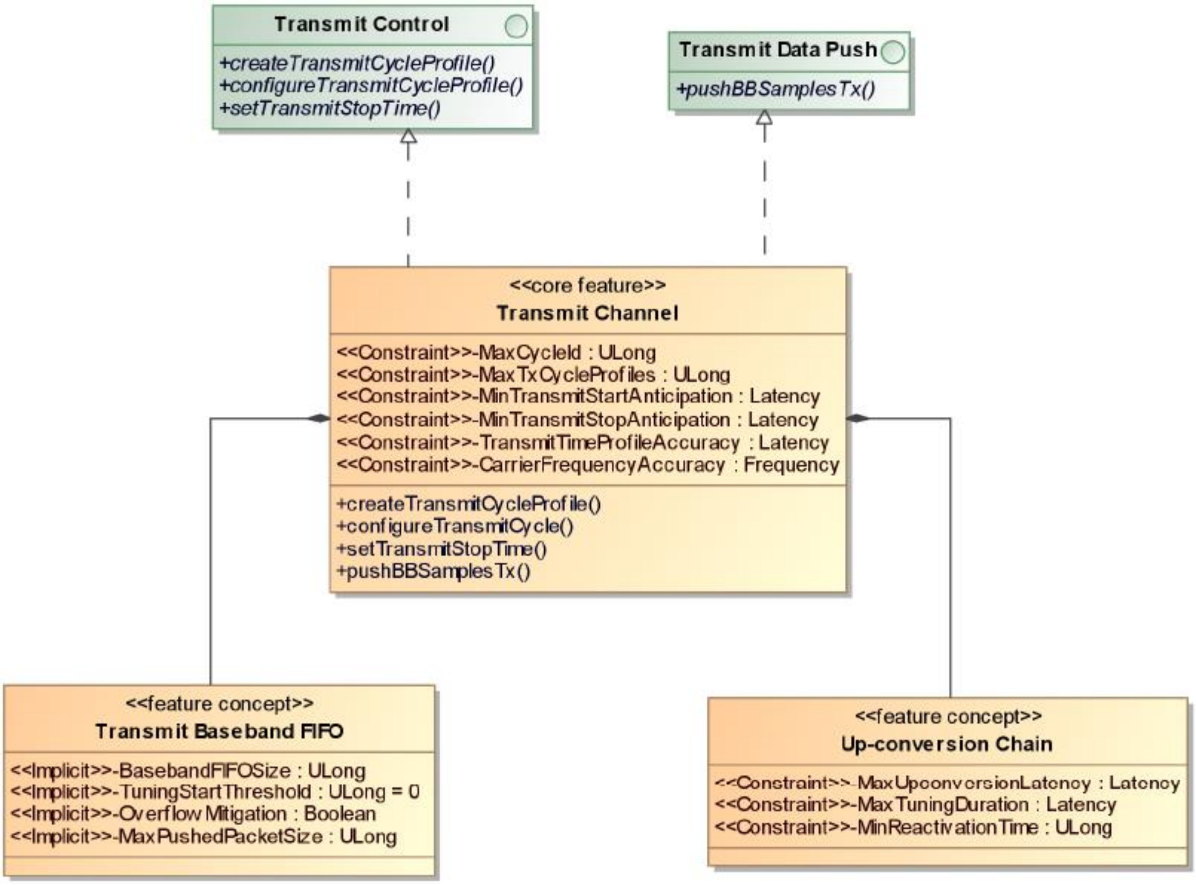


Figure 16: Transmit Channel main composition

] END_FEAT_TX_CHAN_MOD_030

FEAT_TX_CHAN_MOD_040 [Transmit Channel modelling elements 2

Transceiver Subsystem model involving feature *Transmit Channel* **shall** use the modelling elements depicted in the following class diagram :

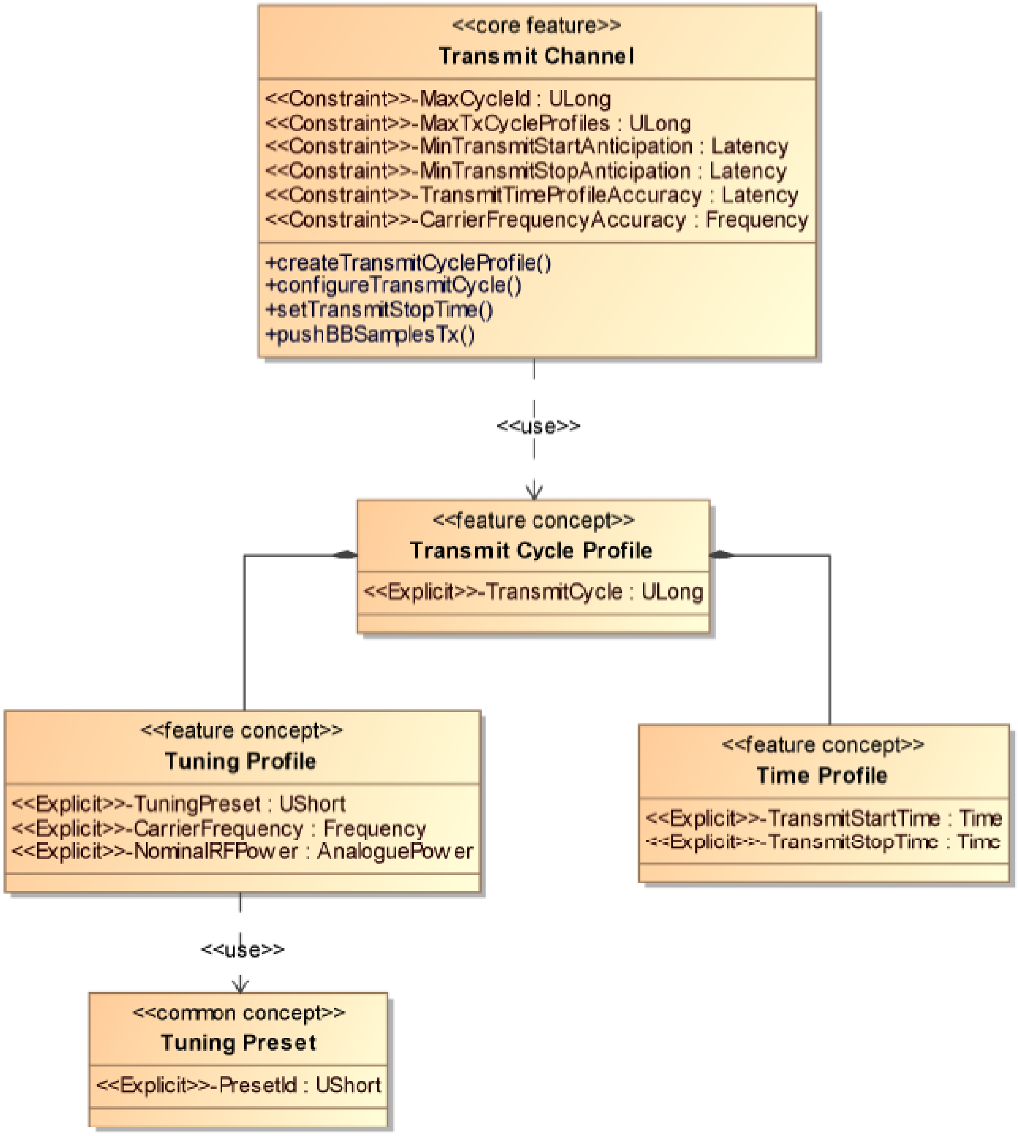


Figure 17: Transmit Channel and Cycle Profile

] END_FEAT_TX_CHAN_MOD_040

3.2.5 Implementation Languages Requirements

3.2.5.1 C++ Requirements

Interfaces between the *Waveform Application* and the *Transceiver Subsystem* for the *Transmit Channel* core feature are:

- *Transmit Control*
- *Transmit Data Push*

Both are seen in a modeling analysis as abstract classes defining abstract operations. In C++ language, this leads to consider these operations as pure virtual methods of its respective classes. These methods can not be implemented by these abstract classes, but shall be overridden and implemented by derived classes.

TransmitControl interface

FEAT_TX_CHAN_CPP_010 [TransmitControl C++ definition

Implementations in C++ using the feature *Transmit Channel* **shall** use the interface *TransmitControl* as declared in the following C++ reference source code extract:

```
class I_TransmitControl
{
public :
      virtual void createTransmitCycleProfile(
                  Time                requestedTransmitStartTime,
                  Time                requestedTransmitStopTime,
                  UShort              requestedPresetId,
                  Frequency           requestedCarrierFrequency,
                  AnaloguePower       requestedNominalRFPower) = 0;

      virtual void configureTransmitCycle(
                  ULong               targetCycleId,
                  Time                requestedTransmitStartTime,
                  Time                requestedTransmitStopTime,
                  Frequency           requestedCarrierFrequency,
                  AnaloguePower       requestedNominalRFPower) = 0;

      virtual void setTransmitStopTime(
                  ULong               targetCycleId,
                  Time                requestedTransmitStopTime) = 0;
};
```

] END_FEAT_TX_CHAN_CPP_010

FEAT_TX_CHAN_CPP_020 [TransmitControl C++ header file

Implementations in C++ using the feature *Transmit Channel* **shall** include the header file entitled *TransmitControl.h*, corresponding to the header file of the *TransmitControl* definition.

] END_FEAT_TX_CHAN_CPP_020

TransmitDataPush interface

FEAT_TX_CHAN_CPP_030 [TransmitDataPush C++ definition

Implementations in C++ using the feature *Transmit Channel* **shall** use the interface *TransmitDataPush* as declared in the following C++ reference source code extract:

```
class I_TransmitDataPush
{
public :
      virtual void pushBBSamplesTx(
                  BBPacket * thePushedPacket,
                  Boolean endOfBurst) = 0;
};
```

] END_FEAT_TX_CHAN_CPP_030

FEAT_TX_CHAN_CPP_040 [TransmitDataPush C++ header file

Implementations in C++ using the feature *Transmit Channel* **shall** include the header file entitled *TransmitDataPush.h*, corresponding to the header file of the *TransmitDataPush* definition.

] END_FEAT_TX_CHAN_CPP_040

3.2.5.2 VHDL Requirements

The *pushBBSamplesTx* operation of the *Transmit* feature is described in VHDL as:

```
library IEEE;
use IEEE.std_logic_1164.ALL;

entity Transmit is
     generic(
            G_CODING_BITS          : natural := 16      -- for instance
     );
     port (
            -- Common Signals
            clk                    : in  std_logic;     -- Clock signal
            rst_n                  : in  std_logic;     -- Reset signal
            -- pushBBSamplesTx
            BBSample_I             : in  std_logic_vector(G_CODING_BITS-1 downto 0);
            BBSample_Q             : in  std_logic_vector(G_CODING_BITS-1 downto 0);
            BBSample_write         : in  std_logic;
            BBSamplesPacket_start  : in  std_logic;
            BBSamplesPacket_end    : in  std_logic;
            BBSample_ready         : out std_logic
            -- Other signals
          );
end entity Transmit;
```

The Transmit entity signals are summarized in the following table:

Signal Name	Direction	Signal Format	Signal Units	Signal Min Value	Signal Max Value	Notes
Clk	in	1-bit Discrete	Clock	0->1 = Active Edge	1->0 = Passive Edge	XX MHz Transmit Clock
rst_n	in	1-bit Discrete	Level	0	1	Active-low, asynchronous reset.
BBSample_I	In	CodingBits-1 bit Signed	N/A	-32,768	32,767	Imaginary Sample value
BBSample_Q	in	CodingBits-1 bit Signed	N/A	-32,768	32,767	Quadrature Sample value
BBSample_write	in	1-bit Discrete	Level	0	1	Data valid flag to request sample transmission
BBSamplesPacket_start	in	1-bit Discrete	Level	0	1	Packet start flag
BBSamplesPacket_end	in	1-bit Discrete	Level	0	1	Packet end flag
BBSample_ready	out	1-bit Discrete	Level	0	1	Accept/ready flag to activate sample transmission

Table 15: VHDL Transmit Entity signals

A *BBSample* is transmitted only if both 'write' and 'ready' signals are asserted.

The *BBSamplePacket* concept is described in VHDL using additional signals to indicate the start and the end of a sample packet:

Every feature using *BBSamplePacket* concept has to implement the following protocol:

- the *BBSamplesPacket_start* signal is asserted with the first sample of a packet
- the *BBSamplesPacket_end* signal is asserted with the last sample of a package
- In any other case, *BBSamplesPacket_start* and *BBSamplesPacket_end* signals are deasserted

This behaviour is shown in the next timing diagram:

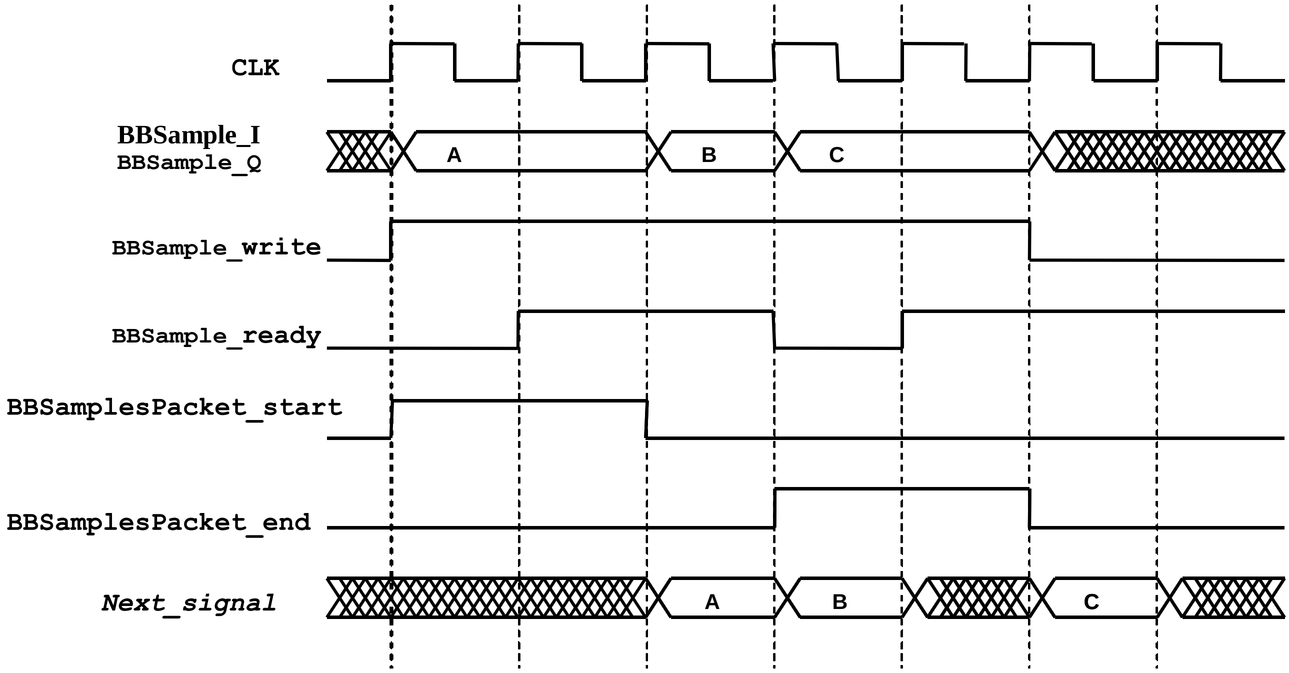


Figure 18: BBSamplePacket Transmit Timing diagram for VHDL

3.3 - Core Feature “Receive Channel”

3.3.1 Principle

For radio reception, a *Receive Channel* within a *Transceiver Subsystem* is down-converting and most probably down sampling and filtering bursts of input *RF signal* into bursts of output *Baseband Signal*. A *Receive Cycle* denotes the phase corresponding to the down-conversion of a particular *RF signal* burst.

A *Receive Channel* implementation is namely composed of the following specific sub-items:

- A *Baseband FIFO*
- A *Down-conversion Chain*

The *Down-conversion Chain* is the signal processing chain which performs the down-conversion of the burst of *RF Signal*, outputting it as a burst of *Baseband Signal* to be contained in *Baseband FIFO*

The programming interface *ReceiveDataPush* enables the samples packets to be retrieved by the *Waveform Application* from the *Receive Channel*. The *ReceiveChannel* takes in charge the packets produced by *Down-Conversion Chain*, stores them into *Baseband FIFO*, where *Waveform Application* retrieves them in real-time and generates the *Baseband Signal*. The samples need to be available in *Baseband FIFO* before the retrieving process starts, and need to be continuously stored in *Baseband FIFO* in a timely manner that prevents signal corruption from occurring.

The programming interface *ReceiveControl* enables the *Waveform Application* to manage when and how *Receive Cycles* shall occur. For each *Receive Cycle*, this control is based on specification of when the concerned RF burst reception shall start and stop, and through characterization of the applicable *Tuning Profile*, which characterizes the exact signal processing transformation to be applied to *RF Signal* by *Down-conversion Chain* on the concerned burst in order to generate the *Baseband Signal*.

The following figure summarizes the previous concepts:

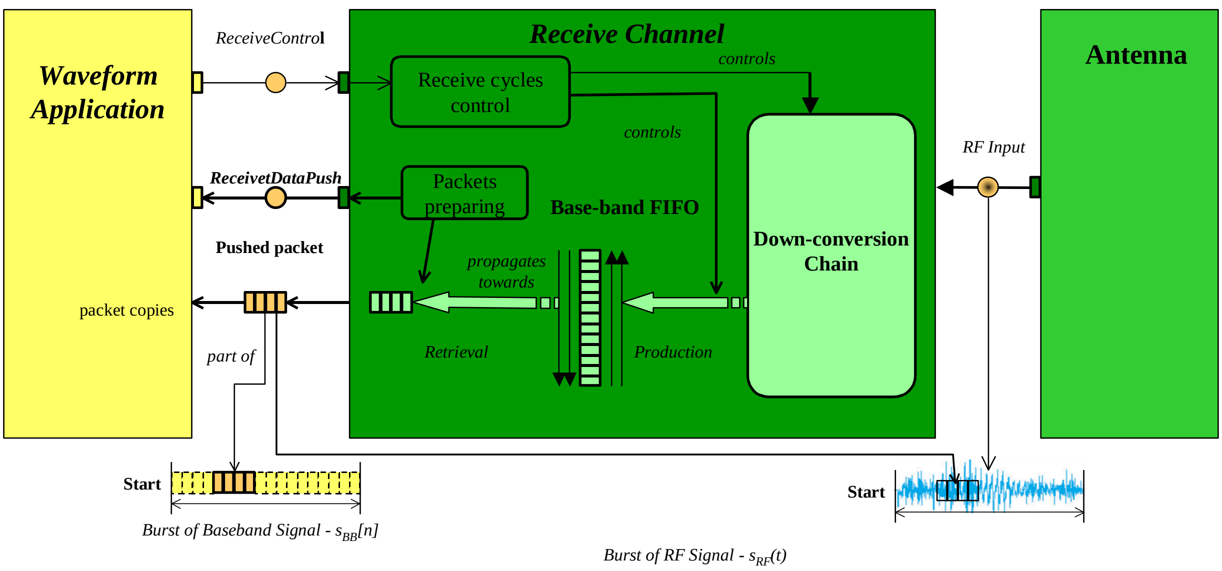


Figure 19: Overview of a Receive Channel

3.3.2 Overview

3.3.2.1 Programming interfaces overview

The following table provides an overview of the programming interface *ReceiveControl:*

Signature (in pseudo-code)	Used by	Realized by	Description

`createReceiveCycleProfile (`
`Time requestedReceiveStartTime,`
`Time requestedReceiveStopTime,`
`Ulong requestedPacketSize,`
`UShort requestedPresetId,`
`Frequency requestedCarrierFrequency);`
	Waveform Application	Transceiver Sub-system	Creation and configuration of a Receive Cycle.

`configureReceiveCycle (`
`Ulong targetCycleId,`
`Time requestedReceiveStartTime,`
`Time requestedReceiveStopTime,`
`Ulong RequestedPacketSize,`
`Frequency requestedCarrierFrequency);`
	Waveform Application	Transceiver Sub-system	Configuration of an existing Receive Cycle.

`setReceiveStopTime (`
`Ulong targetCycleId,`
`Time requestedReceiveStopTime);`
	Waveform Application	Transceiver Sub-system	Configuration or reconfiguration of the end of a Receive Cycle.

Table 16: Overview of programming interface ReceiveControl

The following table provides an overview of the programming interface *ReceiveDataPush*:

Signature (in pseudo-code)	Used by	Realized by	Description

`pushBBSamplesRx(`
`BBPacket thePushedPacket,`
`Boolean endOfBurst);`
	Transceiver Sub-system	Waveform Application	Enable the Waveform Application to retrieve a baseband samples packet from the Receive Channel.

Table 17: Overview of programming interface ReceiveDataPush

3.3.2.2 Characterstistics overview

Explicit Notions

Explicit Notions as defined in §2.3 are *observable*, *implementation **in**-dependent* and *exchanged* characteristics of the *Transceiver Subsystem*. The following table summarizes the *Explicit Notions* related to *Receive Channel*:

Explicit Notion Name	Defined in	Nature	Exchange Mechanism	Associated Argument	Associated Property
`ReceiveCycle`	Core Feature	Logical	Programmable	`targetCycleId`	n.a.
`ReceiveStartTime`	Core Feature	Real-Time	Programmable	`requestedReceiveStartTime`	n.a.
`ReceiveStopTime`	Core Feature	Real-Time	Programmable	`requestedReceiveStopTime`	n.a.
`TuningPreset`	Core Feature	Logical	Programmable	`requestedPresetId`	n.a.
`CarrierFrequency`	Common Concept	Signal Processing	Programmable & Configurable	`requestedCarrierFrequency`	`CarrierFrequency`
`PacketSize`	Common Concept	Signal Processing	Programmable & Configurable	`requestedPacketSize`	`PacketSize`

Table 18: Explicit Notions related to Receive Channel

Implicit Notions

Implicit Notions as defined in §2.3 are *observable*, *implementation **in**-dependent* and *not exchanged* characteristics of the *Transceiver Subsystem*. The following table summarizes the *Implicit Notions* related to *Receive Channel*:

Implicit Notion Name	Defined in	Nature
`BasebandFIFOSize`	Core Feature	Logical
`MaxPushedPacketSize`	Core Feature	Logical
`OverflowMitigation`	Core Feature	Logical

`BasebandSignal:…`
`BasebandSamplingFrequency`
`BasebandCodingBits`
`BasebandNominalPower`
	Common Concept	Signal Processing

Table 19: Implicit Notions related to Receive Channel

Internal Notions

Internal Notions as defined in §2.3 are *observable* and *implementation dependent* characteristics of the *Transceiver Subsystem*. The following table summarizes the *Internal Notions* related to *Receive Channel*:

Name	Defined in	Nature	Transceiver Design Formulas
`TuningDuration`	Core Feature	Real-time	`< MaxTuningDuration`
`TuningStartTime`	Core Feature	Real-time	`= ReceiveStartTime – TuningDuration` `∈ [ReceiveStartTime – MaxTuningDuration; ReceiveStartTime]`
`DownconversionLatency`	Core Feature	Real-time	`< MaxDownConversionLatency`
`ProductionStartTime`	Core Feature	Real-time	`= ReceiveStartTime + DownConversionLatency` `∈ [ReceiveStartTime; ReceiveStartTime + DownConversionLatency]`
`ProductionStopTime`	Core Feature	Real-time	`= ReceiveStopTime + DownConversionLatency` `∈ [ReceiveStopTime; ReceiveStopTime + DownConversionLatency]`
`ReactivationTime`	Core Feature	Real-Time	`= ReceiveStartTime(n) – ReceiveStopTime(n-1)` `> MinReactivationTime`
`RxChannelTransferFunction`	Common Concept	Signal Processing	Fits in mask defined by `ChannelMask`

Table 20: Internal Notions related to Receive Channel

Constraints

Explicit Notions as defined in §2.3 are ***non**-observable* characteristics which are constraining the *Transceiver Subsystem* and/or *Waveform Application* implementation. The following table summarizes the *Constraints* related to *Receive Channel*:

Name of the Constraint	Nature
`MaxTuningDuration`	Real-time
`MaxDownconversionLatency`	Real-time
`MinReceiveStartProximity`	Real-time
`MinReceiveStartAnticipation`	Real-time
`ReceiveTimeProfileAccuracy`	Real-time
`MinReceiveStopAnticipation`	Real-time
`MaxReceiveControlInvocationDuration`	Real-time
`MaxReceiveDataPushInvocationDureation`	Real-time
`MinPacketPreparationTime`	Real-time
`MinReactivationTime`	Real-time
`ChannelMask:…`	Signal Processing
`SpectrumMask:…`	Signal Processing
`GroupDelayMask:…`	Signal Processing
`MaxCycleId`	Logical
`MaxRxCycleProfiles`	Logical

Table 21: Constraints related to Receive Channel

3.3.3 Analysis Requirements

The mnemonic used for identification of requirements of *Transmit Channel* feature is: "RX_CHAN".

3.3.3.1 Receive Channel

FEAT_RX_CHAN_ANA_010 [Definition of a "Receive Channel"

The core feature *Receive Channel* **shall** be used to refer to the capabilities provided by the *Transceiver Sub-system* component that performs down-conversion of a succession of input analogue *RF Signal* bursts into *Baseband Signal* bursts.

] END_FEAT_RX_CHAN_ANA_010

FEAT_RX_CHAN_ANA_020 [Creation of a "Receive Channel"

The *deployment* of a Receive Channel and all its constituents **shall** be undertaken by the Transceiver Sub-system implementation, following implementation specific choices.

The state of a Receive Channel just after its creation and any of its constituent shall be state *Deployed*.

The initialization of the *Receive Channel* shall be conducted by *deployment*.

] END_FEAT_RX_CHAN_ANA_020

The steps setting one *Down-conversion Chain* into state *Deployed* are realized by (re)configuration mechanisms proper to the Transceiver implementation, which are out of the scope of this specification.

FEAT_RX_CHAN_ANA_030 [Composition of a "Receive Channel"

The following concepts **shall** be integrated as constituents of a given *Receive Channel*:

- Concept *Down-conversion Chain*,
- Concept *Receive Baseband Signal FIFO*.

] END_FEAT_RX_CHAN_ANA_030

They are defined in their respective definition requirements as specified in the following paragraphs.

3.3.3.2 Receive Baseband Signal

The *Receive Baseband Signal* is the signal provided by the *Receive Channel* to the *Waveform Application* in order to be processed.

Terms and definitions used in this section are based on those presented within Common Concepts §*4.2.4 Baseband Signal*.

FEAT_RX_CHAN_ANA_040 [Definition of "Receive Baseband Signal"

The concept *Receive Baseband Signal* **shall** describe the useful baseband signal sent by the *Receive Channel* to the *Waveform Application* for baseband signal processing and further treatments.

] END_FEAT_RX_CHAN_ANA_040

Useful baseband signals can include the following possible information:

- Power rising pattern
- Power falling pattern
- ALC and/or AGC patterns
- Data and eventually WF specifics pattern (sequences for synchronization)

FEAT_RX_CHAN_ANA_050 [Implicit Notions of "Receive Baseband Signal"

A *Receive Baseband Signal* **shall** be characterized using the following notions, defined by Common Concept *Baseband Signal*.

- Implicit notion *Baseband Coding Bits*
- Implicit notion *Baseband Nominal Power*

] END_FEAT_RX_CHAN_ANA_050

FEAT_RX_CHAN_ANA_060 [Receive Baseband Signal configuration

The properties of *Receive Baseband Signal* **shall** be set to values appropriate for the considered waveform during Receive Channel deployment.

] END_FEAT_RX_CHAN_ANA_060

3.3.3.3 Receive Baseband FIFO

The *Receive Baseband FIFO* contains baseband signal samples provided by the *Down-conversion Chain* in order for them to be consumed by the *Waveform Application*.

Consumption of baseband samples in the *Receive Baseband FIFO* is realized in a packet mode.

The FIFO monitors samples availability in order to raise error notification in case the samples needed by the *Waveform Application* are not consumed properly (overflow situation).

FEAT_RX_CHAN_ANA_070 [Definition of "Receive Baseband FIFO"

The terminology *Receive Baseband FIFO* **shall** be associated with the part of the sub-part of the *Receive Channel* storing the baseband samples pushed by the *Down-conversion Chain* and consumed by the *Waveform Application*.

] END_FEAT_RX_CHAN_ANA_070

FEAT_RX_CHAN_ANA_080 [Implicit Notion "BasebandFIFOSize"

The implicit notion *BasebandFIFOSize* **shall** be the *Ulong* value which captures the size of the *Receive Baseband FIFO*, capturing the maximum number of baseband samples possibly stored inside the *Baseband FIFO*.

] END_FEAT_RX_CHAN_ANA_080

FEAT_RX_CHAN_ANA_090 [Configuration of "Receive Baseband FIFO"

The properties of *Receive Baseband FIFO* **shall** be set to the values required by the considered *Waveform Application* during the *Receive Channel* deployment.

] END_FEAT_RX_CHAN_ANA_090

Availability deadline and Overflow situations

Any sample of the burst shall be present within baseband FIFO when the Waveform Application is going to consume them. An *availability deadline* is therefore attached to each sample to be consumed.

In case any received sample to be consumed would not be retrieved effectively by the Waveform Application, the Receive Channel goes into a *SignalOverflow* situation. This is an error situation.

FEAT_RX_CHAN_ANA_100 [Error “FIFOOverflow”

During the *Active* state of a *Receive Cycle*, an error *ERR_FIFOOverflow* **shall** be generated if the FIFO runs into an overflow situtation.

] END_FEAT_RX_CHAN_ANA_100

FEAT_TX_CHAN_ANA_110 [FIFO Overrflow ReceiveStop

During the *active* state of a *Receive Cycle*, if *Baseband FIFO* runs into an overflow situation, the reception **shall** be stopped by initiating a *ReceiveStop* transition, immediately after the last sample pushed into *baseband FIFO* has been retrieved by *Waveform Application*.

] END_FEAT_TX_CHAN_ANA_110

It is worth noting that during reception there is only the overflow scenario that deserves consideration as an error, because this means that the waveform application is not consuming samples fast enough to ensure continuous operation. Underflow scenario would signifie only that the waveform is faster than the down-conversion chain.

3.3.3.4 Down-conversion Chain

FEAT_RX_CHAN_ANA_120 [Definition of the “Down-conversion Chain”

Concept *Down-conversion Chain* **shall** denote the signal processing chain in the Receive Channel which undertakes, during periods of time denoted as *Active Receive Cycles*, continuous transformation of one input *RF signal* burst into the corresponding output *Baseband signal* burst.

] END_FEAT_RX_CHAN_ANA_120

FEAT_RX_CHAN_ANA_130 [Constraint “MaxDownconversionLatency”

The constraint *MaxDownconversionLatency* of type *Latency* **shall** be used to characterize the allowable time boundary between the instant an *RF signal* is consumed by the *Down-conversion Chain* and the instant the first corresponding sample is produced for the *Baseband FIFO*.

] END_FEAT_RX_CHAN_ANA_130

This value has to be taken into consideration by the waveform application engineering process. The term *DownconversionLatency* is used in the following sections as an Internal Notion which is implementation related and does not exceed the above constraint.

States and Transitions of Down-conversion Chain

The following figure summarizes the concepts addressed in this section:

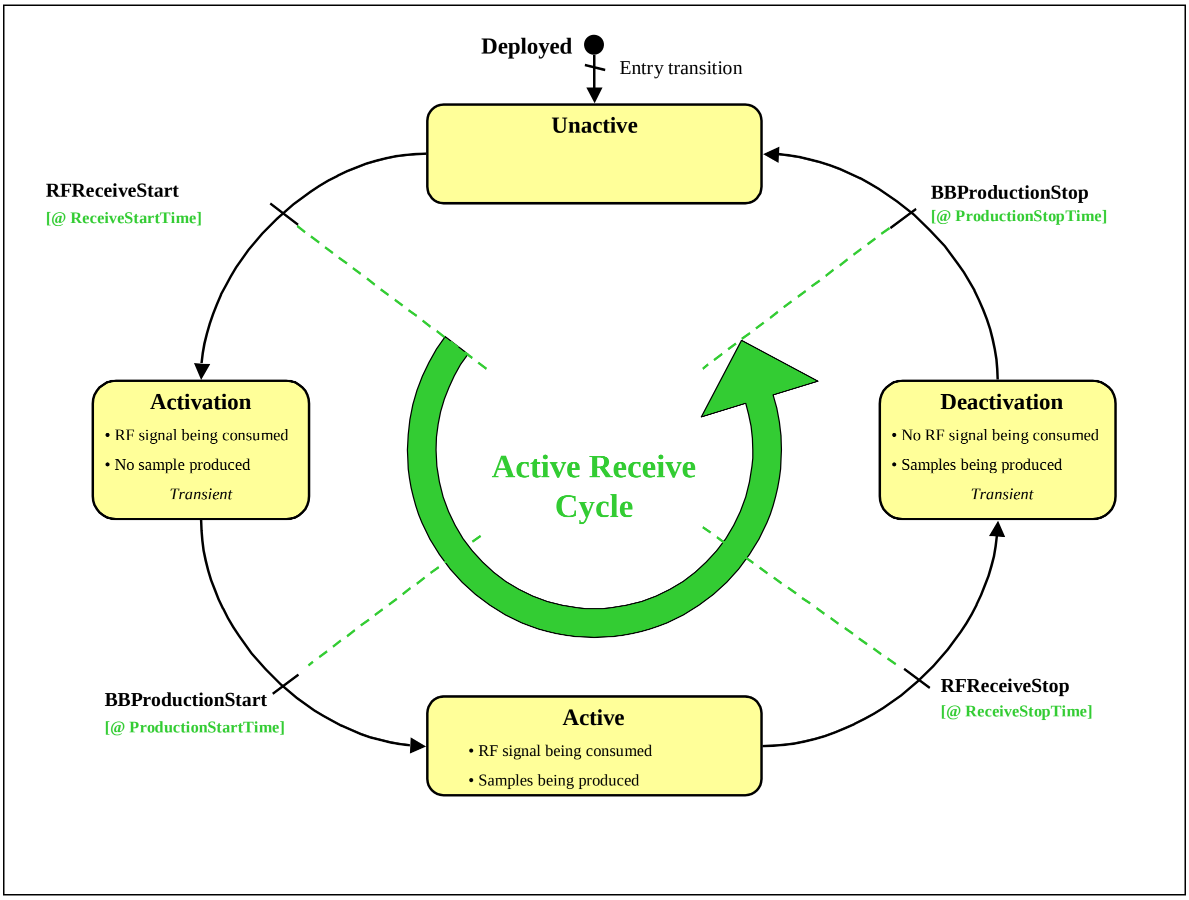


Figure 20: States and transitions of the Down-conversion Chain

FEAT_RX_CHAN_ANA_140 [States of Down-conversion Chain

The following states **shall** be implemented by a *Down-conversion Chain*:

- *Deployed*: Initial state of the chain once *deployed* according to considered Waveform requirements.
- *Unactive*: The state during which the *Down-conversion Chain* is not consuming RF signal, and is not outputing any baseband signal in the *Baseband FIFO*. Automatically reached once *Down-conversion Chain* is *Deployed*.
- *Activation*: The transient state during which the *Down-conversion Chain* has started to consume RF signal, while it is not yet outputting the corresponding baseband signal in the *Baseband FIFO*. Arrival of the useful baseband signal in the *Baseband FIFO* corresponds to the end of the state.
- *Active*: The state during which the input RF signal is continuously consumed by the *Down-conversion Chain*, and transformed into the associated baseband signal which is produced by the chain at its output.
- *Deactivation*: The transient state during which the RF signal is not consumed anymore by the *Down-conversion Chain*, while the previously consumed signal is still being processed by the Down-conversion with associated baseband signal produced. End of useful signal processing corresponds to the end of the state.

] END_FEAT_RX_CHAN_ANA_140

Activation and *Deactivation* are transient states which have a duration equal to the *DownconversionLatency*, which is the time elapsed between (i) the instant when consumption of the *RF signal* occurs at the RF connector and (ii) the instant when the corresponding baseband signal is output in the *Baseband FIFO*.

A *Tuning* activity, not depicted in the preceding figure, is the step during which the *Waveform Application* shall configure a *Receive Cycle* for the reception of the next burst.

FEAT_RX_CHAN_ANA_150 [Transitions between “Down-conversion Chain” states

The instantaneous transitions between states of a Down-conversion Chain **shall** respect the following list:

- *RFReceiveStart*: transition between states *Unactive* and *Activation*
- *BBProductionStart*: transition between states *Activation* and *Active*
- *RFReceiveStop*: transition between states *Active* and *Deactivation*
- *BBProductionStop*: transition between states *Deactivation* and *Unactive*

] END_FEAT_RX_CHAN_ANA_150

FEAT_RX_CHAN_ANA_160 [Timings associated to Down-conversion Chain transitions

The following timings **shall** be associated to instants when instantaneous transitions of Down-conversion Chain are happening:

- *ReceiveStartTime*: associated with transition *RFReceiveStart*
- *ProductionStartTime*: associated with transition *BBProductionStart*
- *ReceiveStopTime*: associated with transition *RFReceiveStop*
- *ProductionStopTime*: associated with transition *BBProductionStop*

] END_FEAT_RX_CHAN_ANA_160

FEAT_RX_CHAN_ANA_170 [Tuning activity

The *Tuning* activity **shall** be used to configure the signal processing requirements captured in *Tuning Profile*. This activity is characterized by the time it starts, *TuningStartTime*, and its duration, *TuningDuration*.

] FEAT_RX_CHAN_ANA_170

FEAT_RX_CHAN_ANA_180 [Constraint “MaxTuningDuration”

The Constraint *MaxTuningDuration* of type *Latency* **shall** be used to characterize the maximum allowable constant duration of the *Tuning* activity.

] END_FEAT_RX_CHAN_ANA_180

This value has to be taken into consideration by the waveform application engineering process. The term *TuningDuration* is used in the following sections as an Internal notion which is implementation related and does not exceed the above constraint.

FEAT_RX_CHAN_ANA_190 [Internal notion “TuningStartTime”

The internal notion *TuningStartTime* **shall** be used to specify the beginning of the *Tuning* activity. Nominal use of *TuningStartTime* is described by the relationships below:

TuningStartTime[n+1] = RFReceiveStart[n+1] – TuningDuration

TuningStartTime[n+1] ≥ ProductionStopTime[n]

Although rarely used, the TuningStartTime[n+1] may start before the ProductionStopTime[n], for example in case of duplicated channels for very fast frequency hopping applications.

] FEAT_RX_CHAN_ANA_190

The following figure summarizes the definitions introduced in this section:

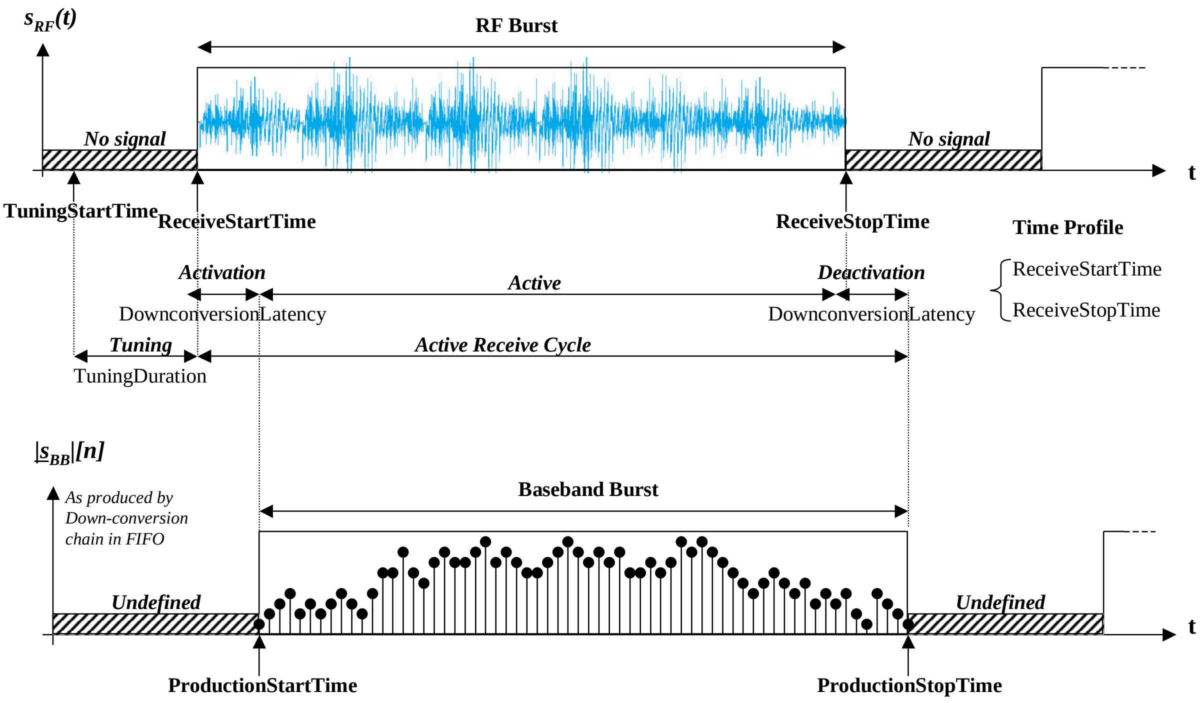


Figure 21: Time Profile of a Receive Cycle

FEAT_RX_CHAN_ANA_200 [Constraint "MinReactivationTime"

The constraint *MinReactivationTime* of type *Latency* **shall** be used to quantify the minimum time elapsed between the ReceiveStopTime of one given Receive Cycle and the ReceiveStartTime of the next one.

] END_FEAT_RX_CHAN_ANA_200

Remark: Transmit Channel has its own *"MinReactivationTime"* constraint, but not equally defined.

Identification of Current Cycle

Default event-based time requests are to be based on the notion of "Current" Transmit Cycle.

FEAT_RX_CHAN_ANA_210 [Definition of the "Current" Receive Cycle

The notion of *current Receive Cycle* **shall** be applied, for operations invoked by the Waveform Application making event-based time requests, as a function of the instant when the considered operation is invoked by *Waveform Application.*

] END_FEAT_RX_CHAN_ANA_210

FEAT_RX_CHAN_ANA_220 [Rule for "Current" Receive Cycle identification

The *current Receive Cycle* **shall** be the *Active Receive Cycle*, if the operation is invoked when a Receive Cycle is active, or the last *Receive Cycle* that finished before invocation.

] END_FEAT_RX_CHAN_ANA_220

Uncertainty of identification of the *current Receive Cycle* thus exists when invocation of *createReceiveCycle()* happens close to the *ReceiveStartTime* of a certain cycle. It is up to Waveform Application design to make appropriate assumptions to avoid calls happening within this time zone.

3.3.3.5 Receive Cycle

A compliant execution by the *Down-conversion Chain* of a given *Receive Cycle* relies on availability of two sorts of information:

- Control data, captured into the *Receive Cycle Profile*
- Output data, denoted as the *Receive Cycle Output Burst*

The *Receive Cycle Output Burst* is the slice of baseband signal, provided by the *Down-conversion Chain* after application of signal processing requirements captured in *Tuning Profile* and real-time requirements captured in *Time Profile*.

Receive Cycle Output Burst

FEAT_RX_CHAN_ANA_230 [Definition of "Receive Cycle Output Burst"

The *Receive Cycle Output Burst* **shall** denote the un-interrupted slice of *Baseband signal* provided by the *Down-conversion Chain* for being transferred and processed by the Waveform Application during the considered *Receive Cycle*.

] END_FEAT_RX_CHAN_ANA_230

FEAT_RX_CHAN_ANA_240 [Definition of "Receive Cycle First Sample"

The *Receive Cycle First Sample* **shall** denote the first sample of the considered output burst.

] END_FEAT_RX_CHAN_ANA_240

FEAT_RX_CHAN_ANA_250 [Definition of "Receive Cycle Last Sample"

The *Receive Cycle Last Sample* **shall** denote the last sample of the considered output burst.

] END_FEAT_RX_CHAN_ANA_250

Receive Cycle Profile

FEAT_RX_CHAN_ANA_260 [Definition of "Receive Cycle Profile"

The concept of *Receive Cycle Profile* **shall** be used to control the *Receive Cycles* implemented during the life-time of one *Down-conversion Chain*, one specific *Receive Cycle Profile* instance being created for each specific *Receive Cycle* occurrence.

] END_FEAT_RX_CHAN_ANA_260

FEAT_RX_CHAN_ANA_270 [Composition of "Receive Cycle Profile"

A *Receive Cycle Profile* **shall** be composed of a *Cycle Identifier*, a *Time Profile* and a *Tuning Profile*.

] END_FEAT_RX_CHAN_ANA_270

Cycle identifier

FEAT_RX_CHAN_ANA_280 [Explicit notion "ReceiveCycle"

Each created cycle has a unique integer identifier *ReceiveCycle* which **shall** be an explicit notion set up during creation to a value incremented by one for each newly created Receive Cycle, starting at 0 (ZERO) for the first created cycle and reaching the value of constraint *MaxCycleId* before restarting counter to 0.

] END_FEAT_RX_CHAN_ANA_280

FEAT_RX_CHAN_ANA_290 [Constraint "MaxRxCycleProfiles"

The constraint *MaxRxCycleProfiles* **shall** refer to the maximum number of *Receive Cycles* simultaneously existing at any time within the Receive Channel.

] END_FEAT_RX_CHAN_ANA_290

Time Profile

The *Time Profile* is set by the *Waveform Application* to define time positioning of the *Receive Cycle*. It is composed of Explicit Notions *ReceiveStartTime* and *ReceiveStopTime*.

They correspond to the only externally observable instants attached to a *Receive Cycle* which are not dependent on the *Down-conversion Chain* implementation.

FEAT_RX_CHAN_ANA_300 [Definition of "Time Profile"

A *Time Profile* **shall** be composed of a *ReceiveStartTime* explicit notion and a *ReceiveStopTime* explicit notion.

] END_FEAT_RX_CHAN_ANA_300

FEAT_RX_CHAN_ANA_310 [Constraint "ReceiveTimeProfileAccuracy"

The constraint *ReceiveTimeProfileAccuracy* **shall** refer to the *Event Accuracy* associated to the *Time Profile* explicit notions, namely *ReceiveStartTime* and *ReceiveStopTime*.

] END_FEAT_RX_CHAN_ANA_310

FEAT_RX_CHAN_ANA_320 [Explicit notion "ReceiveStartTime"

The implementation-independent explicit notion *ReceiveStartTime* **shall** be used within *Time Profile* to contain the *Receive Start Time* of the corresponding *Receive Cycle*.

] END_FEAT_RX_CHAN_ANA_320

FEAT_RX_CHAN_ANA_330 [Referenced event source "ReceiveStart"

The "ReceiveStart" event, associated to the *Receive Start Time* explicit notion, **shall** be the unique *Referenced Event Source* of the Receive Channel.

] END_FEAT_RX_CHAN_ANA_330

FEAT_RX_CHAN_ANA_340 [Explicit notion "ReceiveStopTime"

The implementation-independent explicit notion *ReceiveStopTime* **shall** be used within *Time Profile* to contain the *Receive Stop Time* of the corresponding *Receive Cycle*.

] END_FEAT_RX_CHAN_ANA_340

Explicit notions *ReceiveStartTime* and *ReceiveStopTime* obey accuracy assumptions directly imposed by radio link interoperability conventions between transmitting and receiving radio nodes. Low accuracy on those values is requested for waveforms such as fixed frequency AM/FM speech waveforms, while high accuracy is needed for frequency hopping digitally modulated waveforms.

ReceiveStartTime and *ReceiveStopTime* may be explicitly set by the *Waveform Application* by means of operations arguments. They belong to the Programmable subtype

Implementation-dependent internal notions

Implementation-dependent time internal notions may be derived from implementation-independent explicit notions introduced beforehand, using implementation-dependent durations or latencies:

- *ProductionStartTime* equal to *ReceiveStartTime* plus *DownconversionLatency*
- *TuningStartTime* equal to *ReceiveStartTime* less *TuningDuration*
- *ProductionStopTime* equal to *ReceiveStopTime* plus *DownconversionLatency*

Refer to Figure 21: Time Profile of a Receive Cycle for a graphical representation of above concepts

Tuning Profile

FEAT_RX_CHAN_ANA_350 [Definition of “Receive Tuning Profile”

A *Tuning Profile* **shall** be used to characterize which signal processing explicit notions values shall be applied by the *Down-conversion Chain* when the *Receive Cycle* is activated.

] END_FEAT_RX_CHAN_ANA_350

FEAT_RX_CHAN_ANA_360 [Contents of “Tuning Profile”

The following explicit notions are contained within the *Tuning Profile*:

- PacketSize
- Tuning Preset
- CarrierFrequency

] END_FEAT_RX_CHAN_ANA_360

From one *Receive Cycle* to the next one, depending on the deployed waveform needs, *Tuning Profile* explicit notions may be totally different, partially different, or identical.

Examples

> *In fixed frequency TDD mode with unique channelization, the Tuning Profile is identical from one Receive Cycle to the next.*
>
> *In advanced frequency hopping waveforms, the CarrierFrequency, the Channelization and even the Sampling Rate can be changed from one Receive Cycle to the next, incurring different Tuning Profiles.*

States and Transitions of Receive Cycle Profile

The following figure shows how, for *Active Receive Cycle* number *n* of a given *Down-conversion Chain*, active states of the corresponding *Receive Cycle Profile* are defined:

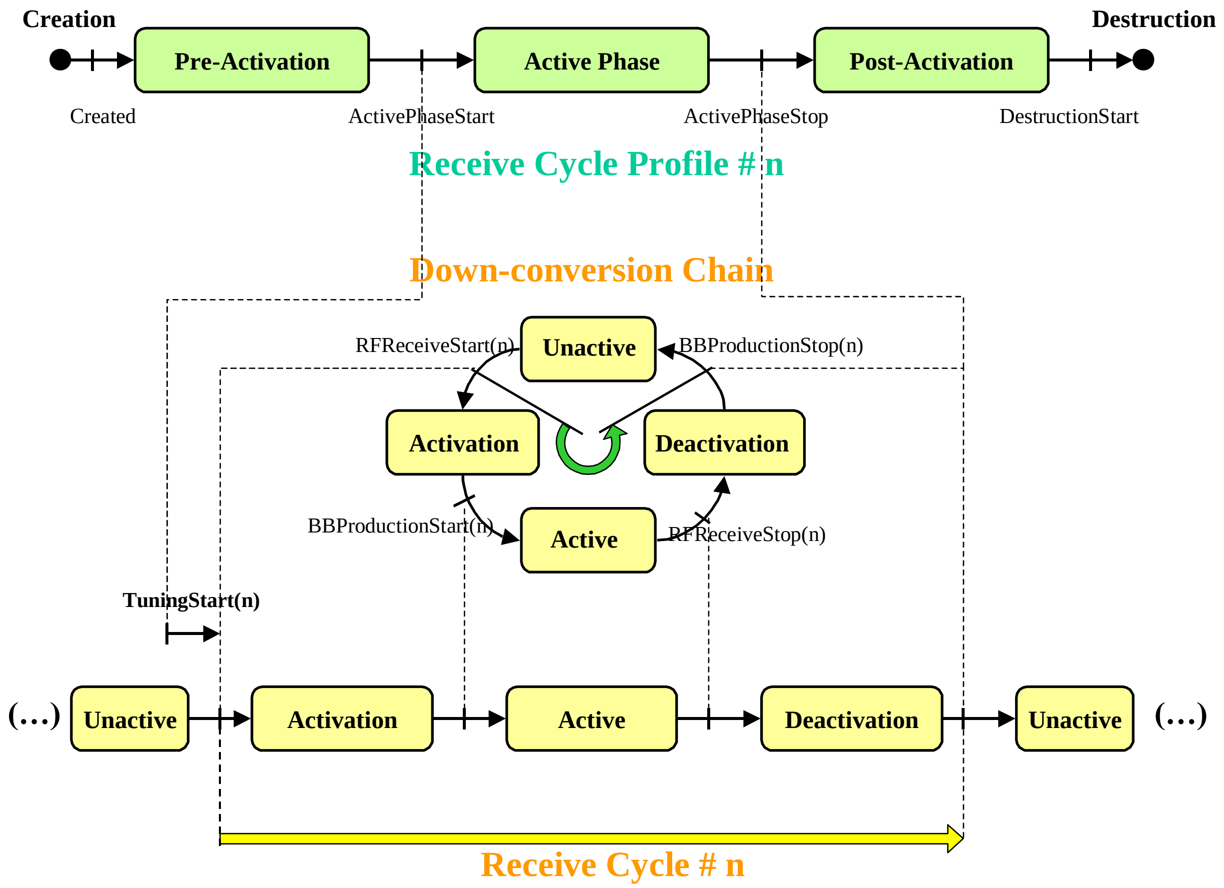


Figure 22: States and Transitions of a Receive Cycle Profile

FEAT_RX_CHAN_ANA_370 [States of a "Receive Cycle Profile"

The following states **shall** be implemented by a *Receive Cycle Profile*:

- *Creation*: The *Receive Cycle Profile* is created by *Receive Channel*.
- *Pre-Activation*: The *Receive Cycle Profile* is kept dormant with possibility for the Waveform Application to update profiles' explicit notions (*Time Profile* or *Tuning Profile*).
- *Active Phase*: The profile values are taken into account by the *Down-conversion Chain* to launch the *Receive Cycle* of interest, with undefined explicit notions completed according to *Tuning*, and updated to reflect what is effectively implemented during the *Receive Cycle.*
- *Post-Activation*: The *Receive Cycle* specified by the *Receive Cycle Profile* is over, but the profile is kept available to let the *Tuning* of the next *Receive Cycle* re-use previous values.
- *Destruction*: The *Receive Cycle Profile* is destroyed by the *Receive Channel*.

] END_FEAT_RX_CHAN_ANA_370

Remark: *Receive Cycle Profile* states are identical to *Transmit Cycle Profile* state, but not their transitions.

FEAT_RX_CHAN_ANA_380 [Transitions of "Receive Cycle Profile"

The instantaneous transitions between states of a *Receive Cycle Profile* **shall** respect the following list:

- *Created*: transition between states *Creation* and *Pre-Activation*
- *ActivePhaseStart*: transition between states *Pre-Activation* and *Active Phase*
- *ActivePhaseStop*: transition between states *Active Phase* and *Post-Activation*
- *DestructionStart*: transition between states *Post-Activation* and *Desctruction*

] END_FEAT_RX_CHAN_ANA_380

FEAT_RX_CHAN_ANA_390 [Transitions correspondance

The transitions of the *Down-conversion Chain* for the *Active Receive Cycle # N* **shall** comply with the following relationships with the transitions of the corresponding instance of *Receive Cycle Profile*:

- *ActivePhaseStart* and *TuningStart(n)* simultaneous
- *ActivePhaseStop* and *BBProductionStop(n)* simultaneous
- *DestructionStart* after than *TransmitStartTime(n+1)*

] END_FEAT_RX_CHAN_ANA_390

3.3.3.6 Active Phase

Triggering conditions

The process leading to the activation of the *Receive Channel Profile* is starting when the *Down-conversion Chain* starts the *Tuning* activity, shortly before state *Activation* starts.

The triggering condition to start *Tuning* activity is based on the definition status of *ReceiveStartTime*:

- If *ReceiveStartTime* is defined, *TuningStart* will be realized so as to have the resulting *RFReceiveStart* happen at *ReceiveStartTime*.
- Another triggering condition can be envisaged. The principle is to trigger the reception at the input of the Down-conversion Chain as soon as the power of the RF signal reaches a threshold power.

If explicit notion *ReceiveStopTime* is set to a defined value when *TuningStart* is happening, the *Receive Cycle* is activated for a pre-defined duration. Otherwise, the cycle ending conditions remain to be defined after the cycle has started.

FEAT_RX_CHAN_ANA_400 [Time-defined transition "StartTuning"

If the explicit notion *ReceiveStartTime* of the *TimeProfile* is defined, the *Receive Channel* **shall** initiate a *TuningStart* transition so as to have the *RFReceiveStart* happen at the instant specified by value of explicit notion *ReceiveStartTime*.

] END_FEAT_RX_CHAN_ANA_400

FEAT_RX_CHAN_ANA_410 [Error "ERR_TooLateRequest"

In case the cycle specified by argument *targetCycleId* has already reached the state *Tuning*, or went further in the life sequence of the *Receive Cycle Profile*, the operation can not be taken into account and an error *ERR_TooLateRequest* **shall** be generated.

] END_FEAT_RX_CHAN_ANA_410

Setting applicable profile

FEAT_RX_CHAN_ANA_420 [Precedence rules for Tuning explicit notions definition

For each Down-conversion processing explicit notion of any given *Receive Cycle*, except the first one, the values applicable by the Down-conversion shall be determined during state *Tuning* in compliance with the following precedence rules, sorted in decreasing order:

- Last value set by a call during the state *Idle*
- Value set during the call to *createReceiveCycle()*
- Value applied for the considered explicit notion on the previous *Receive Cycle*, if defined

] END_FEAT_RX_CHAN_ANA_420

3.3.3.7 Post-Activation

Triggering conditions

State *Post-Activation* of the *Receive Channel Profile* corresponds to when the *Down-conversion Chain* has finished its state *Deactivation* and is back to state *Unactive*.

Entering into state *Post-Activation* is entirely based on the triggering condition *ReceiveStopTime* which causes the *Down-conversion Chain* to enter into state *Deactivation*.

- *ReceiveStopTime* shall be defined and *RF Signal* reception will be stopped at the instant provided by the *ReceiveStopTime* explicit notion value. Baseband samples production will continue until *RF Signal* will be exhausted at instant *BBproductionStop.*
- If ReceiveStopTime value is not defined *RF Signal* receptions will be stopped immediately.

FEAT_RX_CHAN_ANA_430 [Destruction start time

The destruction of a given *Receive Cycle Profile* **shall** not be realized by the Receive Channel until the next *Receive Cycle* has reached its transition *RFReceiveStart*.

] END_FEAT_RX_CHAN_ANA_430

This requirement allows the *Tuning* of the next cycle to be able to re-use the previous *Tuning Profile* for undefined properties.

3.3.3.8 Receive Control

createReceiveCycleProfile()

Purpose

The operation *createReceiveCycleProfile()* enables to request the creation by the Down-conversion chain of a specific *Receive Cycle Profile*. It aims at performing a systematic call for any necessary *Receive Cycle*.

Syntax

FEAT_RX_CHAN_ANA_440 [Operation "createReceiveCycleProfile()"

The operation *createReceiveCycleProfile()* **shall** be implemented by the Receive Channel and be used by the *Waveform Application* in order to request the creation of a particular *Receive Cycle Profile*.

] END_FEAT_RX_CHAN_ANA_440

FEAT_RX_CHAN_ANA_450 [Argument “requestedReceiveStartTime”

The argument *requestedReceiveStartTime* of type *Time* **shall** be used to request an initial value for the explicit notion *ReceiveStartTime*.

] END_FEAT_RX_CHAN_ANA_450

FEAT_RX_CHAN_ANA_460 [Constraint “MinReceiveStartProximity”

If the argument *requestedReceiveStartTime* is an Event-based Time, it shall respect the *Min Event Proximity* constraint specified by “MinReceiveStartProximity”.

] END_FEAT_RX_CHAN_ANA_460

FEAT_RX_CHAN_ANA_470 [Constraint “MinReceiveStartAnticipation”

The constraint *MinReceiveStartAnticipation* of type *Latency* **shall** be used to characterize the minimum allowed elapsed time between the instant the *createReceiveCycleProfile()* operation is invoked and the target Receive start time.

] END_FEAT_RX_CHAN_ANA_470

FEAT_RX_CHAN_ANA_480 [Argument “requestedReceiveStopTime”

The argument *requestedReceiveStopTime* of type *Time* **shall** be used to request an initial value for the explicit notion *ReceiveStopTime*.

] END_FEAT_RX_CHAN_ANA_480

FEAT_RX_CHAN_ANA_490 [Argument “requestedPacketSize”

The argument *RequestedPacketSize* of type ULong **shall** be used in order to request initial values for the *PacketSize* explicit notion.

] END_FEAT_RX_CHAN_ANA_490

FEAT_RX_CHAN_ANA_500 [Argument “requestedPresetId”

The argument *requestedPresetId* of type *UShort* **shall** be used in order to indicate the *PresetId* value of the *Tuning Preset* to be applied on the considered burst.

] END_FEAT_RX_CHAN_ANA_500

The requirements associated to *Tuning Presets* are provided in *Common Concepts*.

FEAT_RX_CHAN_ANA_510 [Argument “requestedCarrierFrequency”

The argument *requestedCarrierFrequency*of type *Frequency* **shall** be used in order to request initial values for the *CarrierFreqcuency* explicit notion.

] END_FEAT_RX_CHAN_ANA_510

Semantics

FEAT_RX_CHAN_ANA_520 [ReceiveCycleProfile initialization

The Receive Channel **shall** set the initial values of the corresponding explicit notions in *ReceiveCycleProfile* to the values specified by passed arguments.

] END_FEAT_RX_CHAN_ANA_520

If argument *requestedReceiveStartTime* is *defined* by the Waveform Application, the Receive Channel will start the Active Phase according to the speficied time. If the initial value of *ReceiveStartTime* was left *undefined*, the Active Phase may not start until the value is further set or signal-based conditions are met.

FEAT_RX_CHAN_ANA_530 [Constraint “MinReceiveStartAnticipation”

If argument *requestedRecieveStartTime* is set to a *defined* value by the Waveform Application, the operation invocation which corresponds with internal notion *ReceiveStartAnticipation* **shall** happen before the corresponding *TuningStartTime* respecting the *MinReceiveStartAnticipation* real-time constraint

] END_FEAT_RX_CHAN_ANA_530

FEAT_RX_CHAN_ANA_540 [Error “ERR_TooManyCreatedRxProfiles”

The Receive Channel **shall** generate an error *ERR_TooManyCreatedRxProfiles* if creation of the *Receive Cycle Profile* would cause a number of created profiles exceeding the value of constraint *MaxRxCycleProfiles*.

] END_FEAT_RX_CHAN_ANA_540

FEAT_RX_CHAN_ANA_550 [“requestedReceiveStartTime” using Absolute Time Request format

Usage of domain type *Absolute Time Request format* **shall** be available as a type possibility for argument *requestedReceiveStartTime*.

] END_FEAT_RX_CHAN_ANA_550

FEAT_RX_CHAN_ANA_560 [“requestedReceiveStartTime” using event-based format

Usage of domain type *Event-based Time Request* **shall** be available as a type possibility for argument *requestedReceiveStartTime*.

] END_FEAT_RX_CHAN_ANA_560

FEAT_RX_CHAN_ANA_570 [Default event source for event-based time requests

The default *Event Source* for event-derived expression of *requestedReceiveStartTime* **shall** be the *ReceiveStartTime* of the *current* Receive Cycle.

] END_FEAT_RX_CHAN_ANA_570

For the first *Receive Cycle* of the *Receive Channel* lifetime, since no previous *Receive Cycle* exists, the *current* Receive Cycle is not defined and an event-based approach can not be used.

FEAT_RX_CHAN_ANA_580 [Default event occurrence for event-derived time requests

The default *event occurrence* for event-derived expression of *requestedReceiveStartTime* **shall** be the *current occurence*.

] END_FEAT_RX_CHAN_ANA_580

`configureReceiveCycle()`

Purpose

This operation is the way a Waveform Application can set the *Down-conversion* profile during its state *Pre-activation*.

Syntax

FEAT_RX_CHAN_ANA_590 [Operation “configureReceiveCycle()”

The operation *configureReceiveCycle()* **shall** be implemented by the Receive Channel and be used by the Waveform Application in order to set all explicit notions values of a previously created *Receive Cycle*.

] END_FEAT_RX_CHAN_ANA_590

FEAT_RX_CHAN_ANA_600 [Target cycle identification in “configureReceiveCycle()”

The *Receive Cycle* for which the requested explicit notions values are applicable **shall** be identified by the argument *targetCycleId* of type *ULong*, which specifies the *Cycle Identifier* of the target Receive Cycle.

] END_FEAT_RX_CHAN_ANA_600

FEAT_RX_CHAN_ANA_610 [Argument “requestedReceiveStartTime”

The argument *requestedReceiveStartTime* of type *Time* **shall** be used to request an initial value for the explicit notion *ReceiveStartTime*.

] END_FEAT_RX_CHAN_ANA_610

FEAT_RX_CHAN_ANA_620 [Constraint “MinReceiveStartAnticipation”

The constraint *MinReceiveStartAnticipation* of type *Latency* **shall** be used to characterize the minimum allowed elapsed time between the instant the *configureReceiveCycle()* operation is invoked and the target Receive start time.

] END_FEAT_RX_CHAN_ANA_620

FEAT_RX_CHAN_ANA_630 [Argument “requestedReceiveStopTime”

The argument *requestedReceiveStopTime* of type *Time* **shall** be used to request an initial value for the explicit notion *ReceiveStopTime*.

] END_FEAT_RX_CHAN_ANA_630

FEAT_RX_CHAN_ANA_640 [Argument “RequestedPacketSize”

The argument *RequestedPacketSize* of type *ULong* **shall** be used to request an initial value for the explicit notion *PacketSize*.

] END_FEAT_RX_CHAN_ANA_640

FEAT_RX_CHAN_ANA_650 [Argument “requestedCarrierFrequency”

The argument *requestedCarrierFrequency* of type *Frequency* **shall** be used to request an initial value for the explicit notion *CarrierFrequency*.

] END_FEAT_RX_CHAN_ANA_650

Errors

FEAT_RX_CHAN_ANA_660 [Error “Unkown Target Cycle”

In case the cycle specified by argument *targetCycleId* is not created, the operation can not be taken into account and an error *Unknown Target Cycle* **shall** be generated.

] END_FEAT_RX_CHAN_ANA_660

setReceiveStopTime()

Purpose

The operation *setReceiveStopTime()* is intended to allow setting the explicit notion *ReceiveStopTime* value for a *Receive Cycle.*

FEAT_RX_CHAN_ANA_670 [Operation "setReceiveStopTime()"

The operation *setReceiveStopTime()* **shall** be implemented by the *Receive Channel* and be used by the Waveform Application in order to set the *ReceiveStopTime* of the considered *Receive Cycle*.

] END_FEAT_RX_CHAN_ANA_670

FEAT_RX_CHAN_ANA_680 [Argument "requestedReceiveStopTime"

The argument *requestedReceiveStopTime* **shall** be available in operation *setReceiveStopTime()* in order to enable the Waveform Application to specify when the transition *RFReceiveStop* shall happen.

] END_FEAT_RX_CHAN_ANA_680

FEAT_RX_CHAN_ANA_690 [Constraint "MinReceiveStopAnticipation"

The constraint *MinReceiveStopAnticipation* of type *Latency* **shall** be used to characterize the minimum allowed elapsed time between the instant the *setReceiveStopTime()* operation is invoked and the target Receive stop time.

] END_FEAT_RX_CHAN_ANA_690

FEAT_RX_CHAN_ANA_700 [Defined "requestedReceiveStopTime"

If argument *requestedReceiveStartTime* is set to a defined value by Waveform Application, the Receive Channel **shall** deactivate the *Receive Cycle* in respect of the specified value.

] END_FEAT_RX_CHAN_ANA_700

FEAT_RX_CHAN_ANA_710 [Undefined "requestedReceiveStopTime" (instant Deactivation)

If argument *requestedReceiveStartTime* is set to *undefined* by Waveform Application, the Receive Channel **shall** immediately turn the current *Transmit Cycle* into *Deactivation* state.

] END_FEAT_RX_CHAN_ANA_710

FEAT_RX_CHAN_ANA_720 [Constraint "MinReceiveStopAnticipation"

The operation *setReceiveStopTime()* **shall** be called with an anticipation relative to the *requestedReceiveStopTime* at least equal to the value of the constraint *"MinReceiveStopAnticipation*, of type *Latency*.

] END_FEAT_RX_CHAN_ANA_720

The operation *setReceiveStopTime()* may be used to modify as often as necessary the *ReceiveStopTime* of a *Transmit Cycle*, provided it always respects the *ReceiveStopAnticipation* invocation margin.

Errors

FEAT_RX_CHAN_ANA_730 [Error "ERR_TooLateRequest"

If the operation *setReceiveStopTime()* is invoked after *ReceiveStopTime – MinReceiveStopAnticipation*, the *Receive Channel* is not able to guarantee correct application, and an error *ERR_TooLateRequest* **shall** be generated.

] END_FEAT_RX_CHAN_ANA_730

Interface real-time constraints

FEAT_RX_CHAN_ANA_740 [Constraint "MaxReceiveControlInvocationDuration"

Operations of the *ReceiveControl* interface **shall** respect the *MaxInvocationDuration* constraint specified by *MaxReceiveControlInvocationDuration*.

] END_FEAT_RX_CHAN_ANA_740

3.3.3.9 Receive Data Push

Overview

The programming interface *Receive Data Push* enables the *Waveform Application* to retrieve packets of baseband samples from the *Receive Channel*.

The programming interface relies on: (i) a single operation, *pushBBSamplesRx()*, (ii) the data retrieved by the *Waveform Application*, denoted as *The Pushed Packet*.

pushBBSamplesRx() is used by the *Transceiver Subsystem* to: notify the *Waveform Application of* the availability of a new packet in the *Baseband FIFO* of size *Packet size.*

The Pushed Packet structure is defined by common concept *Baseband Packet*. It must be prepared by the *Down-Conversion Chain* prior to the *pushBBSamplesRx()* notification. The samples are retrieved by the *Waveform Application* in conformance with specific *packets contents requirements*.

Prior to *pushBBSamplesRx()* notification, the *Receive Channel* undertakes *packets handling* activities:

- First, it *takes in charge* the *pushed packet*, copying its content into an internal memory zone.
- Second, it *stores* the corresponding data into the *Baseband FIFO*.

The way a *Receive Channel* implementation takes in charge the *pushed packet* and stores it into the *Baseband FIFO* is implementation dependent. Most reactive implementations will take in charge and store in baseband FIFO the pushed packet in one single data copy. More complex designs may exist, especially when the digital part of the *Transceiver Subsystem* is distributed over different digital signal processing units.

Under notification by the *pushBBSamplesRx()*, the *Waveform Application* launches a *packet copy* of the *pushed packet* into its own memory. The *packet copy* is an implementation-dependent mechanism, which may involve several elementary copies (for instance pushed samples to be distributed at both ends of a circular buffer).

The Pushed Packet

Structural requirements

FEAT_RX_CHAN_ANA_750 [Definition of "The Pushed Packet"

A data constuct denoted *The Pushed Packet*, of type *Baseband Packet*, **shall** be used to contain the Baseband Signal samples to be retrieved.

] END_FEAT_RX_CHAN_ANA_750

FEAT_RX_CHAN_ANA_760 [Creation of "The Pushed Packet"

The Pushed Packet **shall** be created by the *Receive Channel*.

] END_FEAT_RX_CHAN_ANA_760

FEAT_RX_CHAN_ANA_770 [Visibility on "The Pushed Packet"

The Pushed Packet **shall** be visible by the *Waveform Application* with *read* rights and be visible by the *Receive Channel* with *write* rights.

] END_FEAT_RX_CHAN_ANA_770

FEAT_RX_CHAN_ANA_780[Destruction of "The Pushed Packet"

The Pushed Packet **shall** be destroyed by the *Receive Channel.*

] END_FEAT_RX_CHAN_ANA_780

Packets ordering requirements

This section specifies a certain number of requirements applicable to packets content.

FEAT_RX_CHAN_ANA_790 [Packets burst alignment

The *pushed* baseband samples **shall** be aligned with the *Receive Cycles Input Bursts*, which means no packet contains samples belonging to two different input bursts.

] END_FEAT_RX_CHAN_ANA_790

FEAT_RX_CHAN_ANA_800 [Sequential bursts transmission

The *pushed* baseband samples packets **shall** be transmitted one burst after the other.

] END_FEAT_RX_CHAN_ANA_800

FEAT_RX_CHAN_ANA_810 [Ordered packets pushes

The *pushed* baseband samples packets **shall** not be interverted within a given burst.

] END_FEAT_RX_CHAN_ANA_810

Packets identification

The *first packet* of a *burst* is defined as the packet where the *first sample* is the *first received sample* of the *burst*.

The last packet of a *burst* is defined as the packet where the *last sample* is the *last received sample* of the *burst.*

A packet containing **all** the samples of a given burst is at the same time the *first* and *last* packet of the burst.

The *last packet* of a burst is therefore systematically followed by the *first packet* of the next burst.

pushBBSamplesTx()

Syntax

FEAT_RX_CHAN_ANA_820 [Operation "pushBBSamplesRx()"

Operation *pushBBSamplesRx()* **shall** be used by the *Transceiver Subsystem* and realized by the *Waveform Application*, in order to: (i) notify the *Waveform Application* of the availability of a new packet of samples, and (ii) indicate if the pushed packet is the last packet of the received burst.

] END_FEAT_RX_CHAN_ANA_820

FEAT_RX_CHAN_ANA_830 [Argument "thePushedPacket"

The argument *thePushedPacket*, of type *BBPacket,* **shall** contain information enabling access to the pushed packet.

] END_FEAT_RX_CHAN_ANA_830

The pushed packet content shall have been prepared by the *Receive Cycle* to contain the correct signal before invocation. The size of the pushed packet is determined by the value of argument *requestedPacketSize* set during the Receive Cycle creation or profile configuration.

FEAT_RX_CHAN_ANA_840 [Argument "endOfBurst"

The argument *endOfBusrt*, of type *Boolean*, **shall** be used to indicate that the pushed packet is the last packet of the current *Receive Cycle Burst*.

Value *True* if the packet is the last packet, *False* or *undefined* otherwise.

] END_FEAT_RX_CHAN_ANA_840

Semantics

FEAT_RX_CHAN_ANA_850 [pushBBSamplesRx() Pre-Invocation

When *invocation* of *pushBBSamplesRx()* occurs, the samples values contained in the pushed packet **shall** be compliant with the packet contents requirements.

] END_FEAT_RX_CHAN_ANA_850

FEAT_RX_CHAN_ANA_860 [pushBBSamplesRx() Post-Invocation

When *return* of *pushBBSamplesRx()* returns, the samples values contained in the pushed packet **shall** have been taken into account by the *Waveform Application.*

] END_FEAT_RX_CHAN_ANA_860

Realt-time Constraints

FEAT_RX_CHAN_ANA_870 [Constraint "MaxReceiveDataPushInvocationDuration"

The constraint *MaxReceiveDataPushInvocationDuration*, of type *Latency*, **shall** be used to identify the *Max Invocation Duration* to be met by *pushBBSamplesRx* implementations.

] END_FEAT_RX_CHAN_ANA_870

As noted in §2.3, the *Max Invocation Duration* of an operation corresponds to the time difference between:

- The instant when an operation *invocation* occurs,
- The instant when an operation *return* occurs.

The constraint may remain *undefined* if the real-time constraints of the radio capability do not justify it.

Max pushBBSamplesRx Invocation Duration is generally assigned a unique value, taking into consideration the maximum packet size used by the waveform application.

FEAT_RX_CHAN_ANA_880 [Constraint “MaxPacketPreparationTime”

The constraint *MaxPacketPreparationTime*, of type *Latency*, **shall** be used to identify the *Maximum Time* to be guaranteed in order to have a packet prepared in *Baseband FIFO* by the *Transceiver Subsystem* before a particular target time.

] END_FEAT_RX_CHAN_ANA_880

Transceiver Subsystem should be compliant with this constraint in order to avoid out of synchronization situations or scenarios where *Transceiver Subsystem* delays the signal leading to real-time violation.

Overflow mitigation

Two overflow mitigation options can be selected:

- PacketsTrashing: *Transceiver Subsystem* will trash the packets if the FIFO is completely filled and the waveform application does not retrieve them or does so at a slower pace.
- No Overflow: reaching overflow is corresponded to an error so the *Transceiver Subsystem* is requested to generate an error.

FEAT_RX_CHAN_ANA_890 [Implicit notion “OverflowMitigation”

The enumerated implicit notion *Overflow Mitigation* **shall** be used to set the applicable overflow mitigation option.

The possible values are: *PacketsTrashing*, and *No Overflow*. *Undefined* corresponds to *Packets Trashing*.

] END_FEAT_RX_CHAN_ANA_890

FEAT_RX_CHAN_ANA_900 [PacketsTrashing Overflow Mitigation

When implicit notion *Overflow Mitigation* is set to *PacketsTrashing*, the Transceiver Subsystem **shall** discard extra packets once the FIFO buffer reachs its maximum size.

] END_FEAT_RX_CHAN_ANA_900

This mode is kept as the default mode since it does not requireany supplementary dependency between the Waveform Application and the platform. This occurs at the expense of handling trivial synchronization mechanisms.

FEAT_RX_CHAN_ANA_910 [No Overflow Overflow Mitigation

When implicit notion *Overflow Mitigation* is set to *No Overflow*, the Transceiver Subsystem **shall** raise an error if an overflow of the FIFO occurs.

] END_FEAT_RX_CHAN_ANA_910

This mode can be used in cases where the waveform design determines that overflow cases shall not occur.

3.3.3.10 Analysis Requirements Summary

The following table provides an overview of the Analysis requirements of Receive Channel*:*

Identifier	Tag	Section
FEAT_RX_CHAN_ANA_010	Definition of a “Receive Channel“	Receive Channel
FEAT_RX_CHAN_ANA_020	Creation of a “Receive Channel”	Receive Channel
FEAT_RX_CHAN_ANA_030	Composition of a “Receive Channel”	Receive Channel
FEAT_RX_CHAN_ANA_040	Definition of “ Receive Baseband Signal”	Receive Baseband Signal
FEAT_RX_CHAN_ANA_050	Implicit Notions of “Receive Baseband Signal”	Receive Baseband Signal
FEAT_RX_CHAN_ANA_060	Receive Baseband Signal configuration	Receive Baseband Signal
FEAT_RX_CHAN_ANA_070	Definition of “Receive Baseband FIFO”	Receive Baseband FIFO
FEAT_RX_CHAN_ANA_080	Implicit Notion “BasebandFIFOSize”	Receive Baseband FIFO
FEAT_RX_CHAN_ANA_090	Configuration of “Receive Baseband FIFO”	Receive Baseband FIFO
FEAT_RX_CHAN_ANA_100	Error “FIFOOverflow”	Receive Baseband FIFO
FEAT_TX_CHAN_ANA_110	FIFO Overrflow ReceiveStop	Receive Baseband FIFO
FEAT_RX_CHAN_ANA_120	Definition of the “Down-conversion Chain”	Down-conversion Chain
FEAT_RX_CHAN_ANA_130	Constraint “MaxDownconversionLatency”	Down-conversion Chain
FEAT_RX_CHAN_ANA_140	States of Down-conversion Chain	Down-conversion Chain
FEAT_RX_CHAN_ANA_150	Transitions between “Down-conversion Chain” states	Down-conversion Chain
FEAT_RX_CHAN_ANA_160	Timings associated to Down-conversion Chain transitions	Down-conversion Chain
FEAT_RX_CHAN_ANA_170	Tuning activity	Down-conversion Chain
FEAT_RX_CHAN_ANA_180	Constraint “MaxTuningDuration”	Down-conversion Chain
FEAT_RX_CHAN_ANA_190	Internal notion “TuningStartTime”	Down-conversion Chain
FEAT_RX_CHAN_ANA_200	Constraint “MinReactivationTime”	Down-conversion Chain
FEAT_RX_CHAN_ANA_210	Definition of the “Current” Receive Cycle	Down-conversion Chain
FEAT_RX_CHAN_ANA_220	Rule for “Current” Receive Cycle identification	Down-conversion Chain
FEAT_RX_CHAN_ANA_230	Definition of “Receive Cycle Output Burst”	Receive Cycle
FEAT_RX_CHAN_ANA_240	Definition of “Receive Cycle First Sample”	Receive Cycle
FEAT_RX_CHAN_ANA_250	Definition of “Receive Cycle Last Sample”	Receive Cycle
FEAT_RX_CHAN_ANA_260	Definition of “Receive Cycle Profile”	Receive Cycle
FEAT_RX_CHAN_ANA_270	Composition of “Receive Cycle Profile”	Receive Cycle
FEAT_RX_CHAN_ANA_280	Explicit notion “ReceiveCycle”	Receive Cycle
FEAT_RX_CHAN_ANA_290	Constraint “MaxRxCycleProfiles”	Receive Cycle
FEAT_RX_CHAN_ANA_300	Definition of “Time Profile”	Receive Cycle
FEAT_RX_CHAN_ANA_310	Constraint “ReceiveTimeProfileAccuracy”	Receive Cycle
FEAT_RX_CHAN_ANA_320	Explicit notion “ReceiveStartTime”	Receive Cycle
FEAT_RX_CHAN_ANA_330	Referenced event source “ReceiveStart”	Receive Cycle
FEAT_RX_CHAN_ANA_340	Explicit notion “ReceiveStopTime”	Receive Cycle
FEAT_RX_CHAN_ANA_350	Definition of “Receive Tuning Profile”	Receive Cycle
FEAT_RX_CHAN_ANA_360	Contents of “Tuning Profile”	Receive Cycle
FEAT_RX_CHAN_ANA_370	States of a “Receive Cycle Profile”	Receive Cycle
FEAT_RX_CHAN_ANA_380	Transitions of “Receive Cycle Profile”	Receive Cycle
FEAT_RX_CHAN_ANA_390	Transitions correspondance	Receive Cycle
FEAT_RX_CHAN_ANA_400	Time-defined transition “StartTuning”	Active Phase
FEAT_RX_CHAN_ANA_410	Error “ERR_TooLateRequest”	Active Phase
FEAT_RX_CHAN_ANA_420	Precedence rules for Tuning explicit notions definition	Active Phase
FEAT_RX_CHAN_ANA_430	Desctruction start time	Post-Activation
FEAT_RX_CHAN_ANA_440	Operation “createReceiveCycleProfile()”	Receive Control
FEAT_RX_CHAN_ANA_450	Argument “requestedReceiveStartTime”	Receive Control
FEAT_RX_CHAN_ANA_460	Contraint “MinReceiveStartProximity”	Receive Control
FEAT_RX_CHAN_ANA_470	Constraint “MinReceiveStartAnticipation”	Receive Control
FEAT_RX_CHAN_ANA_480	Argument “requestedReceiveStopTime”	Receive Control
FEAT_RX_CHAN_ANA_490	Argument “requestedPacketSize”	Receive Control
FEAT_RX_CHAN_ANA_500	Argument “requestedPresetId”	Receive Control
FEAT_RX_CHAN_ANA_510	Argument “requestedCarrierFrequency”	Receive Control
FEAT_RX_CHAN_ANA_520	ReceiveCycleProfile initialization	Receive Control

FEAT_RX_CHAN_ANA_530	Constraint "MinReceiveStartAnticipation"	Receive Control
FEAT_RX_CHAN_ANA_540	Error "ERR_TooManyCreatedRxProfiles"	Receive Control
FEAT_RX_CHAN_ANA_550	"requestedReceiveStartTime" using Absolute Time Requests format	Receive Control
FEAT_RX_CHAN_ANA_560	"requestedReceiveStartTime" using event-based format	Receive Control
FEAT_RX_CHAN_ANA_570	Default event source for event-based time requests	Receive Control
FEAT_RX_CHAN_ANA_580	Default event occurrence for event-derived time requests	Receive Control
FEAT_RX_CHAN_ANA_590	Operation "configureReceiveCycle()"	Receive Control
FEAT_RX_CHAN_ANA_600	Target cycle identification in "configureReceiveCycle()"	Receive Control
FEAT_RX_CHAN_ANA_610	Argument "requestedReceiveStartTime"	Receive Control
FEAT_RX_CHAN_ANA_620	Constraint "MinReceiveStartAnticipation"	Receive Control
FEAT_RX_CHAN_ANA_630	Argument "requestedReceiveStopTime"	Receive Control
FEAT_RX_CHAN_ANA_640	Argument "RequestedPacketSize"	Receive Control
FEAT_RX_CHAN_ANA_650	Argument "requestedCarrierFrequency"	Receive Control
FEAT_RX_CHAN_ANA_660	Error "Unkown Target Cycle"	Receive Control
FEAT_RX_CHAN_ANA_670	Operation "setReceiveStopTime()"	Receive Control
FEAT_RX_CHAN_ANA_680	Argument "requestedReceiveStopTime"	Receive Control
FEAT_RX_CHAN_ANA_690	Constraint "MinReceiveStopAnticipation"	Receive Control
FEAT_RX_CHAN_ANA_700	Defined "requestedReceiveStopTime"	Receive Control
FEAT_RX_CHAN_ANA_710	Undefined "requestedReceiveStopTime" (instant Deactivation)	Receive Control
FEAT_RX_CHAN_ANA_720	Constraint "MinReceiveStopAnticipation"	Receive Control
FEAT_RX_CHAN_ANA_730	Error "ERR_TooLateRequest"	Receive Control
FEAT_RX_CHAN_ANA_740	Constraint "MaxReceiveControlInvocationDuration"	Receive Control
FEAT_RX_CHAN_ANA_750	Definition of "The Pushed Packet"	Receive Data Push
FEAT_RX_CHAN_ANA_760	Creation of "The Pushed Packet"	Receive Data Push
FEAT_RX_CHAN_ANA_770	Visibility on "The Pushed Packet"	Receive Data Push
FEAT_RX_CHAN_ANA_780	Destruction of "The Pushed Packet"	Receive Data Push
FEAT_RX_CHAN_ANA_790	Packets burst alignment	Receive Data Push
FEAT_RX_CHAN_ANA_800	Sequential bursts transmission	Receive Data Push
FEAT_RX_CHAN_ANA_810	Ordered packets pushes	Receive Data Push
FEAT_RX_CHAN_ANA_820	Operation "pushBBSamplesRx()"	Receive Data Push
FEAT_RX_CHAN_ANA_830	Argument "thePushedPacket"	Receive Data Push
FEAT_RX_CHAN_ANA_840	Argument "endOfBurst"	Receive Data Push
FEAT_RX_CHAN_ANA_850	pushBBSamplesRx() Pre-Invocation	Receive Data Push
FEAT_RX_CHAN_ANA_860	pushBBSamplesRx() Post-Invocation	Receive Data Push
FEAT_RX_CHAN_ANA_870	Constraint "Max pushBBSamplesRx Invocation Duration"	Receive Data Push
FEAT_RX_CHAN_ANA_880	Constraint "MaxPacketPreparationTime"	Receive Data Push
FEAT_RX_CHAN_ANA_890	Implicit notion "OverflowMitigation"	Receive Data Push
FEAT_RX_CHAN_ANA_900	PacketsTrashsing Overflow Mitigation	Receive Data Push
FEAT_RX_CHAN_ANA_910	No Overflow Overflow Mitigation	Receive Data Push

Table 22: Overview of Receive Channel Analysis requirements

3.3.4 Modelling Requirements

3.3.4.1 Receive Channel interfaces

The following class diagram involves the interfaces realized and used by the *Receive Channel*, between the *Waveform Application* and the *Transceiver Subsystem*:

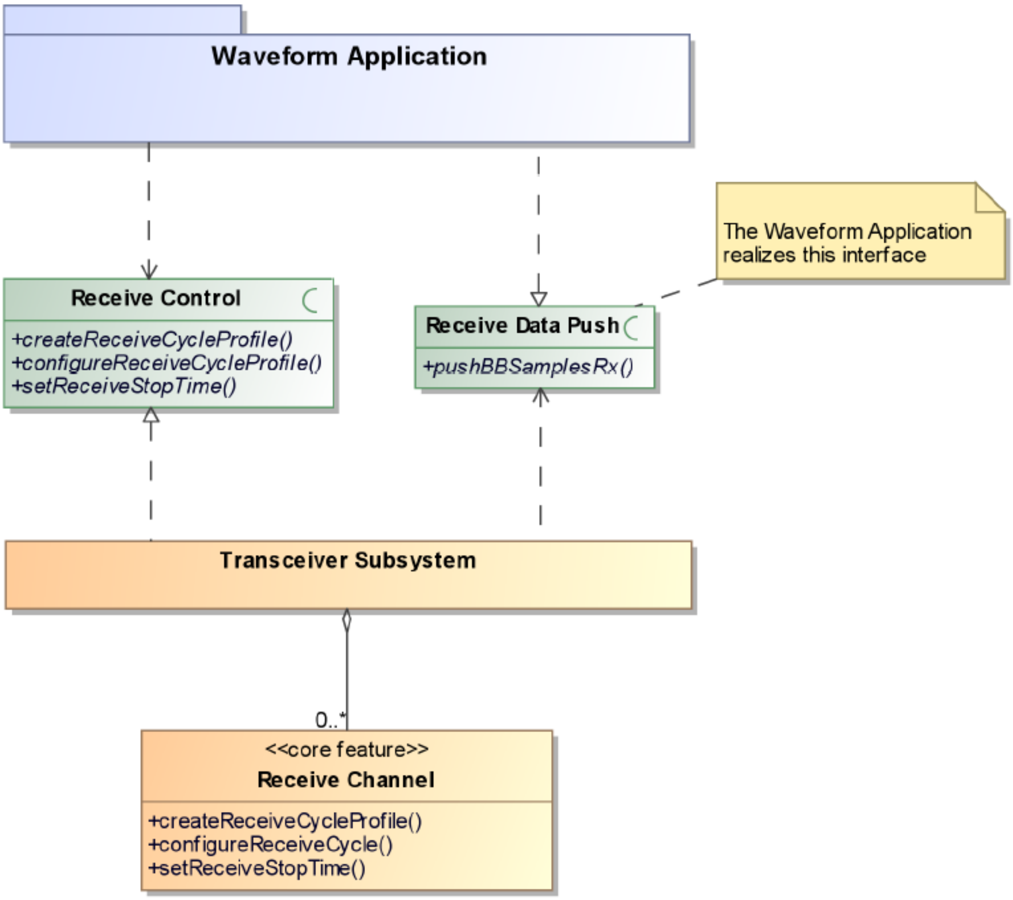


Figure 23: Receive Channel involved interfaces

FEAT_RX_CHAN_MOD_010 [Receive Control interface

Receive Control is the interface used to control the Receive Channel behaviors according to the Waveform needs. This interface **shall** include the following operations:

- *createReceiveCycleProfile()*
- *configureReceiveCycle()*
- *setReceiveStopTime()*

Definitions of operations and behaviors of their realizations by the Receive Channel core feature are specified in the baseline requirements of the Receive Channel.

] END_FEAT_RX_CHAN_MOD_010

FEAT_RX_CHAN_MOD_020 [Receive Data Push interface

Receive Data Push is the interface used for transferring baseband samples from the Transceiver Subsystem to the Waveform Application. This interface **shall** include the following operation:

- *pushBBSamplesRx()*

Definition of this operation and associated behaviour of its realization by the Waveform Application are specified in the baseline requirements of the Receive Channel.

] END_FEAT_RX_CHAN_MOD_020

Note: Signatures of the operations are not provided for better visibility. Specification of the operations is depicted in paragraph *Baseline Requirements* of this chapter.

3.3.4.2 Receive Channel composition

FEAT_RX_CHAN_MOD_030 [Receive Channel modelling elements 1

Transceiver Subsystem model involving feature *Receive Channel* **shall** use the modelling elements depicted in the following class diagram:

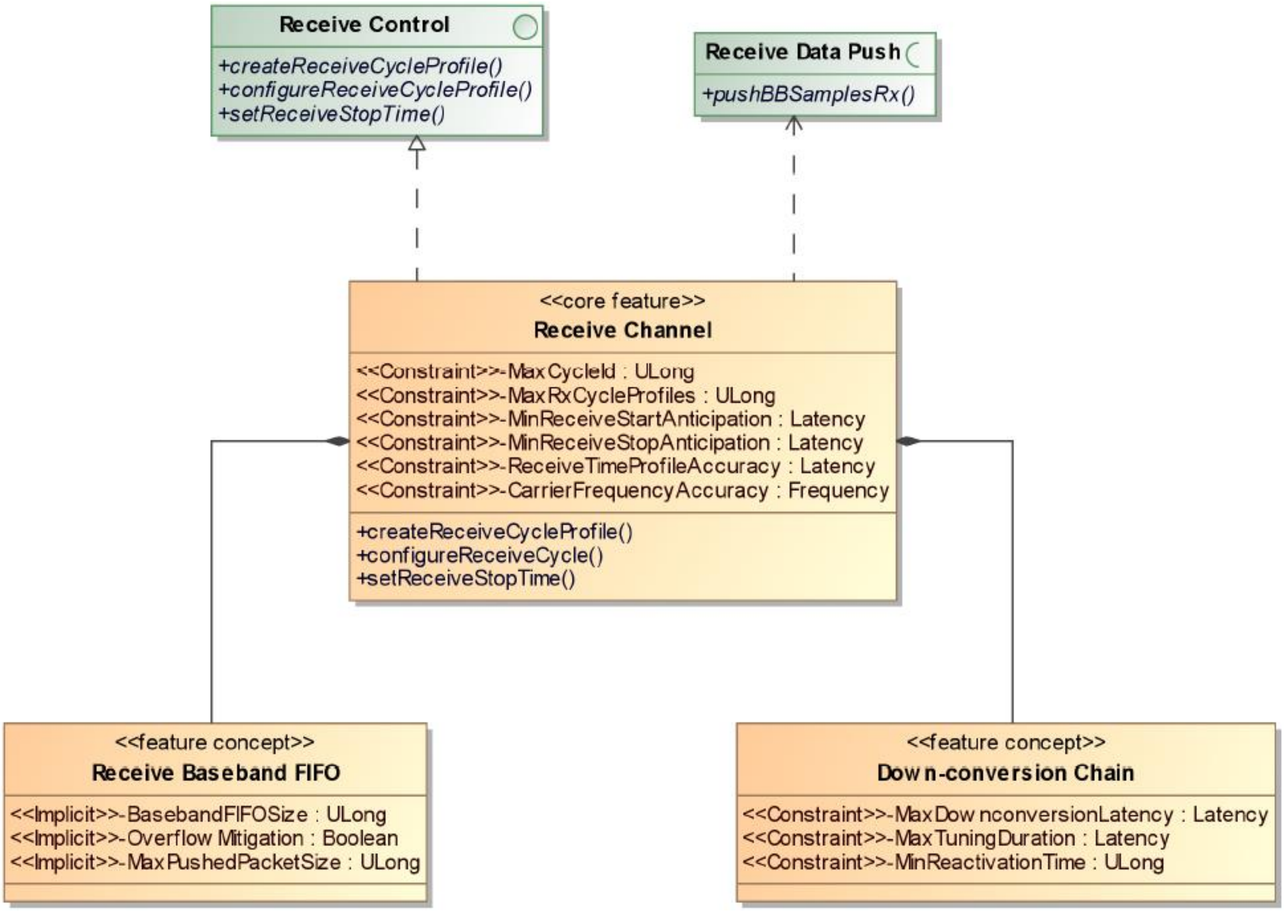


Figure 24: Receive Channel main composition

] END_FEAT_RX_CHAN_MOD_030

FEAT_RX_CHAN_MOD_040 [Receive Channel modelling elements 1

Transceiver Subsystem model involving feature *Receive Channel* **shall** use the modelling elements depicted in the following class diagram:

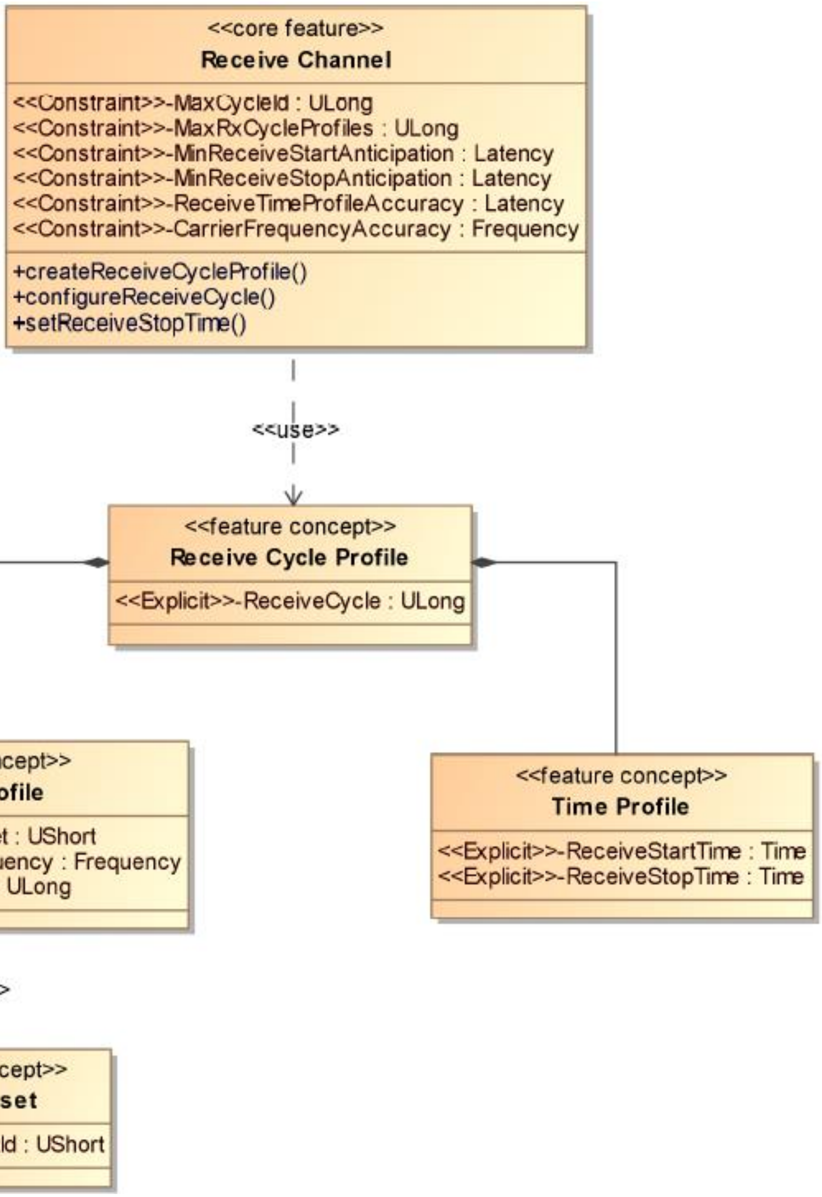


Figure 25: Receive Channel and Cycle Profile

] END_FEAT_RX_CHAN_MOD_040

3.3.5 Implementation Languages Requirements

3.3.5.1 C++ Requirements

Interfaces between the *Waveform Application* and the *Transceiver Subsystem* for the *Receive Channel* core feature are:

- Receive Control,
- Receive Data Push.

Both are seen in a modeling analysis as abstract classes defining abstract operations. In C++ language, this leads to consider these operations as pure virtual methods of its respective classes. These methods can not be implemented by these abstract classes, but shall be overridden and implemented by derived classes.

ReceiveControl interface

FEAT_RX_CHAN_CPP_010 [ReceiveControl C++ definition

Implementations in C++ using the feature *Receive Channel* **shall** use the interface *ReceiveControl* as declared in the following C++ reference source code extract:

```
class I_ReceiveControl
{
public :
	virtual void createReceiveCycleProfile(
			Time			requestedReceiveStartTime,
			Time			requestedReceiveStopTime,
			ULong			requestedPacketSize,
			UShort			requestedPresetId,
			Frequency		requestedCarrierFrequency) = 0;

	virtual void configureReceiveCycle(
			ULong			targetCycleId,
			Time			requestedReceiveStartTime,
			Time			requestedReceiveStopTime,
			ULong			requestedPacketSize,
			Frequency		requestedCarrierFrequency) = 0;

	virtual void setReceiveStopTime(
			ULong			targetCycleId,
			Time			requestedTransmitStopTime) = 0;
};
```

] END_FEAT_RX_CHAN_CPP_010

FEAT_RX_CHAN_CPP_020 [ReceiveControl C++ header file

Implementations in C++ using the feature *Receive Channel* **shall** include the header file entitled *ReceiveControl.h*, corresponding to the header file of the *ReceiveControl* definition.

] END_FEAT_RX_CHAN_CPP_020

ReceiveDataPush interface

FEAT_RX_CHAN_CPP_010 [ReceiveDataPush C++ definition

Implementations in C++ using the feature *Receive Channel* **shall** use the interface *ReceiveDataPush* as declared in the following C++ reference source code extract:

```
class I_ReceiveDataPush
{
public :
	virtual void pushBBSamplesRx(
			BBPacket * thePushedPacket,
			Boolean endOfBurst) = 0;
};
```

] END_FEAT_RX_CHAN_CPP_010

FEAT_RX_CHAN_CPP_020 [ReceiveDataPush C++ header file

Implementations in C++ using the feature *Receive Channel* **shall** include the header file entitled *ReceiveDataPush.h*, corresponding to the header file of the *ReceiveDataPush* definition.

] END_FEAT_RX_CHAN_CPP_020

3.3.5.2 *VHDL Requirements*

The *pushBBSamplesRx* operation of the *Receive* feature is described in VHDL as:

```
library IEEE;
use IEEE.std_logic_1164.ALL;

entity Receive is
     generic(
            G_CODING_BITS          : natural := 16     -- for instance
     );
     port (
            -- Common Signals
            clk                    : in  std_logic;    -- Clock signal
            rst_n                  : in  std_logic;    -- Reset signal
            -- pushBBSamplesTx
            BBSample_I             : out  std_logic_vector(G_CODING_BITS-1 downto 0);
            BBSample_Q             : out  std_logic_vector(G_CODING_BITS-1 downto 0);
            BBSample_write         : out  std_logic;
            BBSamplesPacket_start : out  std_logic;
            BBSamplesPacket_end   : out  std_logic;
            BBSample_ready         : in   std_logic
            -- Other signals
          );
end entity Receive;
```

The Receive entity signals are summarized in the following table:

Signal Name	Direction	Signal Format	Signal Units	Signal Min Value	Signal Max Value	Notes
Clk	in	1-bit Discrete	Clock	0->1 = Active Edge	1->0 = Passive Edge	XX MHz Transmit Clock
rst_n	in	1-bit Discrete	Level	0	1	Active-low, asynchronous reset.
BBSample_I	out	CodingBits-1-bit Signed	N/A	-32,768	32,767	Imaginary Sample value
BBSample_Q	out	CodingBits-1-bit Signed	N/A	-32,768	32,767	Quadrature Sample value
BBSample_write	out	1-bit Discrete	Level	0	1	Data valid flag to request sample transmission
BBSamplesPacket_start	out	1-bit Discrete	Level	0	1	Packet start flag
BBSamplesPacket_end	out	1-bit Discrete	Level	0	1	Packet end flag
BBSample_ready	in	1-bit Discrete	Level	0	1	Accept/ready flag to activate sample transmission

Table 23: VHDL Receive Entity signals

A *BBSample* is transmitted only if both 'write' and 'ready' signals are asserted.

The *BBSamplePacket* concept is described in VHDL using additional signals to indicate the start and the end of a sample packet.

Every Feature using BBSamplePacket concept has to implement the following protocol:

- the *BBSamplesPacket_start* signal is asserted with the first sample of a packet,
- the *BBSamplesPacket_end* signal is asserted with the last sample of a package,
- In any other case, *BBSamplesPacket_start* and *BBSamplesPacket_end* signals are deasserted

This behaviour is shown in the next timing diagram:

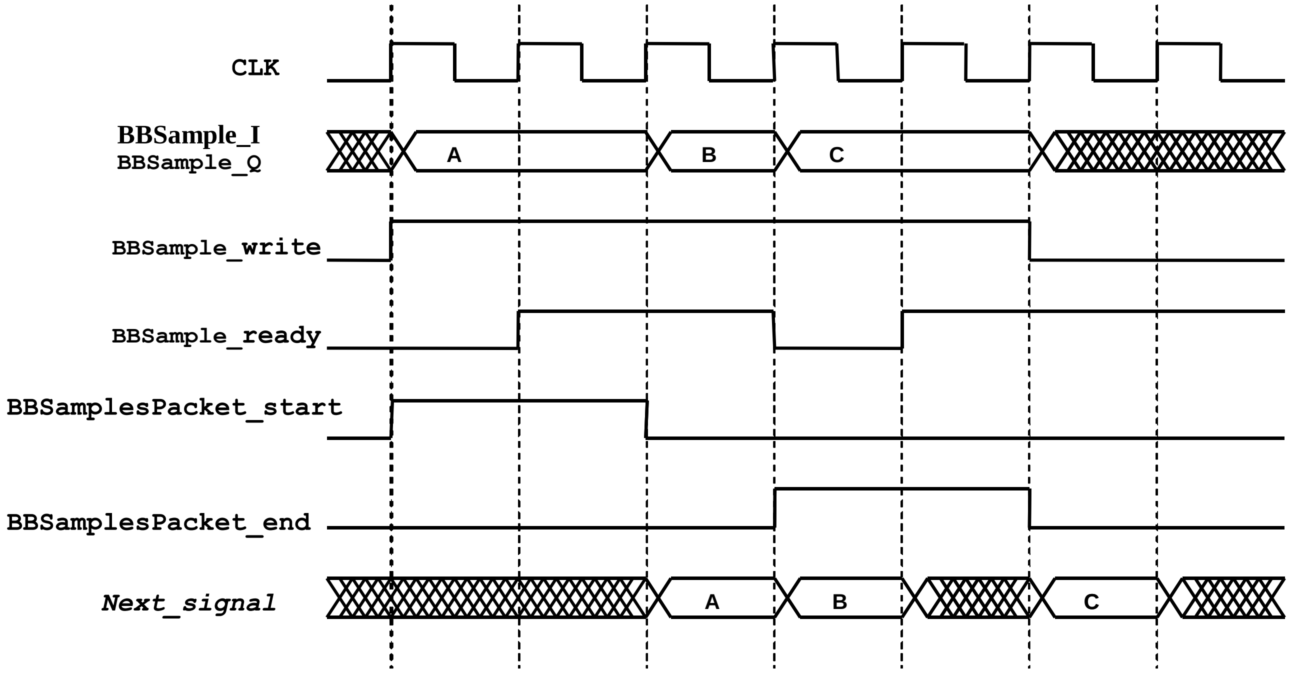


Figure 26: BBSamplePacket Transmit Timing diagram for VHDL

4 - Common Concepts Specification

4.1 - Introduction

The following chapters specify *Common Concepts*:

- Time handling Domain Types
- Tuning Characteristics Definition
- Tuning Characteristics Setting
- Baseband Packets

Overview of specified Characteristics

The following table identifies the *notions* specified by *Common Concepts*:

Name of the Notion		Category	Sub-category (if applicable)
`Baseband Signal`	`BasebandSamplingFrequency`	Implicit	
	`BasebandCodingBits`	Implicit	
	`BasebandNominalPower`	Implicit	
`RF Signal`	`CarrierFrequency`	Explicit	Programmable & Configurable
	`NominalRFPower`	Explicit	Programmable & Configurable
`ChannelTransferFunction`		Internal	
`PresetId`		Explicit	Programmable

Table 24: Overview of Common Concepts notions

`PresetId` is a programmable argument, and takes its values from a requested argument through interfaces defined in core features. In Transmit Channel and Receive Channel features, **`PresetId`** is requested through argument **`requestedTuningPreset`**, respectively in operations *createTransmitCycleProfile()* and *createReceiveCycleProfile()*.

The following tables identify the *constraints* specified by *Common Concepts*:

Name of the Constraint		Category
`Channel Mask`	`ChannelBandwidth`	Signal Processing
	`CarrierFrequencyAccuracy`	Signal Processing
`Spectrum Mask`	`Ripple`	Signal Processing
	`HighBoundTransitionBand`	Signal Processing
	`HighBoundRejectionGain`	Signal Processing
	`HighBoundRejectionSlope`	Signal Processing
	`LowBoundTransitionBand`	Signal Processing
	`LowBoundRejectionGain`	Signal Processing
	`LowBoundRejectionSlope`	Signal Processing
`Group Delay Mask`	`MaxGroupDelayDispersion`	Signal Processing

Table 25: Overview of Common Concepts constraints

Refer to Figure 27: Characteristics of Spectrum Mask and Figure 28: Characteristics of Group Delay Mask in the following paragraphs for an schematic representation of the above Constraints.

4.2 - Time handling Domain Types

This chapter specifies requirements applicable to domain times used to specify time values.

CC_TIME_000 [Generic Domain Type "Time"

The generic domain type *Time* **shall** be used for arguments of *control operations* used to specify a *time request*, specifying the *Event Time* of a particular event. The associated physical unit is the *second*.

The generic type can be derived into type *Absolute Time* or *Event-based Time*.

] END_CC_TIME_000

MinAnticipation and *Accuracy* are *constraints* attached by the *Facility* core specification to each control operation. Each *Waveform Capability* is subject to define specific values for those properties, else they remain undefined.

4.2.1 Absolute Time

Principle

Absolute Time uses a time representation convention which counts the duration between an *Absolute Time Reference* and the *Event Time*. The *Absolute Time Reference* is equal to the begining of year 2000, i.e. January 1st 2000, 00:00:00.

This count is realized using an *Absolute Time*, composed of an *unsigned 32 bit* value, complemented by an *unsigned 32 bit value*, counting the nanoseconds.

Absolute Time values handle a notion of time which corresponds to the time accessible to the *Transceiver Sub-system*.

Handling of the corresponding *time source* (by the *Waveform Application* or by *Radio Set* specific mechanisms), namely to set it to the right time with the required accuracy, is not in the scope of the current specification.

Specification

CC_TIME_010 [Domain type "Absolute Time"

The domain type *Absolute Time* **shall** be used to express time requests defined as the time elapsed from January 1st 2000, 00:00:00 (24h) to the *Event Time*.

It is derived from generic domain type *Time*.

] END_CC_TIME_010

CC_TIME_020 [Default setting for "Time Specification"

The default setting for the domain type *Time Specification* **shall** be expressed using a couple of *unsigned 32 bit* values, respectively representing the number *Second Count* of seconds (*s*) elapsed since the reference time, and the number *Nanosecond Count* of supplementary nano-seconds (*ns*) within the referenced second.

] END_CC_TIME_020

The maximal accessible date approximately equals to 4.10^9 seconds after 1-jan-00, i.e. slightly more than 136 years. The default setting provides a resolution of 1 ns.

4.2.2 Event-based Time

Principle

Event-based Time uses a time representation convention which identifies the requested *Event Time* using a reference event available within the platform, from which a time shift enables accurate identification of the desired *Event Time*.

The reference event is identified using a particular event source, which is identified using its public identifier, in conformance with the principles exposed in § *Event Sources*. The occurrence count value and the notion of last and previous events, introduced in this chapter, are used as well.

Event-based time presents the advantage of avoiding (i) the *Waveform Application* part realizing control operations on the *Transceiver Sub-system* and (ii) the *Transceiver Sub-system* itself having knowledge of the absolute date.

It requires, as a counterpart, to carry on a generally more difficult real-time engineering effort, and may not be applicable to the radio interoperability protocols of most demanding waveforms.

Requirements

CC_TIME_030 [Domain type “Event-based Time”

The domain type *Event-based Time* **shall** be used to express a *Time Request* defined through (i) identification of a *Reference Event*, a specific past or coming event generated by the *Platform Support*, (ii) provision of a *Time Shift* to apply from this *Reference Event* to compute the *Event Time*.

It is derived from generic domain type *Time*.

] END_CC_TIME_030

CC_TIME_040 [Composition of an “Event-based Time”

The domain type *Event-based Time* **shall** be a structure composed of the four following fields:

- Event Source Id,
- Event Count Origin,
- Event Count,
- Time Shift.

] END_CC_TIME_040

Fields *Event Source Id*, *Event Count Origin* and *Event Count* together define the *Reference Event*.

CC_TIME_050 [Field “Event Source Id”

The field *Event Source Id* of the domain type *Event-based Time* **shall** be an *unsigned 16 bit* value, identifying the *Event Source* from the *Platform Support* used to define the *Reference Event*.

] END_CC_TIME_050

CC_TIME_060 [Undefined “Event Source Id” value

An *Undefined* value for field *Event Source Id* **shall** correspond to the *Default Event Source* attached to the *control operation*.

] END_CC_TIME_060

CC_TIME_070 [Defined field "Event Source Id"

A *Defined* value for field *Event Source Id* **shall** refer to the *Event Source Identifier* of the event source used to identify the *Reference Event*.

] END_CC_TIME_070

CC_TIME_080[Field "Event Count Origin"

The field *Event Count Origin* of the domain type *Event-based Time* **shall** be an *enumerated* field, taking values between {*Beginning*, *Previous*, *Next*}, identifying the origin from which the events of the event source are counted in order to identify the *Reference Event*.

] END_CC_TIME_080

CC_TIME_090 [Enumerated value "Beginning"

The enumerated value *Beginning* of the enumerated field *Event Count Origin* **shall** be used to indicate that the *Reference Event* shall be counted from **first occurrence** of the *Event Source*.

] END_CC_TIME_090

CC_TIME_100 [Enumerated value "Previous"

The enumerated value *Previous* of the enumerated field *Event Count Origin* **shall** be used to indicate that the *Reference Event* shall be counted from the **previous occurrence** of the *Event Source*, relative to the instant when the operation is invoked.

] END_CC_TIME_100

CC_TIME_110 [Enumerated value "Next"

The enumerated value *Next* of the enumerated field *Event Count Origin* **shall** be used to indicate that the *Reference Event* shall be counted from the **next occurrence** of the *Event Source*, relative to the instant when the operation is invoked.

] END_CC_TIME_110

CC_TIME_120 [Field "Event Count"

The field *Event Count* of the domain type *Event-based Time* **shall** be a *signed 32 bit* value which identifies the number of *Event Source* occurences separating the *Reference Event* from the *Event Count Origin*.

] END_CC_TIME_120

A value of 0 for *Event Count Value* means that the *Reference Event* is equal to the *Event Count Origin*.

CC_TIME_130 [Undefined "Event Count" value

An *Undefined* value for field *Event Count* **shall** not be used by the *Waveform Application*.

] END_CC_TIME_130

CC_TIME_140 [Field "Time Shift"

The field *Time Shift* of the domain type *Event-based Time* **shall** be a domain type *Latency* value which gives the positive or negative amount of time separating the *Requested Time* from the *Reference Event*.

] END_CC_TIME_140

4.3 - Tuning Characteristics Definition

4.3.1 Introduction

The *Tuning Characteristics Definition* exhaustively identifies the set of *notions* and *constraints* which need to be assigned a value in order for the Up-/Down-conversion Chains to be tuned before activation.

This chapter explains the question of definition of the *Tuning Characteristics*.

4.3.2 Domain definitions

Baseband Power

The *Baseband Power* of *Baseband Signal* is measured over a time period consistent with the stationarity of the measured signal.

dB_{FS}

A *Baseband Power* value is expressed in "dB relative to full scale", abbreviated as "dB_{FS}", capturing the power ratio between the measured signal and a reference harmonic signal. The reference full scale signal is defined as a complex harmonic signal with an envelope magnitude equal to the maximum envelope possible coded using the full scale of the stream defined by the number of coding bits.

Nominal reception conditions

In reception, nominal conditions are defined as conditions when a reference signal with a constant level is applied at Low Power RF level, while a baseband signal is generated towards the Waveform Application, using nominal reception gain settings.

Nominal transmissionconditions

In transmission, nominal conditions are defined as conditions when a reference signal with a constant level is applied at baseband signal level by the Waveform Application, while a Low Power RF Level is generated at Transceiver Subsystem output, using nominal transmission gain settings.

Absolute uncertainty

The *Absolute Uncertainty*, denoted *dx*, attached to a specified property value *x*, shall be such that the *Transceiver Subsystem* implementation guarantees a measurable implementation value comprised in the interval [*x*-*dx*;*x*+*dx*].

4.3.3 Domain types

This chapter introduces *domain types* applicable to characteristics of (i) the *common concept* and (ii) *features specifications*.

The requirement attached to the *domain type* definition indicates the signification of the type with the associated physical unit, while the *default setting* is attached to a default numerical representation.

The default setting is not useful in the context of the characteristic remaining virtually defined with no representation in the *Transceiver Subsystem* implementation. Any design value can then be taken into consideration.

If the considered characteristic is represented, in any respect, in the *Transceiver Subsystem* implementation, the default setting is then applicable to characterize how the associated characteristic shall be numerically represented. This is realized using a *base type* and identifying the chosen sub-unit for this representation.

Floating point representation is voluntarily ignored to achieve generic radio configuration designs.

When accessing implemented characteristics, a read operation will retrieve values compliant with the setting chosen for the characteristic. This value will represent the true value with an *accuracy* eventually documented in the *Transceiver Sub-system* implementation release notes.

Frequency

CC_DOMAIN_TYPE_000 [Domain type "Frequency"

The domain type *Frequency* **shall** be used to refer to any frequency expressions within the specification, using *hertz (Hz)* as the default associated physical unit.

] END_CC_DOMAIN_TYPE_000

CC_DOMAIN_TYPE_020 [Default setting for "Frequency"

The default setting for the domain type *Frequency* **shall** be expression of values represented using *ULong* values, representing positive values in steps of 1 hertz.

] END_CC_BB_DOMAIN_TYPE_020

The default setting provides a resolution of 1 Hz, a minimal value of 0 Hz, and a maximal value of 4,294,967,295 Hz.

Note that for some applications, i.e satellite communications, the frequency range is not large enough. Further improvements are thus needed, such as a so-called "type of range" setting, which would set the appropriate frequency range (application dependent) during the deployment.

Baseband Power

CC_DOMAIN_TYPE_030 [Domain type "Baseband Power"

The domain type *Baseband Power* **shall** be used to refer any baseband signal power expression, using *decibels-relative-to-full-scale* (dB_{FS}) as the associated unit.

] END_CC_DOMAIN_TYPE_030

CC_DOMAIN_TYPE_040 [Default setting for "Baseband Power"

The default setting for the domain type *Baseband Power* **shall** be an expression of values represented using *Short* values, representing signed values in steps of 0.1 dB_{FS}.

] END_CC_BB_DOMAIN_TYPE_040

The default setting provides a resolution of 0.1 dB_{FS}, a minimum value of -3276.8 dB_{FS}, and a maximum value of +3276.7 dB_{FS}. Such values largely exceed meaningful physical values, towards low or high values. In particular, exceeding by more than a few dB the full scale signal is not physically possible.

Analogue Power

CC_DOMAIN_TYPE_050 [Domain type "Analogue Power"

The domain type *Analogue Power* **shall** be used to refer any analogue signal power expression, using *decibels-relative-to-1-milliwatt* (dB_m) as the associated physical unit.

] END_CC_DOMAIN_TYPE_050

CC_DOMAIN_TYPE_060 [Default setting for "Analogue Power"

The default setting for the domain type *Analogue Power* **shall** be an expression of values represented using *Short* values, representing signed values in steps of 0.1 dBm.

] END_CC_DOMAIN_TYPE_060

The default setting provides a resolution of 0.1 dB_m, a minimum value of -3276.8 dB_m, and a maximum value of +3276.7 dB_m. The step value chosen, 0.1 dBm, satisfy most of the current Transceiver architectures.

Latency

CC_DOMAIN_TYPE_070 [Domain type “Latency”

The domain type *Latency* **shall** be used to refer to any analogue signal power expression, using *seconds (s)* as the associated physical unit.

] END_CC_DOMAIN_TYPE_070

CC_DOMAIN_TYPE_080 [Default setting for “Latency”

The default setting for the domain type *Latency* **shall** be an expression of values represented using *Ulong* values, representing signed values in steps of 1 ns.

] END_CC_DOMAIN_TYPE_080

The default setting provides a resolution of 1 ns, a minimum value of 0 ns, and a maximum value of +4,294,967,295 ns.

Gain

CC_DOMAIN_TYPE_090 [Domain type “Gain”

The domain type *Gain* **shall** be used to refer to any gain expression, using *decibel (dB)* as the associated physical unit.

] END_CC_DOMAIN_TYPE_090

CC_DOMAIN_TYPE_100 [Default setting for “Gain”

The default setting for the domain type *Gain* **shall** be an expression of values represented using *Short* values, representing signed values in steps of 0.1 dB.

] END_CC_DOMAIN_TYPE_100

The default setting provides a resolution of 0.1 dB, a minimum value of -3276.8 dB, and a maximum value of +3276.7 dB.

Gain Slope

CC_DOMAIN_TYPE_110 [Domain type “Gain Slope”

The domain type *Gain Slope* **shall** be used to refer to any gain slope expression, using *decibels per kilohertz (dB/kHz)* as the associated physical unit.

] END_CC_DOMAIN_TYPE_110

CC_DOMAIN_TYPE_120 [Default setting for “Gain Slope”

The default setting for the domain type *Gain Slope* **shall** be expression of values represented using *Short* values, representing signed values in steps of 0.1 dB/kHz.

] END_CC_DOMAIN_TYPE_120

The default setting provides a resolution of 0.1 dB/kHz, a minimum value of -3276.8 dB/kHz, and a maximum value of +3276.7 dB/kHz.

4.3.4 Baseband Signal

This chapter identifies the burst tuning characteristics attached to the *Baseband Signal*.

CC_BB_SIGNAL_000 [Definition of "Baseband Signal"

The concept of *Baseband Signal* **shall** be used to refer to the baseband analytic signal exchanged between Waveform Application and (i) *Transmit Channels* for transmission, (ii) *Receive Channels* for reception.

] END_CC_BB_SIGNAL_000

The implicit notions attached to *Baseband Signal* are:

- *Baseband Sampling Frequency*
- *Baseband Coding Bits*
- *Baseband Nominal Power*

Notion "Baseband Sampling Frequency"

CC_BB_SIGNAL_010 [Notion "Baseband Sampling Frequency"

The implicit notion *Baseband Sampling Frequency*, of type *Frequency*, **shall** be used for the sampling rate of the complex analytic baseband signal exchanged at the *Baseband Signal*.

] END_CC_BB_SIGNAL_010

Usage of default settings provide a range of values going from 0 to4.2 GHz by steps of 1 Hz.

Notion "Baseband Coding Bits"

CC_BB_SIGNAL_020 [Notion "Baseband Coding Bits"

The implicit notion *Baseband Coding Bits*, expressed using *UShort* values, **shall** be used for the number of bits used for quantification of each of the InPhase and InQuadrature (I,Q) components of the *Baseband Signal* it is referring to.

] END_CC_BB_SIGNAL_020

Any signal in the stream with (I,Q) components instant values exceeding the maximum possible dynamic range shall be cut off in order to respect the value of *Baseband Coding Bits*.

The typical values for *Baseband Coding Bits* are from 12 to 20, depending on the dynamic range of the radio signal represented. Lower values can be encountered for radio environments with low co-channel interference, thus reducing the desired dynamic range.

Denoting *n* as a certain *Baseband Coding Bits* value, the possible expressed signal values range from -2^{n-1} to $+2^{n-1}-1$.

Notion "Baseband Nominal Power"

CC_BB_SIGNAL_030 [Notion "Baseband Nominal Power"

The implicit notion *Baseband Nominal Power*, of type *Baseband Power*, **shall** be used as a representation of the power of the *Baseband Signal* measured under nominal conditions.

] END_CC_BB_SIGNAL_030

The *Baseband Signal* measured under nominal conditions is defined differently in receive and transmit cases.

The dynamic range of a *Baseband Signal* is defined as the ratio between the maximum instantaneous *Baseband Power* and the smallest positive instantaneous *Baseband Power*. Denoting *n* as a certain *Signal*

Coding Bits value, the dynamic range of a given digital representation is then equal to $20*\log(2^n) = 6.02 * n$ dB.

With a typical value for n 20 this results in a thereoretical maximum dynamic range of 120 dB, if the analogue *RF Signal* is scaled appropriately.

4.3.5 RF Signal

This chapter identifies the tuning characteristics attached to *RF Signal*.

CC_RF_SIGNAL_000 [Definition of "RF Signal"

The concept of *RF Signal* **shall** be used to refer to the RF analogue signal on the antenna either in transmission or reception.

] END_CC_BB_RF_SIGNAL_000

The characteristics attached to *RF Signal* are:

- *Carrier Frequency*
- *Nominal RF Power*

Notion "Carrier Frequency"

CC_RF_SIGNAL_010 [Notion "Carrier Frequency"

The explicit notion *Carrier Frequency*, of type *Frequency*, **shall** be used for the carrier frequency around which the *RF Signal* is centered.

] END_CC_RF_SIGNAL_010

Since the default setting for domain type *Frequency* has a maximum value around 4.2 GHz, addressing "beyond 4 GHz" radio capabilities would require introduction of supplementary settings.

Since the default setting for domain type *Frequency* has a resolution of 1 Hz, addressing "VLF" radio capabilities would require introduction of supplementary settings.

Notion "Nominal RF Power"

CC_RF_SIGNAL_020 [Notion "Nominal RF Power"

The explicit notion *Nominal RF Power*, of type *Analogue Power*, **shall** be used for the nominal power attached to the *RF Signal*.

] END_CC_RF_SIGNAL_020

The notion *Nominal RF Power* will be attached to different nature of signals dependent on the receive or transmit case.

4.3.6 Channelization

This chapter identifies the tuning characteristics attached to *Channelization*. *Channelization* denotes signal processing characterization of the treatments performed by Up- / Down-conversion chains.

CC_CHAN_000 [Definition of "Channel Transfer Function"

The internal notion of *Channel Transfer Function* **shall** designate the transfer function response of the transformation operated, by *Up-conversion Chain* between the *Baseband Signal* and *RF Signal*, and by *Down-conversion Chain* between the *RF Signal* and the *Baseband Signal*.

It shall not operate spectrum inversion.

] END_CC_CHAN_000

Note that *Channel Transfer Function* is an internal notion and not a constraint. The constraints applicable to the *Channel Transfer Function* are defined below.

Channel Mask

CC_CHAN_010 [Definition of "Channel Mask"

The concept *Channel Mask* **shall** be used to define the requirements which shall be met by the *Channel Transfer Function* of a given Conversion Chain.

] END_CC_CHAN_010

The characteristics directly attached to a *Channel Mask* are:

- *Channel Bandwidth*
- *Carrier Frequency Accuracy*

CC_CHAN_020 [Constraint "Channel Bandwidth"

The signal processing constraint *Channel Bandwidth*, of type *Frequency*, **shall** determine the bandwidth of the channel.

] END_CC_CHAN_020

CC_CHAN_030 [Constraint "Carrier Frequency Accuracy"

The signal processing constraint *Carrier Frequency Accuracy*, of type *Frequency*, **shall** be used to characterize the *absolute uncertainty* attached to the *Carrier Frequency*.

] END_CC_CHAN_030

Additional *characteristics* attached to *Channel Mask* are regrouped under the following concepts:

- *SpectrumMask*
- *GroupDelayMask*

Spectrum Mask characteristics

CC_CHAN_040 [Concept "Spectrum Mask"

The concept *Spectrum Mask* **shall** be used to characterize the spectrum mask to be satisfied by the modulus of the *Channel Transfer Function*.

] END_CC_CHAN_040

The characteristics attached to *Spectrum Mask* are:

- Ripple,
- High Bound Transition Band,
- High Bound Rejection Gain,
- High Bound Rejection Slope,
- Low Bound Transition Band,
- Low Bound Rejection Gain,
- Low Bound Rejection Slope.

The following figure illustrates those *characteristics* signification:

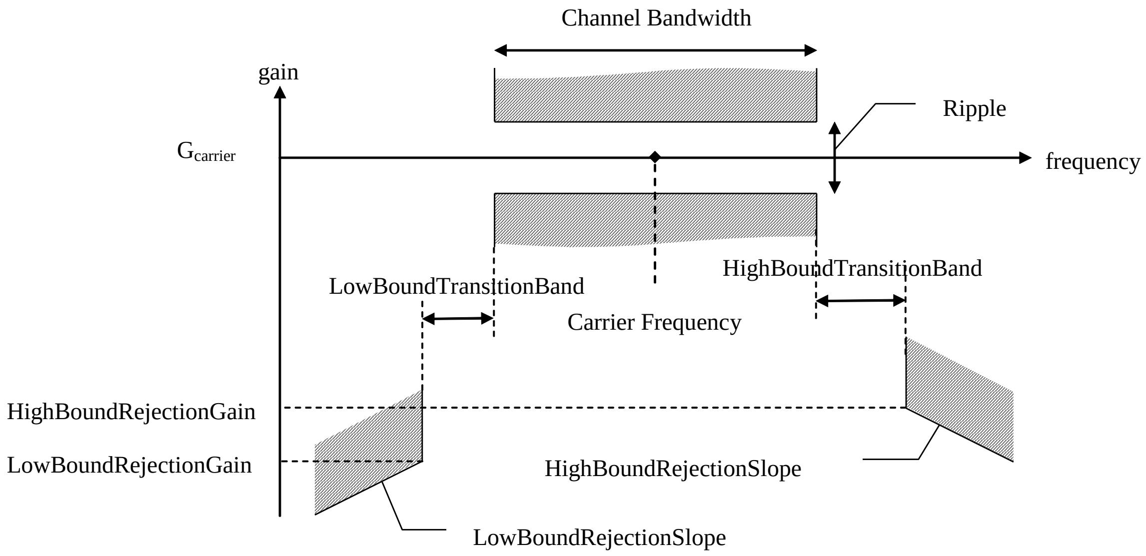


Figure 27: Characteristics of Spectrum Mask

The *Carrier Frequency* appearing on the figure corresponds to the null frequency of the baseband signal.

CC_CHAN_050 [Constraint "Ripple"

The signal processing constraint *Ripple*, of type *Gain*, **shall** be used to characterize the *absolute uncertainty* to be met by the *Channel Transfer Function* relative to the gain at *Carrier Frequency*.

] END_CC_CHAN_050

CC_CHAN_060 [Constraint "High Bound Transition Band"

The signal processing constraint *High Bound Transition Band*, of type *Frequency*, **shall** be used to characterize the frequency span from *Carrier Frequency*, equal to *Carrier Frequency* + (*Signal Bandwith*)/2 + *High Bound Transition Band*, after which a certain rejection level shall be met by *Channel Transfer Function*.

] END_CC_CHAN_060

CC_CHAN_070 [Constraint "High Bound Rejection Gain"

The signal processing constraint *High Bound Rejection Gain*, of type *Gain*, **shall** be used to characterize the minimum attenuation met by the *Channel Transfer Function* at the frequency span specified using *High Bound Transition Bound*.

] END_CC_CHAN_070

CC_ CHAN_080 [Constraint “High Bound Rejection Slope”

The signal processing constraint *High Bound Rejection Slope*, of type *GainSlope*, **shall** be used to characterize the attenuation progression to be followed by the *Channel Transfer Function* at frequency spans higher than the point specified using *High Bound Transition Band*.

] END_CC_ CHAN_080

CC_ CHAN_090 [Constraint “Low Bound Transition Band”

The signal processing constraint *Low Bound Transition Band*, of type *Frequency*, **shall** be used to characterize the frequency span from *Carrier Frequency*, equal to *Carrier Frequency* - (*Signal Bandwith*)/2 - *Low Bound Transition Band*, before which a certain rejection level shall be met by *Channel Transfer Function*.

] END_CC_ CHAN_090

CC_ CHAN_100[Constraint “Low Bound Rejection Gain”

The signal processing constraint *Low Bound Rejection Gain*, of type *Gain*, **shall** be used to characterize the minimum attenuation met by the *Channel Transfer Function* at the frequency span specified using *Low Bound Transition Band*.

] END_CC_ CHAN_100

CC_ CHAN_110[Constraint “Low Bound Rejection Slope”

The signal processing constraint *Low Bound Rejection Slope*, of type *GainSlope*, **shall** be used to characterize the attenuation progression to be followed by the *Channel Transfer Function* at frequency spans lower than the point specified using *Low Bound Transition Band*.

] END_CC_ CHAN_110

Group Delay Mask Properties

CC_ CHAN_120[Concept “Group Delay Mask”

The concept *Group Delay Mask* **shall** be used to characterize the group delay response to be satisfied by the *Channel Transfer Function*.

] END_CC_ CHAN_120

The characteristic attached to *Group Delay Mask* is the *Max Group Delay Dispersion*.

The following figure illustrates this characteristic signification:

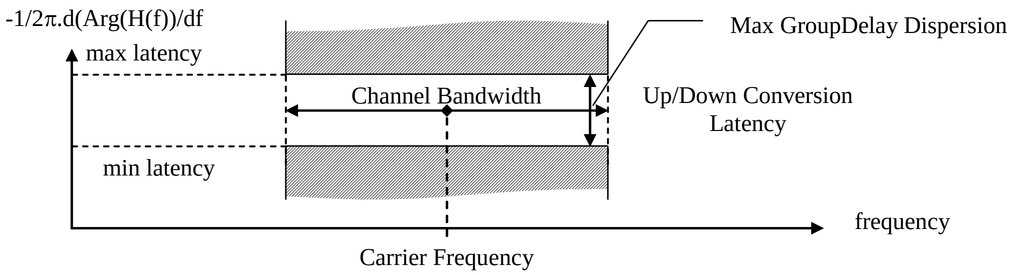


Figure 28: Characteristics of Group Delay Mask

CC_ CHAN_130 [Constraint "Max Group Delay Dispersion"

The signal processing constraint *Max Group Delay Dispersion*, of type *Latency*, **shall** be used to characterize absolute uncertainty possible tolerated from *Carrier Latency* for values of group delay of frequencies comprised within *Channel Bandwidth*.

] END_CC_BB_ CHAN_130

4.3.7 Implementation Language Requirements

4.3.7.1 C++ Implementation

Base Types

Base Types are platform dependent and can be set as follows:

```
typedef     bool            Boolean
typedef     short           Short
typedef     long            Long
typedef     unsigned short  UShort
```

```
typedef     unsigned long     ULong
```

CC_CPP_010 [BaseTypes C++ header file

Base types **shall** be defined in the header file untitled *BaseTypes.h*.

] END_CC_CPP_010

CC_CPP_020 [BaseTypes.h edition

BaseTypes.h header file **shall** be edited in consideration with the target processor, as base types depend on the processor.

] END_CC_CPP_020

Domain Types

Requirements on domain types are relative to the default setting. Use of the domain types with settings different from the default ones is not in the scope of this specification.

The default settings can be defined as follows:

```
typedef     ULong   typeFrequency
typedef     Short   typeBasebandPower
typedef     Short   typeAnaloguePower
typedef     ULong   typeLatency
typedef     Short   typeGain
typedef     Short   typeGainSlope
typedef     Short   typeIQ
```

CC_CPP_030 [DefaultSetting.h C++ header file

Implementation using default settings for domain types **shall** include the *DefaultSetting.h* header file.

] END_CC_CPP_030

CC_CPP_040 [DefaultSetting.h edition

DefaultSetting.h header file **shall** be edited in consideration with the target processor, as for *BaseTypes.h*.

] END_CC_CPP_040

The domain types are set as follow, in case of using default settings:

```
typedef      typeFrequency       Frequency;

typedef      typeGain            Gain;

typedef struct EventBasedTimeStruct
{
       UShort eventSourceId;
       enum{ Beginning, Previous, Next } eventCountOrigin;
       ULong        eventCount;
       Latency      timeShift;
}EventBasedTime;

etc…
```

Note that other definitions of domain types can be used, for example in order to use different settings for different variables.

CC_CPP_050 [Domain type "Frequency" C++ definition

The domain type *Frequency* with the associated default setting **shall** be defined as follows:

```
typedef      typeFrequency       Frequency;
```

] END_CC_CPP_050

CC_CPP_060 [Domain type "Baseband Power" C++ definition

The domain type *Baseband Power* with the associated default setting **shall** be defined as follows:

```
typedef      typeBasebandPower   BasebandPower;
```

] END_CC_CPP_060

CC_CPP_070 [Domain type "Analogue Power" C++ definition

The domain type *Analogue Power* with the associated default setting **shall** be defined as follows:

```
typedef      typeAnaloguePower   AnaloguePower;
```

] END_CC_CPP_070

CC_CPP_080 [Domain type "Latency" C++ definition

The domain type *Latency* with the associated default setting **shall** be defined as follows:

```
typedef      typeLatency         Latency;
```

] END_CC_CPP_080

CC_CPP_090 [Domain type "Gain" C++ definition

The domain type *Gain* with the associated default setting **shall** be defined as follows:

```
typedef      typeGain            Gain;
```

] END_CC_CPP_090

CC_CPP_100 [Domain type "Gain Slope" C++ definition

The domain type *Gain Slope* with the associated default setting **shall** be defined as follows:

```
typedef       typeGainSlope       GainSlope;
```

] END_CC_CPP_100

CC_CPP_110 [Domain type "Event-based Time" C++ definition

The domain type *Event-based Time* with the associated default setting **shall** be defined as follows:

```
typedef struct EventBasedTimeStruct
{
       UShort eventSourceId;
       enum{ Beginning, Previous, Next } eventCountOrigin;
       ULong        eventCount;
       Latency      timeShift;
}EventBasedTime;
```

] END_CC_CPP_110

CC_CPP_120 [Domain type "Absolute Time" C++ definition

The domain type *Absolute Time* with the associated default setting **shall** be defined as follows:

```
typedef struct AbsoluteTimeStruct
{
       ULong  secondCount;
       ULong  nanosecondCount;
}AbsoluteTime;
```

] END_CC_CPP_120

CC_CPP_130 [Domain type "Time" C++ definition

The domain type *Time* with the associated default setting **shall** be defined as follows:

```
class C_Time
{
public:
       AbsoluteTime        absolute;
       EventBasedTime      eventBased;

       // Instances of this class will be either AbsoluteTime, or
       // EventBasedTime, depending on the type of time provided
       // in the constructor
       Time(AbsoluteTime);
       Time(EventBasedTime);
       ~Time();
};
```

] END_CC_CPP_130

CC_CPP_140 [DomainTypes.h C++ header file

Implementations in C++ using domain types **shall** include the header file untitled *DomainTypes.h*, corresponding to the header file of the domain types definitions.

] END_CC_CPP_140

4.4 - Tuning Characteristics Setting

4.4.1 Introduction

The *Tuning Characteristics Setting* specifies mechanisms that enable access to *tuning characteristics*.

Tuning actors

Dependent on the nature of characteristics, *Tuning characteristics* can be set by the *Deployment*, the *Configuration* or the *Waveform Application*.

Deployment setting of a *characteristic* consists of initializing the *Transceiver Subsystem* so as to support the considered *characteristic* initial value. This initial value may be explicitly described to *Deployment* thanks to meta-data, or be hard coded with no explicit implementation.

Waveform Configuration setting of a characteristic consists in the setting the *Transceiver Subsystem* so as to support the desired *characteristic* value.

The *Waveform Application* can only set programmable characteristics (explicit notions), thanks to programming interface operations introducing a *forward* relationship between the *Waveform Application* and the *Transceiver Subsystem*.

Any implementation-level representation of a *characteristic* shall use the format specified for the considered *characteristic*.

Access mechanisms

Two sorts of *access mechanisms* are identified as access *characteristics*:

- *Elementary access*
- *Preset usage*

A given notion may exclusively be subject to one of the possibilities, and this choice is mandated by the *specification*.

These access mechanisms are not applicable on internal notions. They pertain to characteristics which are implicit and explicit notions.

Elementary access

Elementary access corresponds to individual access to a *notion*. It is used for *notions* taking values independently from others, with possible values among a presumably large range of values.

Elementary access can be realized using:

- Programming interface operations access, when the *Waveform Application* is using a Programming interface,
- *Direct access*, when *Configuration* is accessing the characteristic, through an explicit notion.

Preset usage

Preset usage corresponds to simultaneous access to a collection of *tuning characteristics*. A unique preset is defined in the specification, which characterizes a possible signal processing performance tuning for the *Conversion Chains*.

The *Deployment* initializes all the presets applicable for a given radio configuration, each of them being given a unique identifier eventually known by the *Waveform Application*.

At run-time, the *Waveform Application* or the *Configuration* is subject to modification by applicable preset.

4.4.2 Preset characteristics

CC_PRESET_010 [Definition of "Tuning Preset"

A concept *Tuning Preset* **shall** be used to identify a set of tuning characteristics values, and provides the corresponding requested values.

] END_CC_PRESET_010

CC_PRESET_020[Composition of a "Tuning Preset"

A *Tuning Preset* **shall** be composed of a unique identifier *PresetId*, of type *UShort*, followed by a list of requested characteristics values attached to the preset.

] END_CC_PRESET_020

CC_PRESET_030 [List of characteristics in a "Tuning Preset"

A *Tuning Preset* shall be composed of the following characteristics:

From *Baseband Signal*:

- Baseband Sampling Frequency
- Baseband Coding Bits
- Baseband Nominal Power

From *Channel Mask*:

- Channel Bandwidth
- Carrier Frequency Accuracy

From *Spectrum Mask*:

- Ripple
- High Bound Transition Band
- High Bound Rejection Gain
- High Bound Rejection Slope
- Low Bound Transition Band
- Low Bound Rejection Gain
- Low Bound Rejection Slope

From *Group Delay Mask*:

- Max Group Delay Dispersion

] END_CC_PRESET_030

Refer to Figure 27: Characteristics of Spectrum Mask and Figure 28: Characteristics of Group Delay Mask for a graphical representation of above constraints.

In the current version of the Facility the Preset characteristics are only accessible via the "Tuning Preset" or "Configuration".

4.4.3 Modelling Support Requirements

CC_PRESET_040 [Tuning Preset modelling elements

Transceiver Subsystem model involving concept Tuning Preset shall use the modelling elements depicted in the following class diagram:

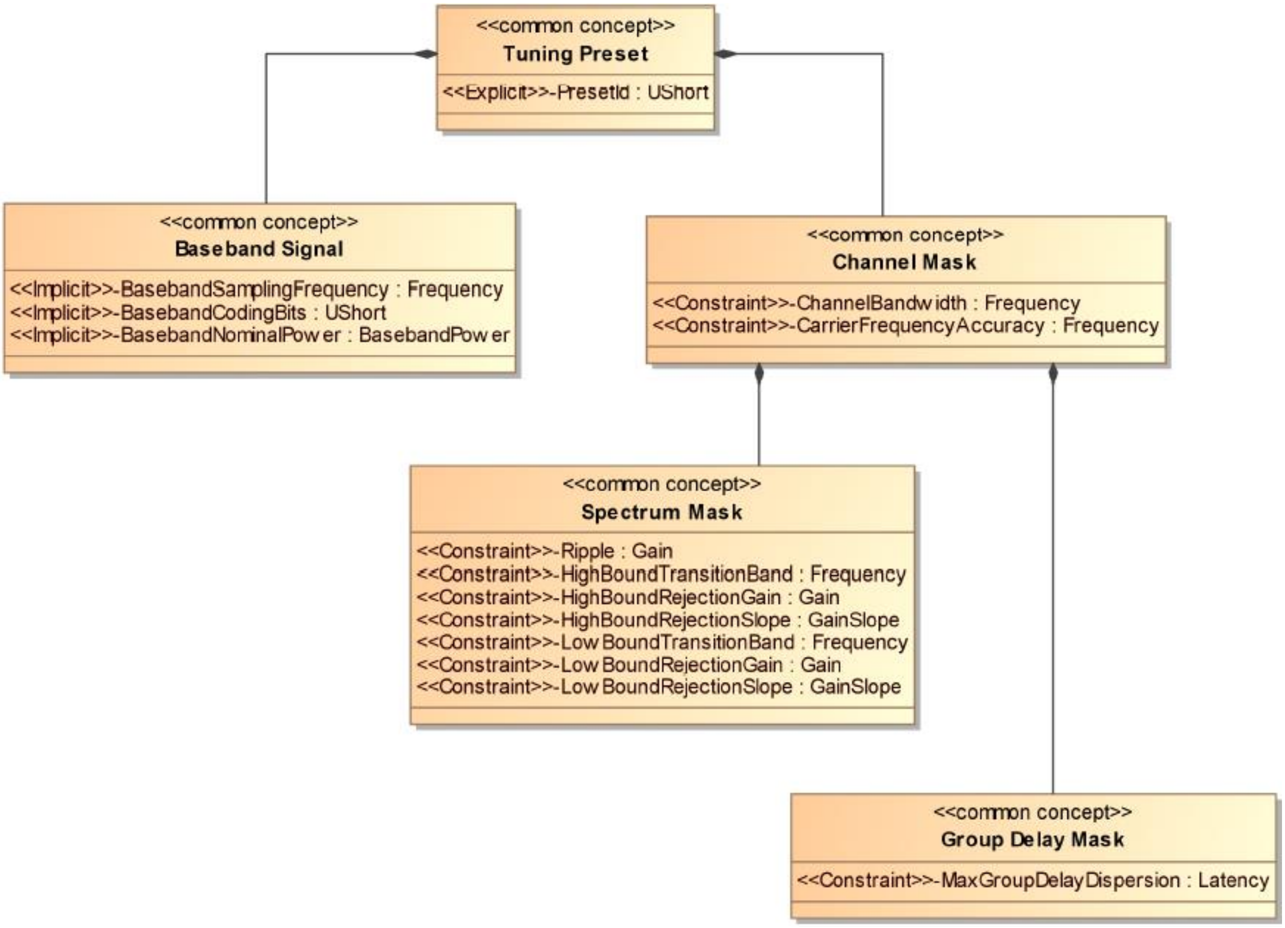


Figure 29: Tuning Preset class diagram

] END_CC_PRESET_040

4.5 - Baseband Packets

4.5.1 Introduction

Overview

The concept of *Baseband Packets* adresses the transmit or receive baseband samples interface, which are the main data interfaces exchanged between the *Waveform Application* and the *Transceiver Subsystem*.

The following *concepts* are defined by *Baseband Packets*:

- *Baseband Packet*
- *Baseband Sample*

Baseband Packets are the data structures used for signal exchange between the *Waveform Application* and the *Transceiver Subsystem*.

Baseband Samples are the elementary information items contained in *Baseband Packets*.

4.5.2 Specification

Baseband packet

Baseband Packets are the elementary pieces of information involved during effective real-time exchanges between the *Waveform Application* and the *Transceiver Subsystem*. Usage of *Baseband Packets* is specified with *operations* using them.

CC_BB_PACKET_010 [Concept "Baseband Packet"

The concept of *Baseband Packet* **shall** be used to refer to the packet of samples exchanged between the *Waveform Application* and *Transceiver Sub-system*.

A *Baseband packet* is a finite number of sequenced baseband samples.

The private attribute *Samples Number*, of type *ULong*, contains the number of samples contained in the *Baseband Packet*.

The packet is composed of a quantity of *Baseband Samples* equal to *Samples Number*.

] END_CC_BB_PACKET_010

Baseband sample

CC_BB_PACKET_020 [Concept "Baseband Sample"

The concept *Baseband Sample* **shall** represent an elementary complex sample of the *Baseband Signal*. It is composed of a *valueI* and *valueQ*, represented on an implementation language-dependent type.

] END_CC_BB_PACKET_020

The type chosen for implementation to represent the Baseband Samples components shall be capable of representing values coded on a number of bits equal to the implicit notion *BasebandCodingBits*.

4.5.3 Implementation Language Requirements

4.5.3.1 C++ Implementation

BBSample

CC _CPP_150 [BBSample C++ definition

Implementations in C++ using the Common Concept BasebandPackets **shall** use the class *BBSample* as declared in the following C++ reference source code extract:

```
typedef struct BBSampleStruct
{
     typeIQ valueI;
     typeIQ valueQ;
}BBSample;
```

] END_CC_XXX_CPP_150

CC _CPP_160 [BBSample.h C++ header file

Implementations in C++ using the Common Concept BasebandPackets **shall** include the header file untitled *BBSample.h*, corresponding to the header file of the *BBSample* definition.

] END_CC_CPP_160

BBPacket

CC _CPP_170 [BBPacket C++ definition

Implementations in C++ using the Common Concept BasebandPackets **shall** use the class *BBPacket* as declared in the following C++ reference source code extract:

```
class C_BBPacket
{
private:
	ULong		SamplesNumber;
	BBSample *	packet;

public:
	BBPacket(ULong, BBSample *);
	~BBPacket();
};
```

] END_CC_CPP_170

CC_CPP_180 [BBPacket.h C++ header file

Implementations in C++ using the Common Concept BasebandPackets **shall** include the header file untitled *BBPacket.h*, corresponding to the header file of the *BBPacket* definition.

] END_CC_CPP_180

4.5.4 List of requirements

For convenience, the following table provides an overview of the requirements specified in the chapter Common Concepts:

Identifier	Tag	Section
CC_TIME_010	Generic Domain Type "Time"	Time handling Domain Types
	Domain type "Absolute Time"	Time handling Domain Types
	Default setting for "Time Specification"	Time handling Domain Types
	Domain type "Event-based Time"	Time handling Domain Types
5	Composition of an "Event-based Time"	Time handling Domain Types
	Field "Event Source Id"	Time handling Domain Types
	Undefined "Event Source Id" value	Time handling Domain Types
	Defined field "Event Source Id"	Time handling Domain Types
	Field "Event Count Origin"	Time handling Domain Types
10	Enumerated value "Beginning"	Time handling Domain Types
	Enumerated value "Previous"	Time handling Domain Types
	Enumerated value "Next"	Time handling Domain Types
	Field "Event Count"	Time handling Domain Types
	Undefined "Event Count" value	Time handling Domain Types
15	Field "Time Shift"	Time handling Domain Types
	Domain type "Frequency"	Domain types
	Default setting for "Frequency"	Domain types
	Domain type "Baseband Power"	Domain types
	Default setting for "Baseband Power"	Domain types
20	Domain type "Analogue Power"	Domain types
	Default setting for "Analogue Power"	Domain types
	Domain type "Latency"	Domain types
	Default setting for "Latency"	Domain types
	Domain type "Gain"	Domain types
25	Default setting for "Gain"	Domain types
	Domain type "Gain Slope"	Domain types
	Default setting for "Gain Slope"	Domain types
	Definition of "Baseband Signal"	Baseband Signal
	Property "Baseband Sampling Frequency"	Baseband Signal
30	Property "Baseband Coding Bits"	Baseband Signal
	Property "Baseband Nominal Power"	Baseband Signal
	Definition of "RF Signal"	RF Signal
	Property "Carrier Frequency"	RF Signal
	Property "Nominal RF Power"	RF Signal
35	Definition of "Channel Transfer Function"	Channelization
	Definition of "Channel Mask"	Channelization
	Property "Channel Bandwidth"	Channelization
	Property "Carrier Frequency Accuracy"	Channelization
	Concept "Spectrum Mask"	Channelization
40	Property "Ripple"	Channelization
	Property "High Bound Transition Band"	Channelization
	Property "High Bound Rejection Gain"	Channelization
	Property "High Bound Rejection Slope"	Channelization
	Property "Low Bound Transition Band"	Channelization
45	Property "Low Bound Rejection Gain"	Channelization
	Property "Low Bound Rejection Slope"	Channelization
	Concept "Group Delay Mask"	Channelization
	Constraint "Max Group Delay Dispersion"	Channelization
	BaseTypes C++ header file	Tuning Properties Definition C++
50	BaseTypes.h edition	Tuning Properties Definition C++
	DefaultSetting.h C++ header file	Tuning Properties Definition C++
	DefaultSetting.h edition	Tuning Properties Definition C++
	Domain type "Frequency" C++ definition	Tuning Properties Definition C++

	Domain type “Basband Power” C++ definition	Tuning Properties Definition C++
55	Domain type “Analogue Power” C++ definition	Tuning Properties Definition C++
	Domain type “Latency” C++ definition	Tuning Properties Definition C++
	Domain type “Gain” C++ definition	Tuning Properties Definition C++
	Domain type “Event-based Time” C++ definition	Tuning Properties Definition C++
	Domain type “Absolute Time” C++ definition	Tuning Properties Definition C++
60	Domain type “Time” C++ definition	Tuning Properties Definition C++
	DomainTypes.h C++ header file	Tuning Properties Definition C++
	Definition of “Tuning Preset”	Preset Properties
	Composition of a “Tuning Preset”	Preset Properties
	List of properties in a “Tuning Preset”	Preset Properties
65	Concept “Baseband Packet”	Baseband Packets
	Concept “Baseband Sample”	Baseband Packets
	BBSample C++ definition	Baseband Packets C++
	BBSample.h C++ header file	Baseband Packets C++
	BBPacket C++ definition	Baseband Packets C++
70	BBPacket.h C++ header file	Baseband Packets C++

Table 26: Overview of Common Concepts requirements

5 - Annexes

5.1 - Using implementation standards

5.1.1 JTRS radios with MHAL RF Chain Coordinator

This figure illustrates a typical case of the defense radio industry:

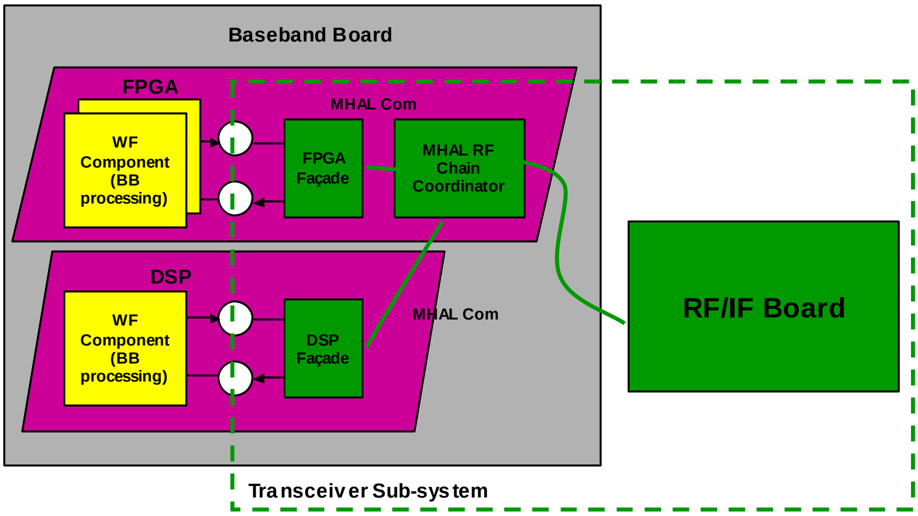


Figure 30: JTRS radios with MHAL RF Chain Coordinator

5.1.2 OBSAI/CPRI compliant Base-stations

This figure illustrates a typical case of the cellular base-stations industry:

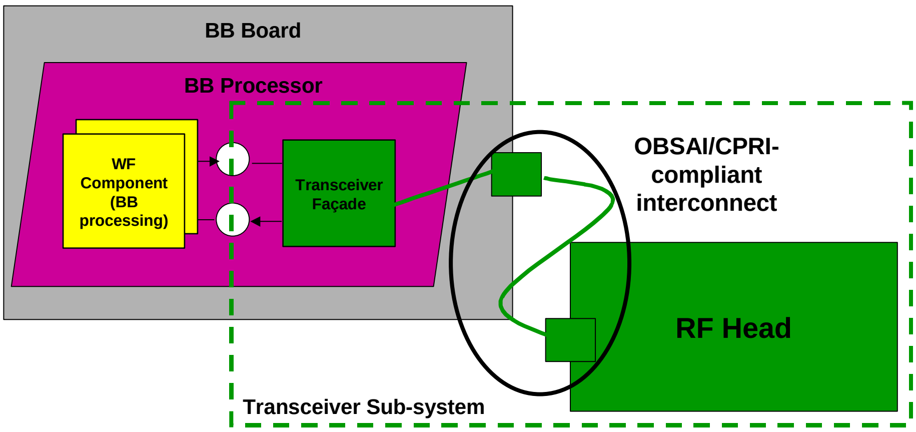


Figure 31: Transceiver Subsystem in OBSAI / CPRI compliant Base stations

5.1.3 DigRF-compliant Handsets

This figure illustrates a typical case of the cellular phones industry:

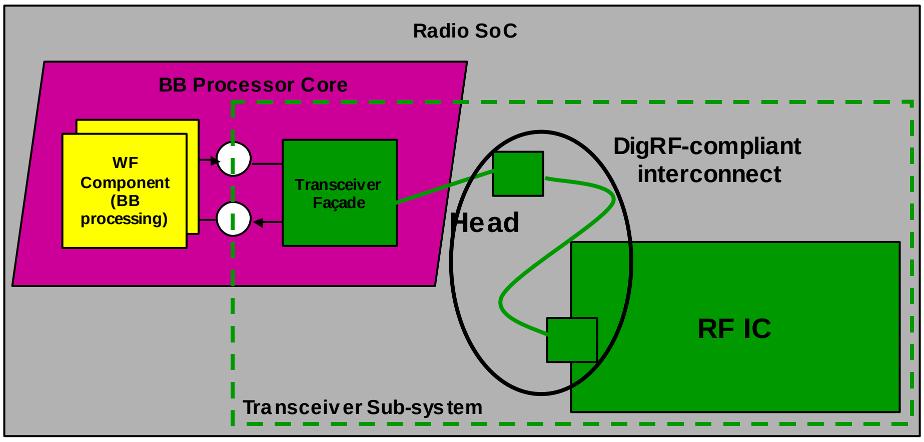


Figure 32: Transceiver Subsystem in DigRF-comploant Handsets

5.2 - Implementation Language Reference Source Code

5.2.1 C++ Implementation

Inclusion rules

The following sources are organized in header files, and correspond to the content of these header files. Note that for a C++ implementation, it is required to include header files in the following strict order:

```
#include “BaseTypes.h”
#include “DefaultSetting.h”
#include “DomainTypes.h”
```

Next, use of interfaces specified in core features requires the Common Concepts header files to be included. The extract below shows the right order of inclusion:

```
#include “BBSample.h”
#include “BBPacket.h”
```

BaseTypes.h

```
typedef     bool              Boolean
typedef     short             Short
typedef     long              Long
typedef     unsigned short    UShort
typedef     unsigned long     ULong
```

DefaultSetting.h

```
typedef      ULong   typeFrequency

typedef      Short   typeBasebandPower

typedef      Short   typeAnaloguePower

typedef      ULong   typeLatency

typedef      Short   typeGain

typedef      Short   typeGainSlope
```

```
typedef      Short  typeIQ
```

DomainTypes.h

```
// Domain type Frequency
typedef      typeFrequency       Frequency;

// Domain type BasebandPower
typedef      typeBasebandPower   BasebandPower;

// Domain type AnaloguePower
typedef      typeAnaloguePower   AnaloguePower;

// Domain type Latency
typedef      typeLatency         Latency;

// Domain type Gain
typedef      typeGain            Gain;

// Domain type Gain Slope
typedef      typeGainSlope       GainSlope;

// Domain type AbsoluteTime
typedef struct AbsoluteTimeStruct
{
      ULong  secondCount;
      ULong  nanosecondCount;
}AbsoluteTime;

// Domain type EventBasedTime
typedef struct EventBasedTimeStruct
{
      UShort eventSourceId;
      enum{ Beginning, Previous, Next } eventCountOrigin;
      ULong        eventCount;
      Latency      timeShift;
}EventBasedTime;

// Domain type Time
class C_Time
{
public:
      AbsoluteTime        absolute;
```

```
        EventBasedTime      eventBased;

        // Instances of this class will be either AbsoluteTime, or
        // EventBasedTime, depending on the kind of time provided
        // in the constructor
        Time(AbsoluteTime);
        Time(EventBasedTime);
        ~Time();
};
```

BBSample.h

```
typedef struct BBSampleStruct
{
        typeIQ valueI;
        typeIQ valueQ;
}BBSample;
```

BBPacket.h

```
class C_BBPacket
{
private:
        ULong           SamplesNumber;
        BBSample *      packet;

public:
        BBPacket(ULong, BBSample *);
        ~BBPacket();
};
```

TransmitControl.h

```
class I_TransmitControl
{
public :
        virtual void createTransmitCycleProfile(
                        Time                    requestedTransmitStartTime,
                        Time                    requestedTransmitStopTime,
                        UShort                  requestedPresetId,
                        Frequency               requestedCarrierFrequency,
                        AnaloguePower           requestedNominalRFPower) = 0;

        virtual void configureTransmitCycle(
                        ULong                   targetCycleId,
                        Time                    requestedTransmitStartTime,
                        Time                    requestedTransmitStopTime,
                        Frequency               requestedCarrierFrequency,
                        AnaloguePower           requestedNominalRFPower) = 0;

        virtual void setTransmitStopTime(
                        ULong                   targetCycleId,
                        Time                    requestedTransmitStopTime) = 0;
};
```

TransmitDataPush.h

```
class I_TransmitDataPush
{
public :
	virtual void pushBBSamplesTx(
			BBPacket * thePushedPacket,
			Boolean endOfBurst) = 0;
};
```

ReceiveControl.h

```
class I_ReceiveControl
{
public :
	virtual void createReceiveCycleProfile(
			Time			requestedReceiveStartTime,
			Time			requestedReceiveStopTime,
			ULong			requestedPacketSize,
			UShort			requestedPresetId,
			Frequency		requestedCarrierFrequency) = 0;

	virtual void configureReceiveCycle(
			ULong			targetCycleId,
			Time			requestedReceiveStartTime,
			Time			requestedReceiveStopTime,
			ULong			requestedPacketSize,
			Frequency		requestedCarrierFrequency) = 0;

	virtual void setReceiveStopTime(
			ULong			targetCycleId,
			Time			requestedTransmitStopTime) = 0;
};
```

ReceiveDataPush.h

```
class I_ReceiveDataPush
{
public :
	virtual void pushBBSamplesRx(
			BBPacket * thePushedPacket,
			Boolean endOfBurst) = 0;
};
```