

European SDR for wireless in joint security operations

# Transceiver Facility v1.0 Implementation In the EULER project Framework



# Overview

1. FP7-SEC Project EULER context
2. The Transceiver API role within EULER
3. Main Issues in v1.0 from an implementation stand point
4. Assumptions made for implementation in EULER
5. Implementation Description

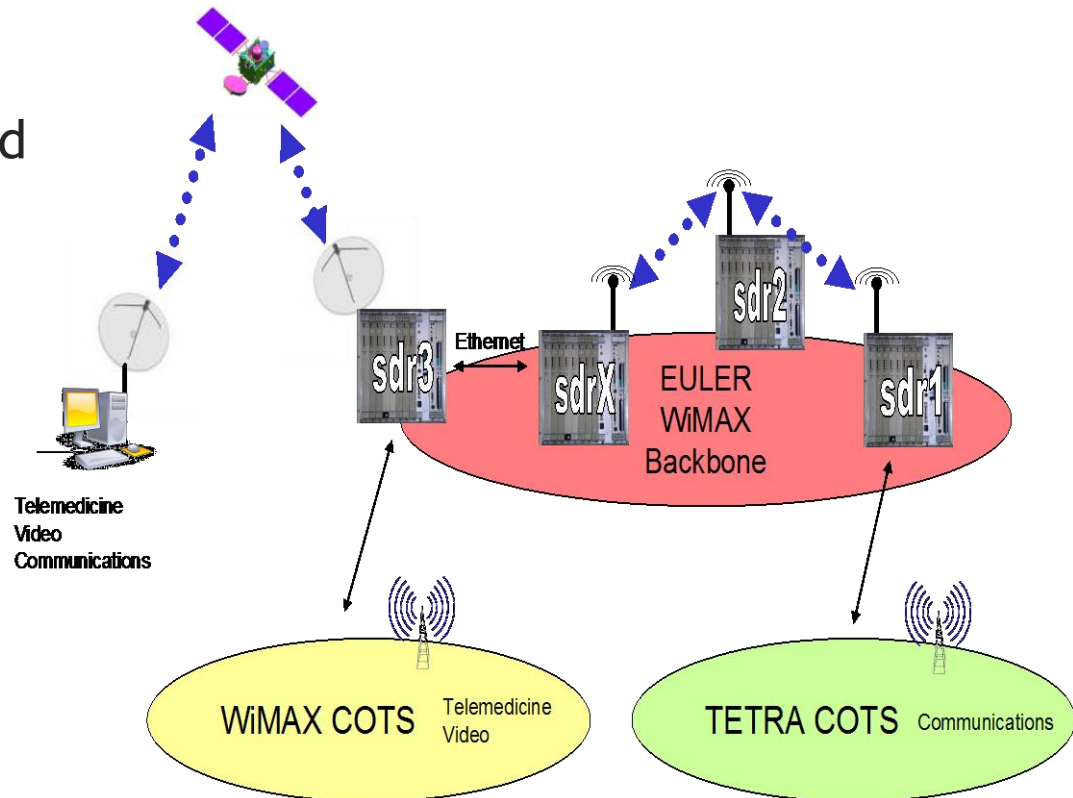
# FP7-SEC EULER Project - 1 / 2

- EULER is an European funded research project under the 7th Framework Programme, addressing the Theme 10-Securities.
- It is an SDR focused project, encompassing up to 18 European partners (Industrial leaders, SMEs and Universities)
- Its main Goal is to demonstrate the benefits of SDR for public safety scenarios considering international crisis situations:
  - « *law enforcement organizations, rescue services, military, converge and share the same deployment area while using its own communication equipment and radio standards* »

# FP7-SEC EULER Project - 2/2

- EULER is developing a new waveform based on IEEE802.16e-2005
- The WF is basically divided into two SCA resources: MAC(SS) and PHY
- This WF will be ported on two different Platforms: Thales & Prismtech.
- A Final demonstration is planned in order to showcase the WF and the SDR portability.

## Final Demonstration Outlook



## The Transceiver API role within EULER 1 / 3

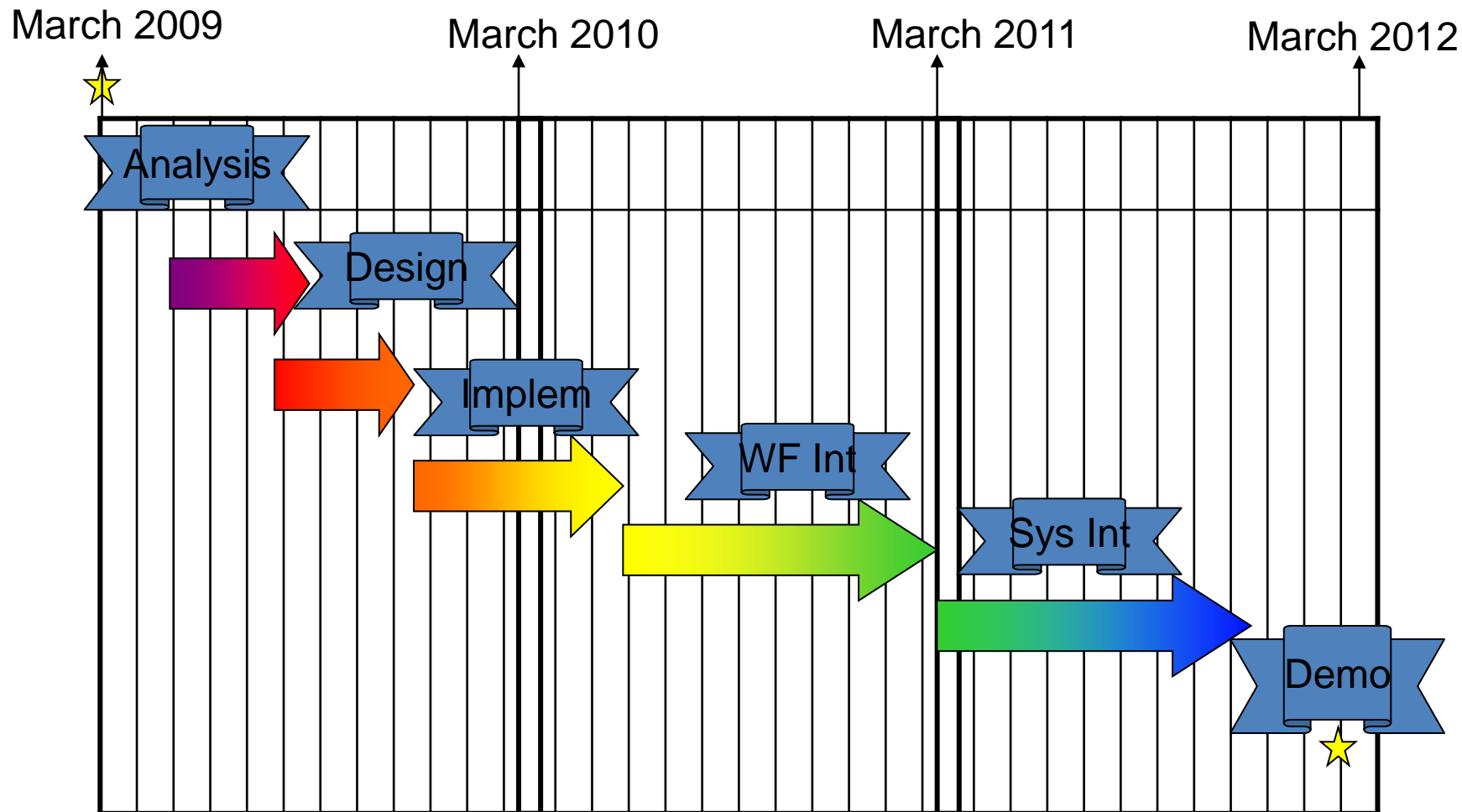
- The Transceiver API (in its v1.0 version) has been selected as the interface between the WF and the Platforms, therefore it defines the boundary between those and enables the separated PF/WF providers business model.
- Two Platforms shall then implement a Transceiver compliant interface on top of which the EULER WF will be ported.
- Since one of the main goals of EULER is to demonstrate portability for an SDR WF, this makes the Transceiver API a fundamental pillar for the success of the project.

# The Transceiver API role within EULER 2/3

## Consortium working scheme

- In order to achieve the implementation of a Transceiver Specification compliant API into two distinct platforms, the consortium working methodology is as follows:
  - **Starting Point:** Transceiver Facility Specification v1.0 shared among the partners (essentially the WF providers and PF providers)
  - **Phase 1:** “Analysis Phase” Narrow down the specification in front of the WF requirements (Scenarios, Use Cases analysis). Output: list of ambiguous points, implementation issues (**Use Cases and Constraints document**)
  - **Phase 2:** “Design Phase” Decisions on implementation, assumptions made, specification “workarounds”. Output: **Definition of an IDL** as the basis for partners discussions.
  - **Phase 3:** “Implementation Phase” Implementation on the two platforms

# The Transceiver API role within EULER Consortium working scheme 3/3



## Summary of issues found in v1.0 when addressing implementation

- *Undefined* and *Immediate* ambiguity in the specification, which led us to arbitrarily set our own definitions and slightly modify the Time management modes proposed.
- The ambiguous identification of the events sources and the events occurrences used for Transmit and Receive cycle creation (the main drawback of the v1.0 from the implementation team perspective)
- Data storing and buffering (inout BBpacket argument)

# Time management mechanisms related issues 1/11

## *Undefined and Immediate ambiguity*

- The v1.0 states in requirements FEAT\_TX\_CHAN\_ANA\_770 and FEAT\_RX\_CHAN\_ANA\_710 that the Transmit/Receive stop time may be let “undefined” and this shall be interpreted as a request for an immediate or “asap” transmission/reception stop. Nothing is stated about the use of an “undefined” mode for the start time.

# Time management mechanisms related issues 2/11

## *Undefined and Immediate ambiguity*

- These statements appear as confusing ones from an implementation stand point:
  - “undefined” could then be understood as “Immediate”, but then how to request for a true undefined stop time, for example for a synchronization procedure?
  - What happens if we use “undefined” for start times, does it mean “start as soon as you can”?,
  - what about the use of “undefined” in other operations than `setReceive/TransmitStopTime()`
- Additionally how to implement this “undefined” in a programming language?. Absolute and event Based modes easily are identified as two different types of time, but what about undefined?

# Time management mechanisms related issues 3/11

## *Undefined and Immediate ambiguity*

- **The Specification is unclear**, it seems that two different behaviors “immediate” and “unknown” are mixed in the word “undefined”.
- In EULER, decision was made to complete the types of Time by adding two new categories:
  - Immediate Time: For requesting an immediate action, start/stop transmission/reception.
  - Undefined: Typically used to let the stop time of a Transmission/reception open for further request and enable thus synchronization procedures that could need a listening window of “a priori” unknown size.
- Besides, those types are valid for any operation and can therefore be easily expressed in any programming language by IDL enums.

# Time management mechanisms related issues 4/11

## Recall of Time Management Mechanisms

- The v1.0 proposes two time management mechanisms:
  - Absolute Time
  - Event Based Time
- Absolute Time is used for requesting a Transmit/Receive at a well defined target time.
- Event Based Time makes use of a set of parameters to indicate a target time which depends on events occurrences.

# Time management mechanisms related issues 5/11

## The problem of event Sources definition

- Event Based Time management uses a complete set of parameters to identify events and provide target times for performing operations based on those.
- The basic concept for Event Based Time are the *Event Sources*
- During the EULER implementation what actually is a *Event Source* was a little bit unclear when reading the specification.
  - *In an example in page 33 up to four event sources are mentioned: RF Transmit Start, Stop, RF Receive Start, Stop.*
  - *FEAT\_TX\_CHAN\_ANA\_350 states that the “TransmitStart” is the unique valid event reference source for Transmit Channel, FEAT\_RX\_CHAN\_ANA\_330 makes the same statement but for Receive Channel where only the “ReceiveStart” event source would be allowed.*
  - *Similarly, FEAT\_TX\_CHAN\_ANA\_640 and FEAT\_RX\_CHAN\_ANA\_570 speak about “default” event sources*

# Time management mechanisms related issues 6/11

## The problem of event Sources definition

- Despite the confusing sentences, the following understanding may be inferred as a requirement:
  - *Only TransmitStartTime and ReceiveStartTime are valid event sources.*
  - *No cross-references are possible. Transmit Channel is bind to the TransmitStartTime and Receive Channel is bind to the ReceiveStopTime. Thus Only one event source is possible!*
- These conclusions difficult implementation for EULER waveform.
  - Outlined Transceiver use cases for the WF require cross references. Typically the synchronization procedure that needs the “ReceiveStartTime” as a reference for the TDD frame next Transmission.

## Time management mechanisms related issues 7/11

### The problem of event Sources definition

- Thus in EULER, we decided to define 4 event sources:
  - ReceiveStartTime, ReceiveStopTime, TransmitStartTime, TransmitStopTime.
  - They may be use by any operation, no matter whether it is a Transmit or Receive Channel operation
  - Their event occurrences are well know by the Transceiver Platform.
  - They are shared with the waveform application by means of an identifier.
  - Any potential waveform to be loaded to the platform and that intends to use Event Based Time may only refer to these four since those are actually the only implemented.

# Time management mechanisms related issues 8/11

## The problem of Events identification

- Another critical issue we faced during Transceiver design was the events identification. The Specification is quite rich in options regarding this :
  - *There is an event Count Origin parameter allowing for beginning, previous, next events identification*
  - *There is an event Count allowing for counting events*
  - *There is a time shift to add to the event for computing a target time for an action*
- Thus it is critical to identify the event that is the reference, count its occurrences if requested by the event Count parameter and finally apply the requested time shift.
- Event Count origin is then fundamental.

# Time management mechanisms related issues 9/11

## The problem of Events identification

- Beginning, Previous and Next semantics appear quite clear, but how the Waveform knows that an event happened so it can use the right value?
  - In the current specification there is no synchronization mechanism between Waveform and Platform Transceiver.
  - The Waveform is unable to know when events happened, only the transceiver is aware of this information!
  - Since the transport mechanism between Waveform and Transceiver is unknown the time between invocation and its arrival may vary, and the events happen (or not) so Previous and Next values could then be wrong!
- In our opinion this is an important drawback...

# Time management mechanisms related issues 10/11

## The problem of Events identification

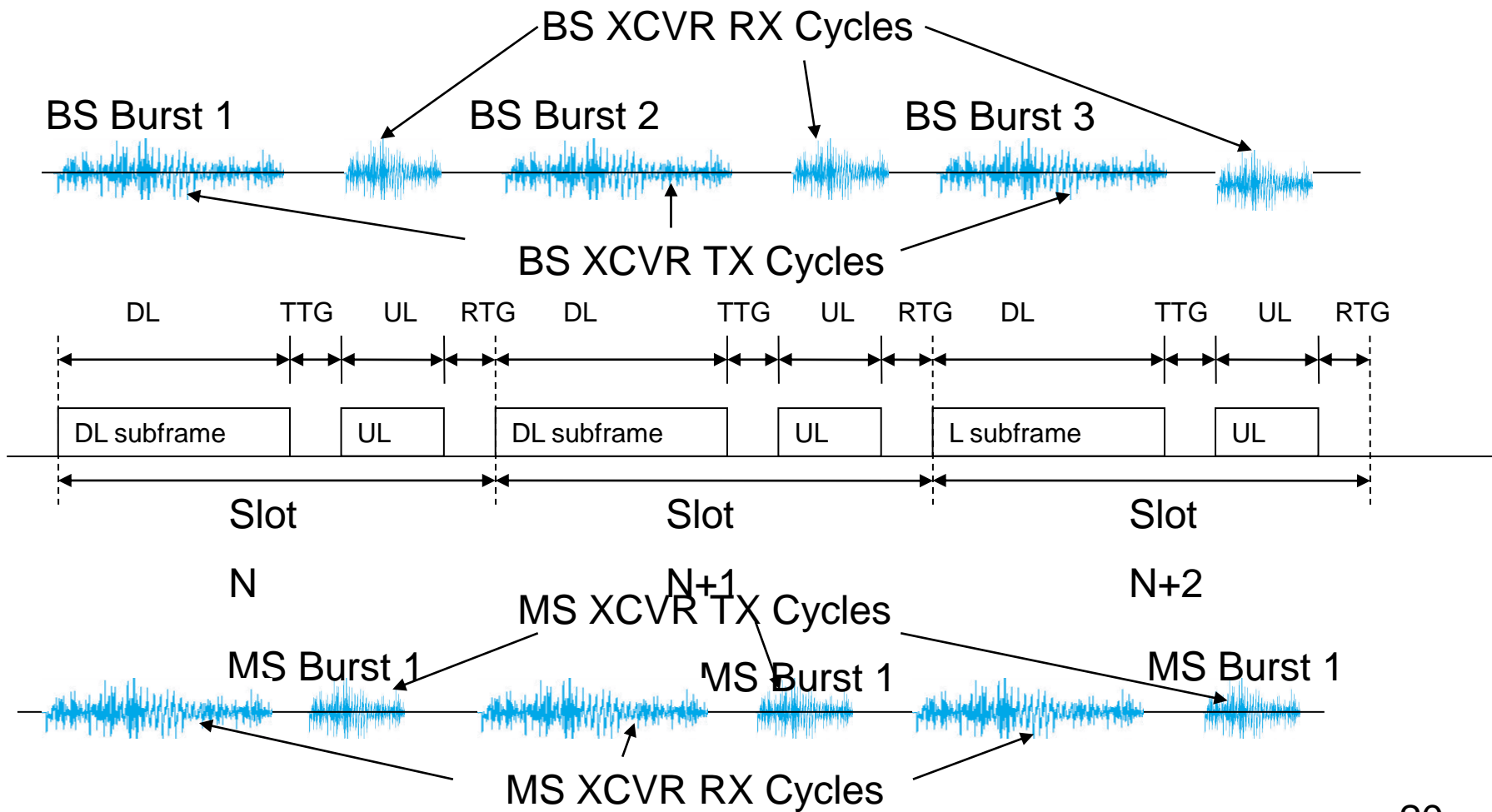
- EULER solution is a workaround that takes advantage of the EULER waveform: a TDD waveform with alternate Transmit and Receive.
- The basic idea is simple: Use as much as possible the sole event source whose events occurrences may be deducted by the Waveform:
  - The ReceiveStartTime: when a reception takes place, the waveform will receive samples (pushBBSamplesRx() arrival), then it is aware of the ReceiveStartTime occurrence and can use the Previous value in a secured way.
  - The previous principle, together with assumption that no new receiveStartTime will happen until the next frame (5 ms later) gives the Waveform enough time to program new TX/RX cycles (actually two) before the next occurrence. The procedure is iterated.

# Time management mechanisms related issues 11/11

## The problem of Events identification

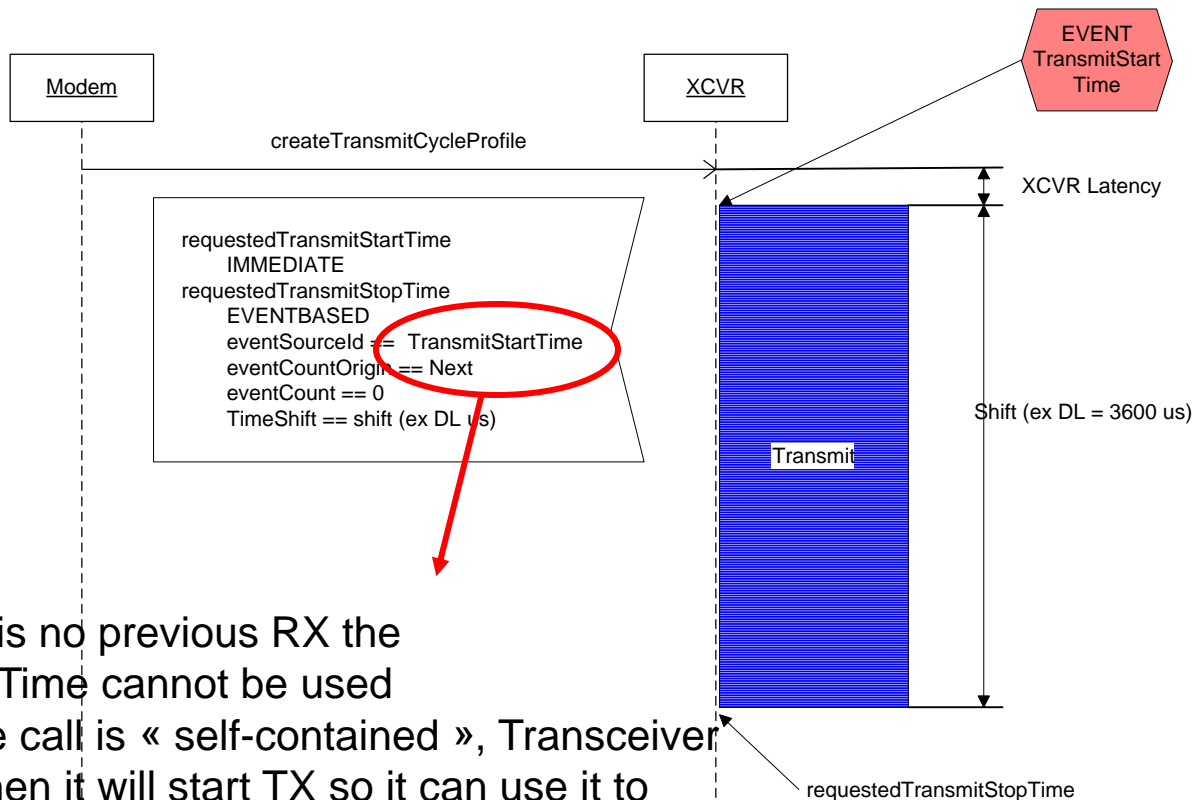
- Some additional thoughts:
  - The previously described strategy is valid provided that a reception took place. A number of use cases do not fulfill that and other solutions are implemented for those (next slides).
  - The approach makes use of the *previous* value alone. WF refer to the event only when it knows that it happened: *next* is then no useful (beginning could be, but it is discarded to avoid tracking time from the very beginning)
  - The accurate occurrence time of the event is not known by the WF. This works basically because the EULER waveform doesn't need it (only frame accuracy).
- The next slides describe one by one the use cases that a TDD waveform such as the EULER one needs. In each slide the parameters values are detailed

# EULER waveform



# Time management mechanisms related issues

## EULER waveform Use Case 1: Base Station First Transmission

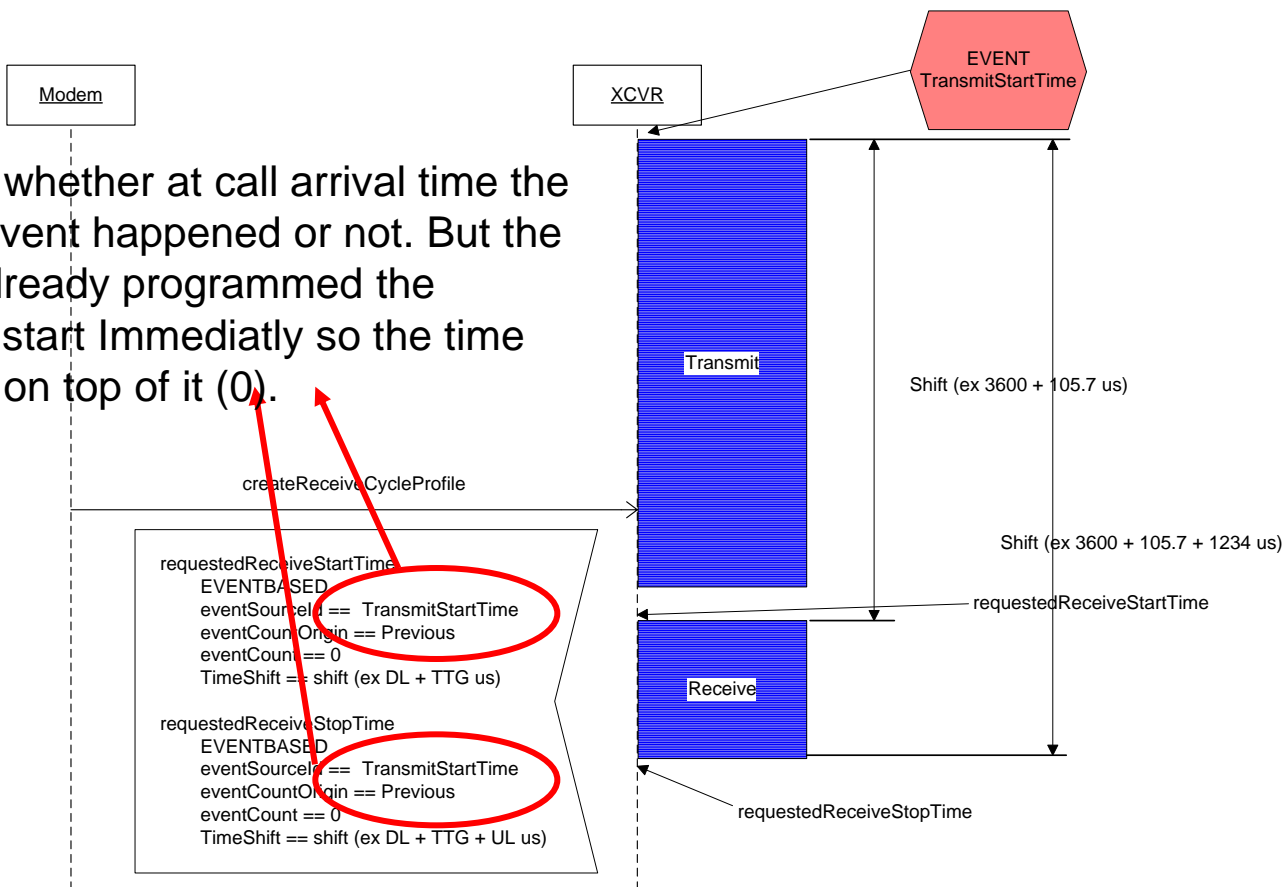


Since there is no previous RX the receiveStartTime cannot be used  
 However the call is « self-contained », Transceiver will know when it will start TX so it can use it to compute the TX stop time (thus no sync is needed between PF and WF).

# Time management mechanisms related issues

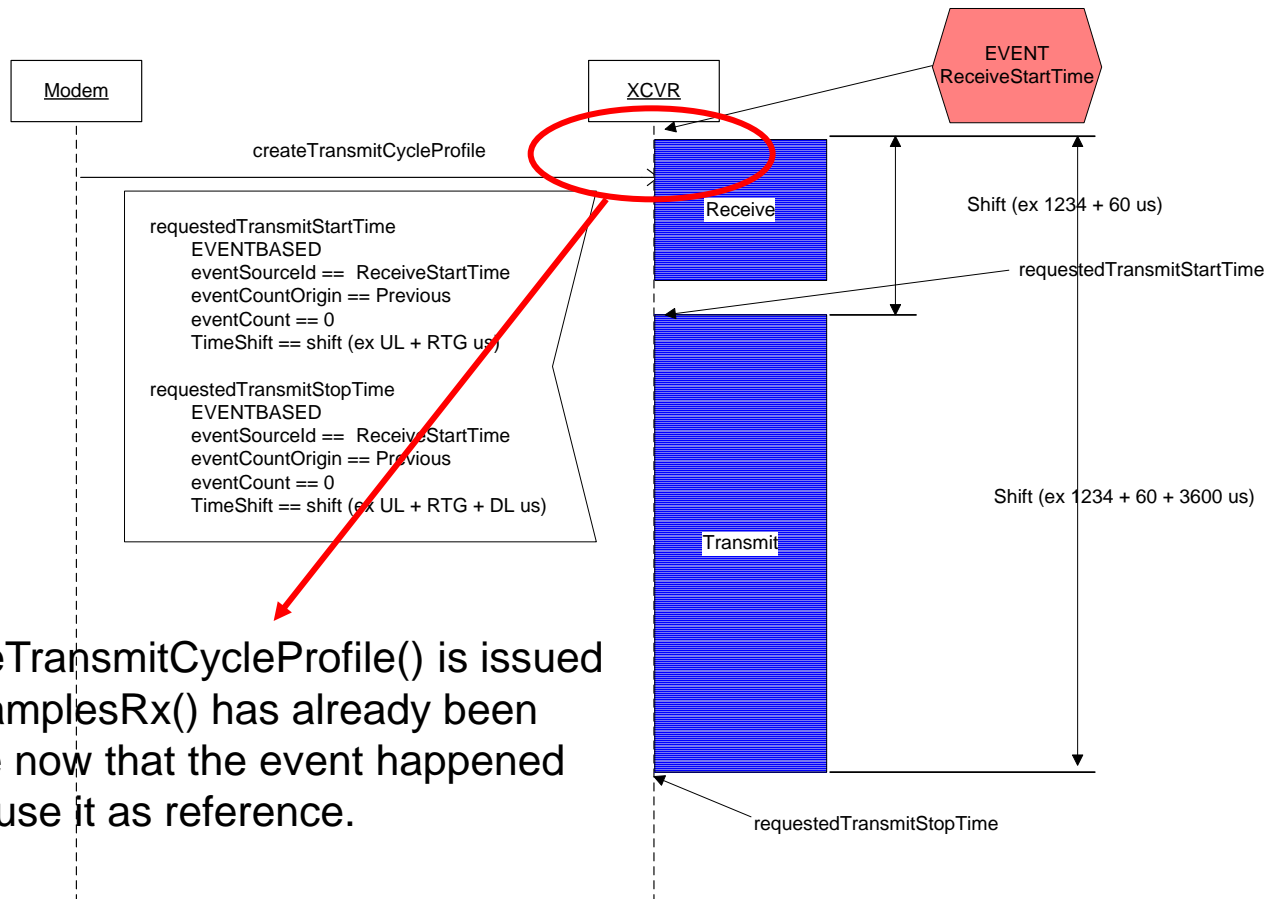
## EULER waveform Use Case 2: Base Station First Reception

We don't know whether at call arrival time the transmitStart event happened or not. But the previous call already programmed the Transceiver to start immediately so the time shift is applied on top of it (0).



# Time management mechanisms related issues

## EULER waveform Use Case 3: Base Station second and further Transmissions

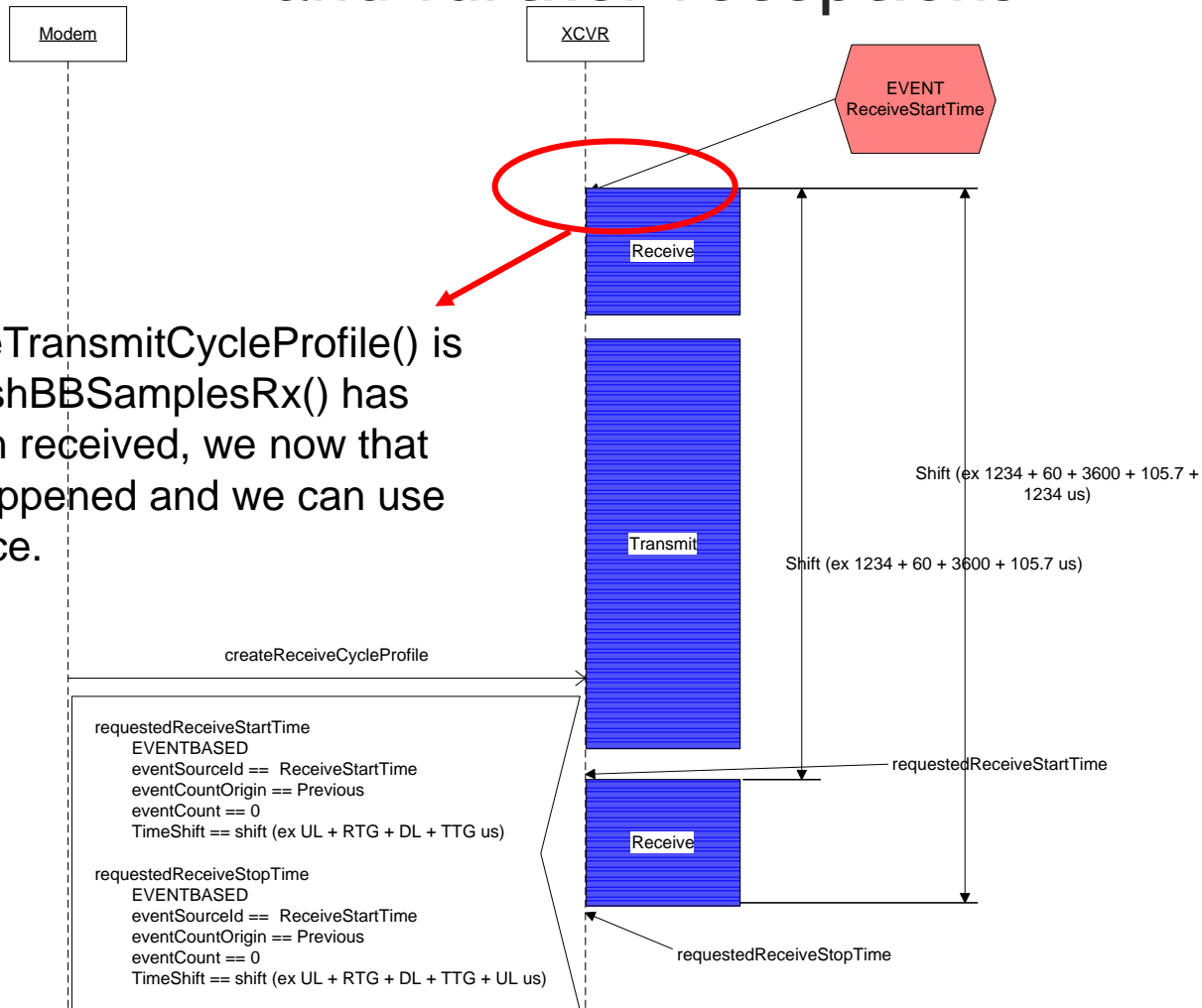


When createTransmitCycleProfile() is issued a PushBBSamplesRx() has already been received, we now that the event happened and we can use it as reference.

# Time management mechanisms related issues

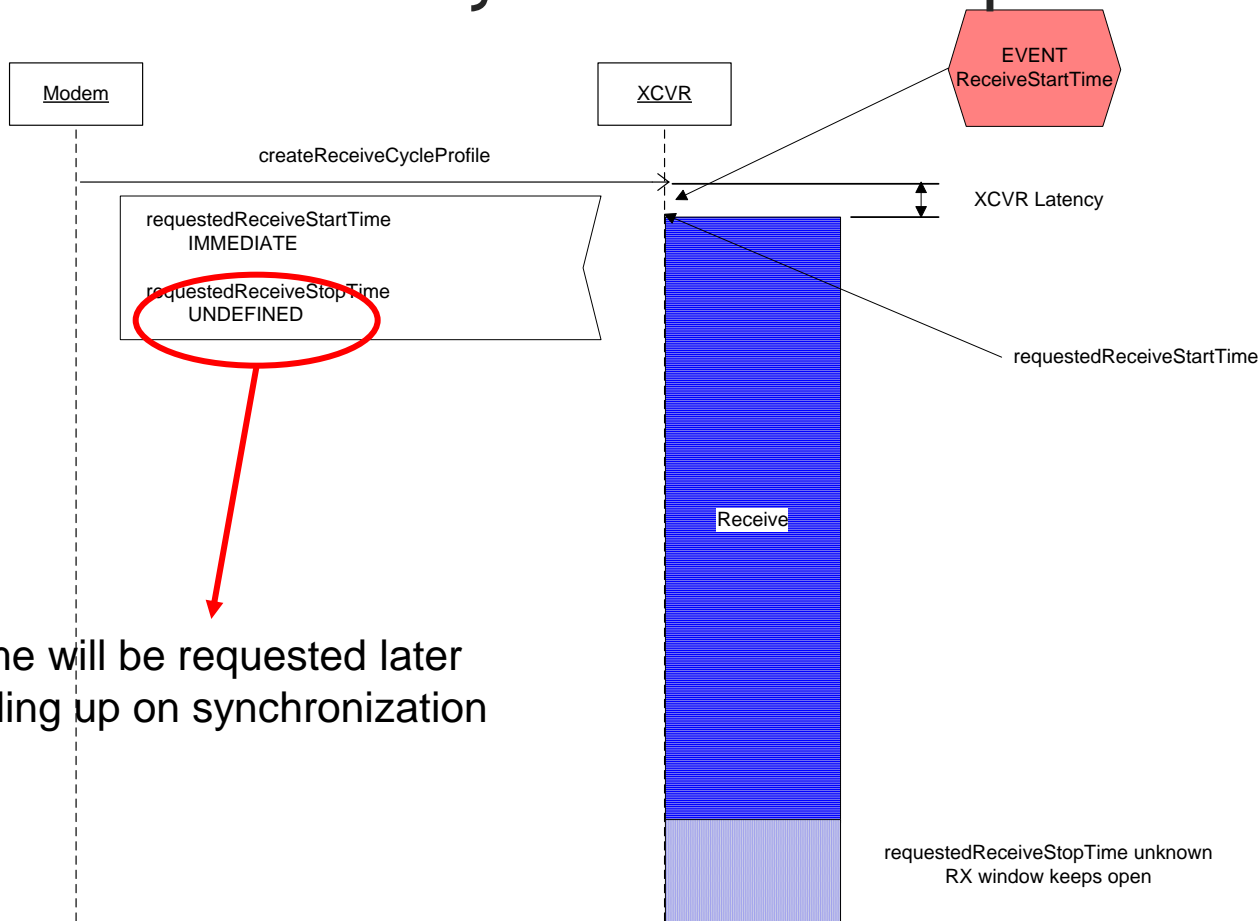
## EULER waveform Use Case 4: Base Station second and further receptions

When createTransmitCycleProfile() is issued a PushBBSamplesRx() has already been received, we now that the event happened and we can use it as reference.



# Time management mechanisms related issues

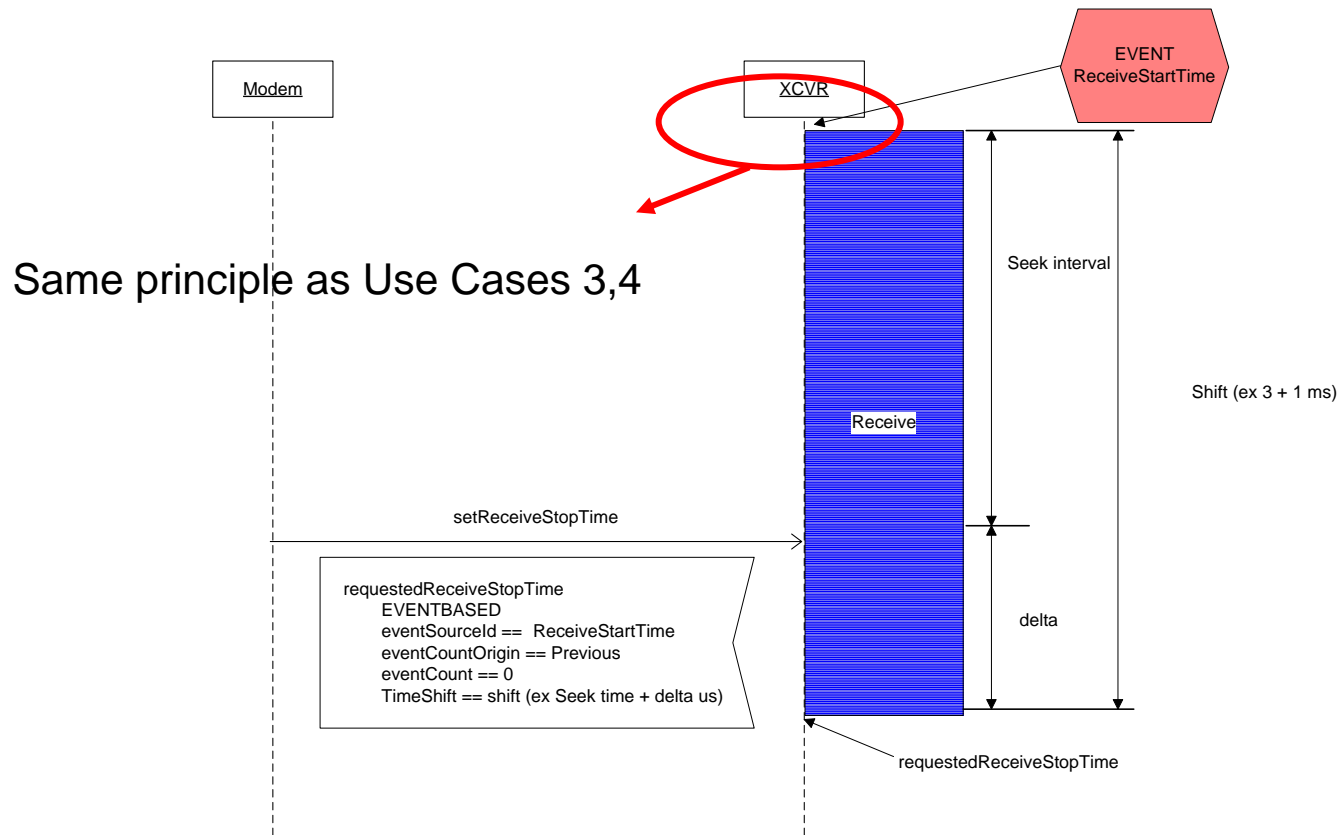
## EULER waveform Use Case 5: Mobile Station First Receive for synchronization procedure



RX stop time will be requested later on, depending up on synchronization duration

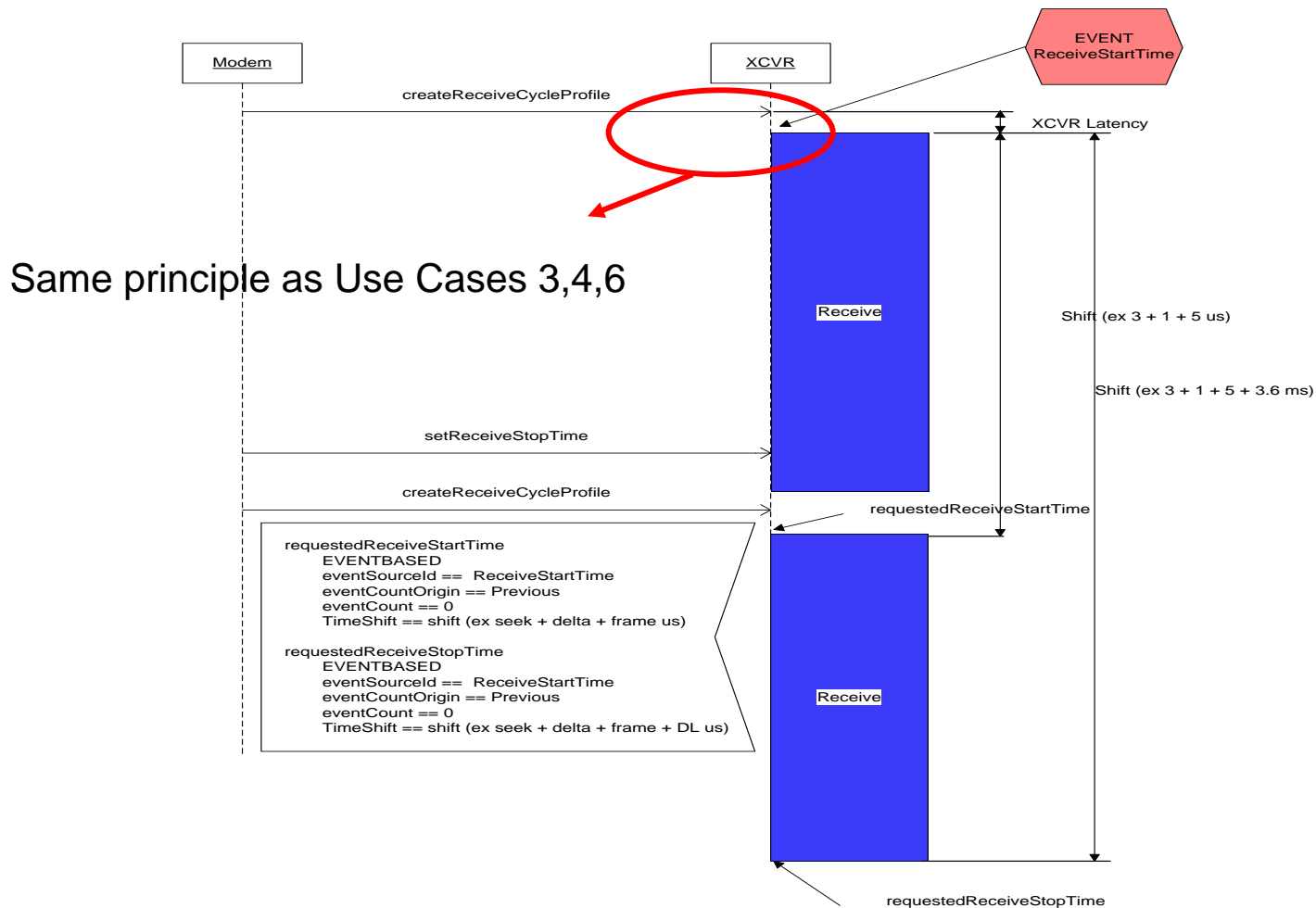
# Time management mechanisms related issues

## EULER waveform Use Case 6: Mobile Station Stop Receive for stop synchronization procedure



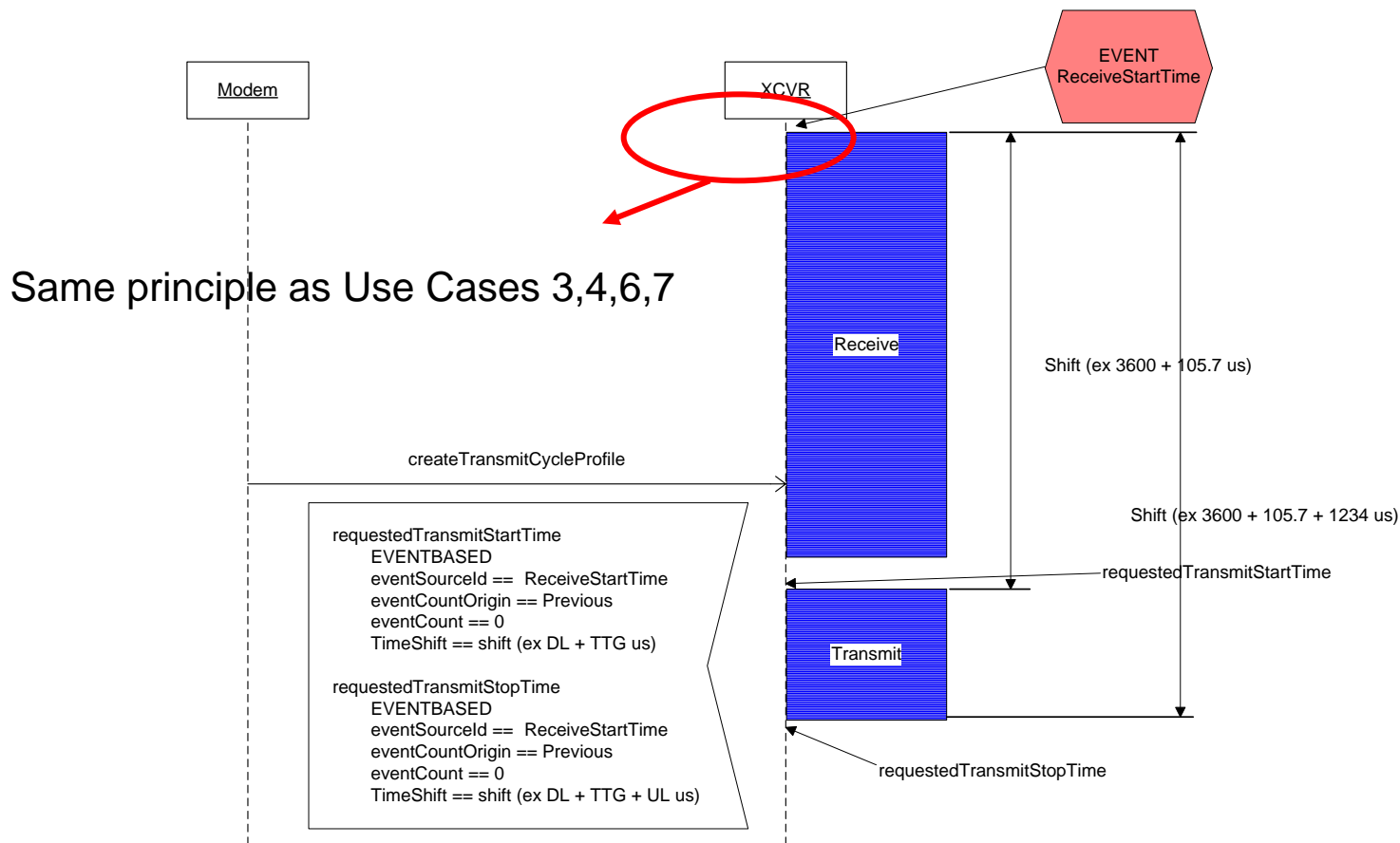
# Time management mechanisms related issues

## EULER waveform Use Case 7: Mobile Station First Receive after synchronization



# Time management mechanisms related issues

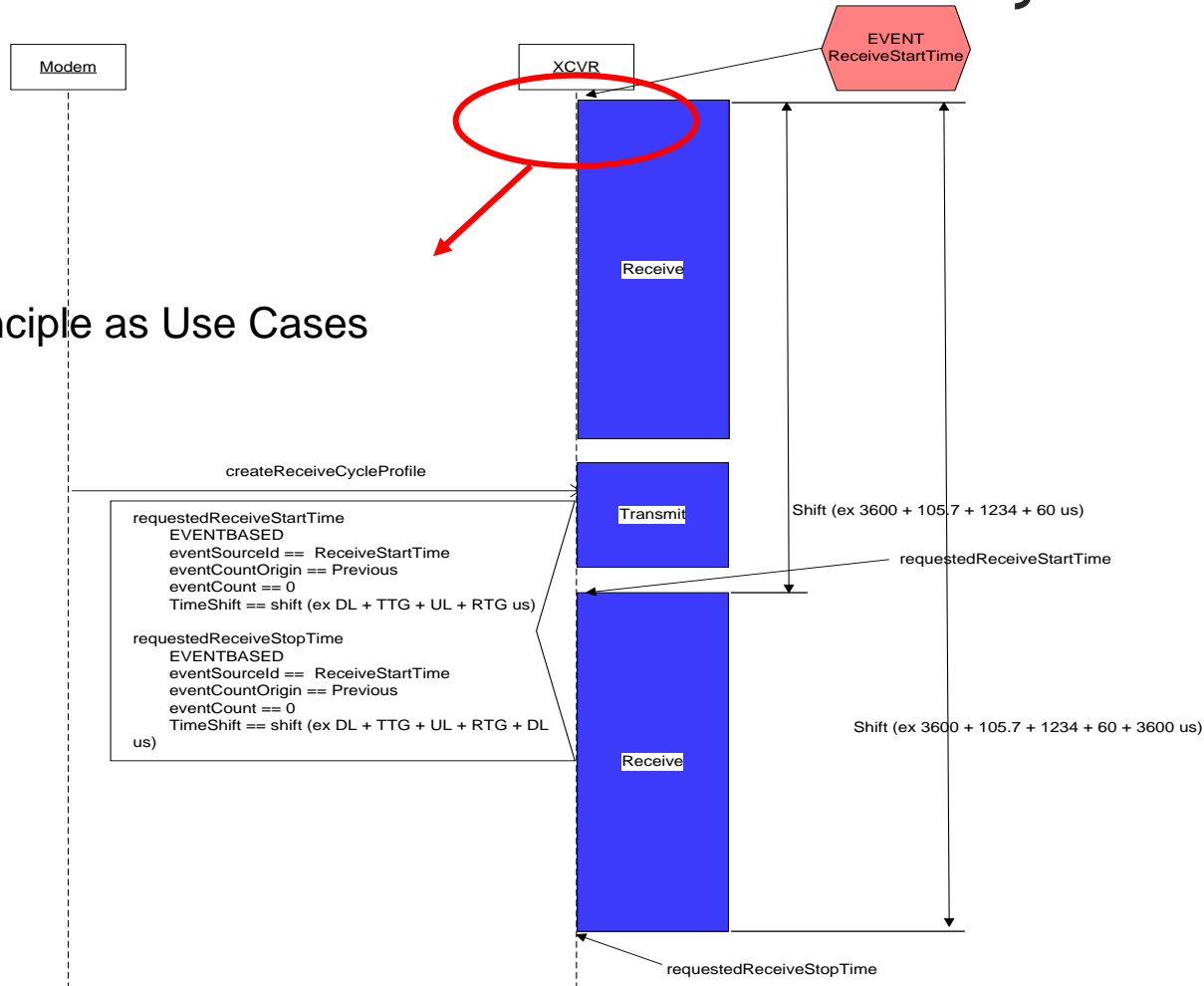
## EULER waveform Use Case 8: Mobile Station Transmissions when synchronized



# Time management mechanisms related issues

## EULER waveform Use Case 9: Mobile Station second and further receive when synchronized

Same principle as Use Cases 3,4,6,7,8



# The EULER Transceiver IDL

## Assumptions made

1. Synchronous (two-way) operations with an associated execution bounded time requirement: 20us to transfer 4.5 KB
  2. `pushBBSamplesTx/Rx()` methods with `inout` packet parameter, to allow `getbuffer` implementation option.
  3. Four time management options supported
  4. `BBPacket` `samplesnumber` removed, it is now straightforward expressed as a sequence
- Following these assumptions a Transceiver Interface IDL was worked out (since the specification was lacking one)

# The EULER Transceiver IDL 1 / 7

```
struct AbsoluteTime
{
    unsigned long secondCount;
    unsigned long nanoSecondCount;
};

enum EventOrigin
{
    Beginning,
    Previous,
    Next
};
```

# The EULER Transceiver IDL 2/7

```
enum EventSourceId
{
    ReceiveStartTime,
    ReceiveStopTime,
    TransmitStartTime,
    TransmitStopTime
};

struct EventBasedTime
{
    unsigned short eventSourceId;
    EventOrigin    eventCountOrigin;
    unsigned long  eventCount;
    Latency        timeShift;
};
```

# The EULER Transceiver IDL 3/7

```
enum TimeDiscriminatorType
{
    Immediate,
    Undefined,
    Absolute,
    Event
};

union Time switch (TimeDiscriminatorType)
{
    case Absolute:
        AbsoluteTime absoluteDate;
    case Event:
        EventBasedTime eventBased;
};
```

# The EULER Transceiver IDL 4/7

```
typedef short TypeIq;
struct BBSample
{
    TypeIq valueI;
    TypeIq valueQ;
};
typedef sequence <BBSample> BBSampleSeq;
typedef BBSampleSeq BBPacket;
typedef unsigned long Frequency;
typedef short AnaloguePower;
typedef unsigned long Latency;
const unsigned long MaxPushedPacketSize = 28000;
```

# The EULER Transceiver IDL 5/7

```
interface TransmitControl
{
    void createTransmitCycleProfile (
        in Time          requestedTransmitStartTime,
        in Time          requestedTransmitStopTime,
        in unsigned short requestedPresetId,
        in Frequency     requestedCarrierFrequency,
        in AnaloguePower requestedNominalRFPower
    );

    void configureTransmitCycle(
        in unsigned long  targetCycleId,
        in Time          requestedTransmitStartTime,
        in Time          requestedTransmitStopTime,
        in Frequency     requestedCarrierFrequency,
        in AnaloguePower requestedNominalRFPower
    );
    void setTransmitStopTime (
        in unsigned long targetCycleId,
        in Time          requestedTransmitStopTime
    );
};
```

# The EULER Transceiver IDL 6/7

```
interface ReceiveControl
{
    void createReceiveCycleProfile (
        in Time            requestedReceiveStartTime,
        in Time            requestedReceiveStopTime,
        in unsigned long   requestedPacketSize,
        in unsigned short  requestedPresetId,
        in Frequency       requestedCarrierFrequency
    );
    void configureReceiveCycle (
        in unsigned long   targetCycleId,
        in Time            requestedReceiveStartTime,
        in Time            requestedReceiveStopTime,
        in unsigned long   requestedPacketSize,
        in Frequency       requestedCarrierFrequency
    );
    void setReceiveStopTime (
        in unsigned long   targetCycleId,
        in Time            requestedReceiveStopTime
    );
};
```

# The EULER Transceiver IDL 7/7

```
interface TransmitDataPush
{
    void pushBBSamplesTx (
        inout BBPacket thePushedPacket,
        in boolean endOfBurst
    );
};
```

```
interface ReceiveDataPush
{
    void pushBBSamplesRx (
        inout BBPacket thePushedPacket,
        in boolean endOfBurst
    );
};
```

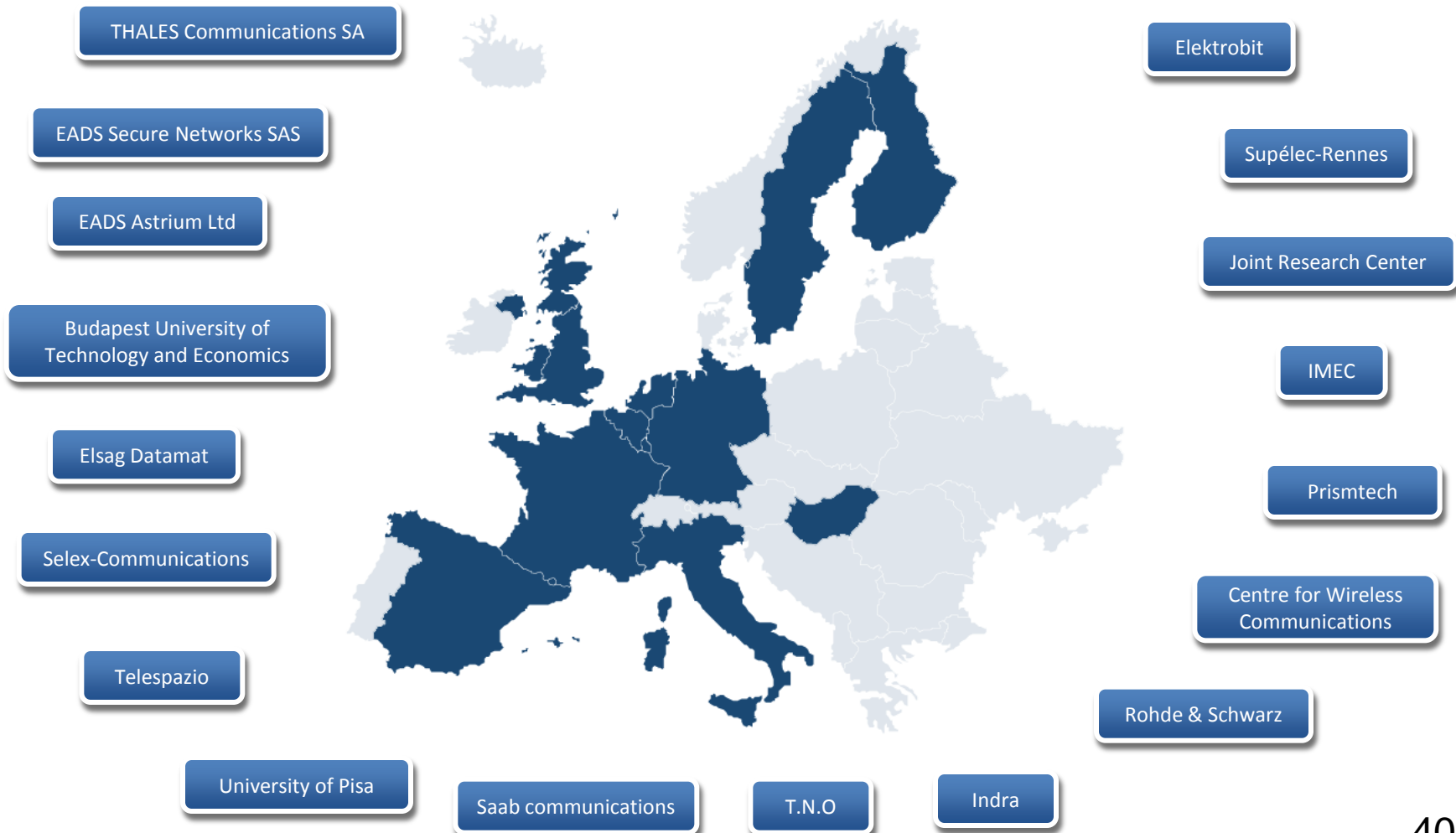
# EULER Transceiver Architecture Overview

1. Implementation on top of two processing resources: DSP and FPGAs (two RX and TX)
2. DSP: Basically implements the API which is wrapping the board BSP and Drivers
3. FPGAs: Handling the real-time, with a samples counter. Implementation provides a set of registers quite close to the Facility concepts.

# EULER Transceiver Architecture

## FPGAs important assumptions

1. Real-time clock: Which counts in samples at a given (programmed) sampling frequency. Fixed frequency, but easily tunable in a potential implementation extension.
2. Data buffers (FIFOs): Tailored for the WF application. 128KB data buffers. Large enough to store a complete Euler WF frame.



THANKS!

