Transceiver Facility / API

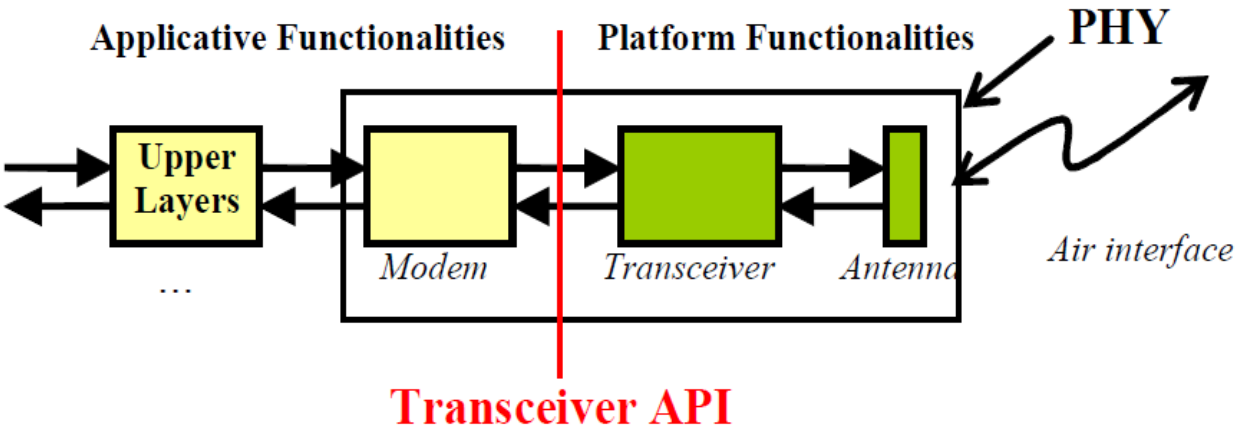
SDR applications Fast-prototyping on Ettus Research USRP platform

S. Thao, based on A. Sanchez, Implementation on USRP2 highlights

WInnF 72nd Working Meeting,
June 2012, Brussels

THALES

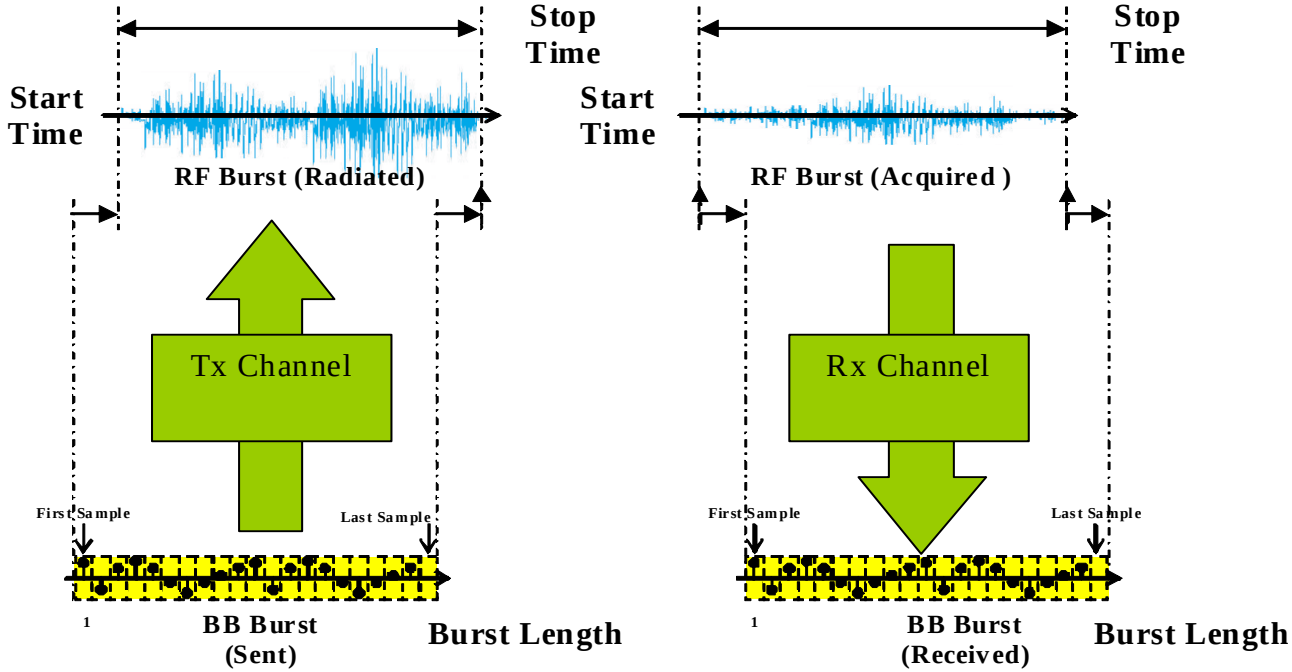
WInnF Transceiver Facility / API



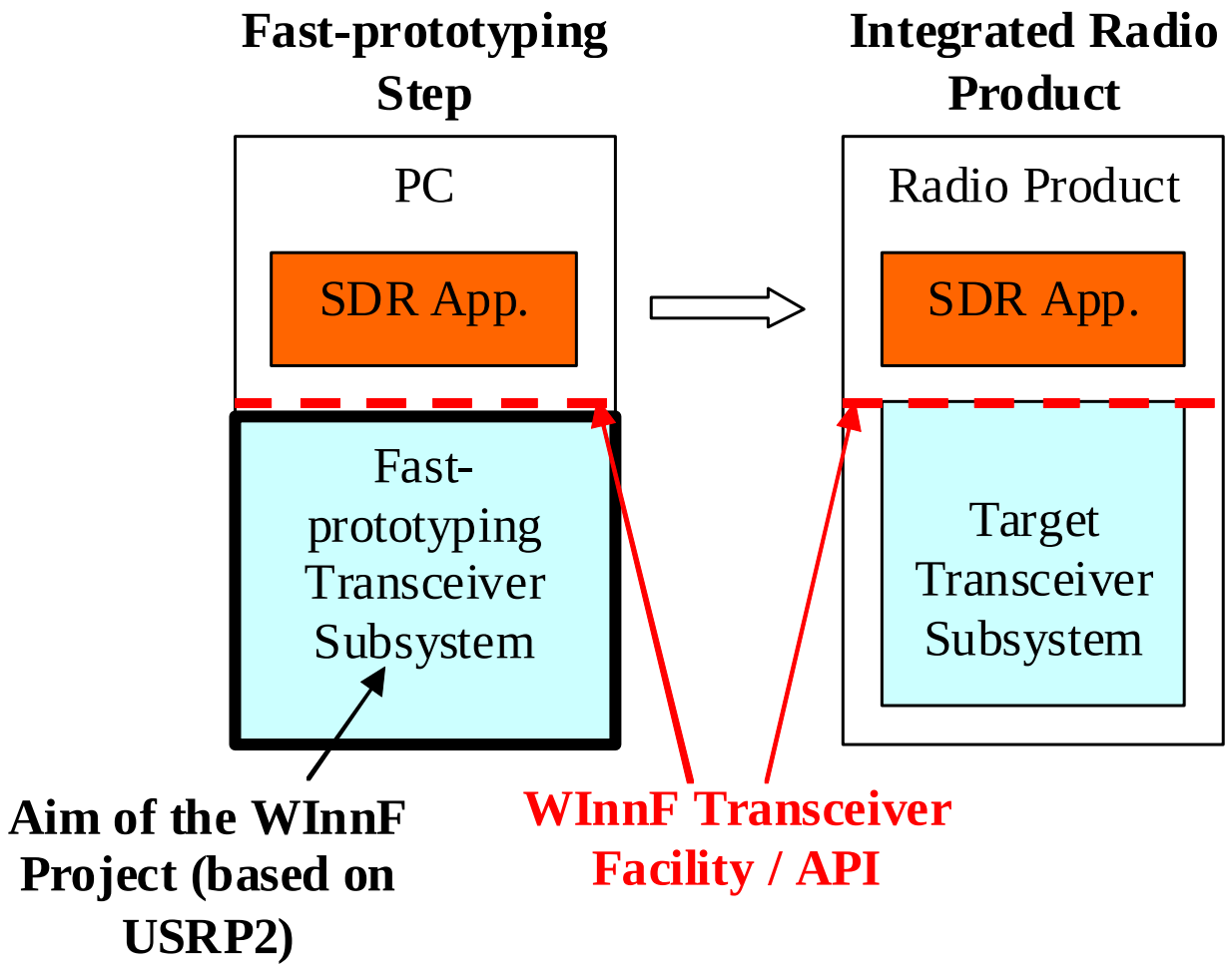
Transceiver interface sits between the "waveform" and the "platform"

Transceiver works on a Burst basis

- Time Profile: *Start Time* and *Stop Time*
- Tuning profile: *Carrier frequency*, *Tx power*, *Rx packet size* direct tuning parameters. Additional parameter called *Preset*



TCS Vision of fast-prototyping based on WInnF Transceiver API

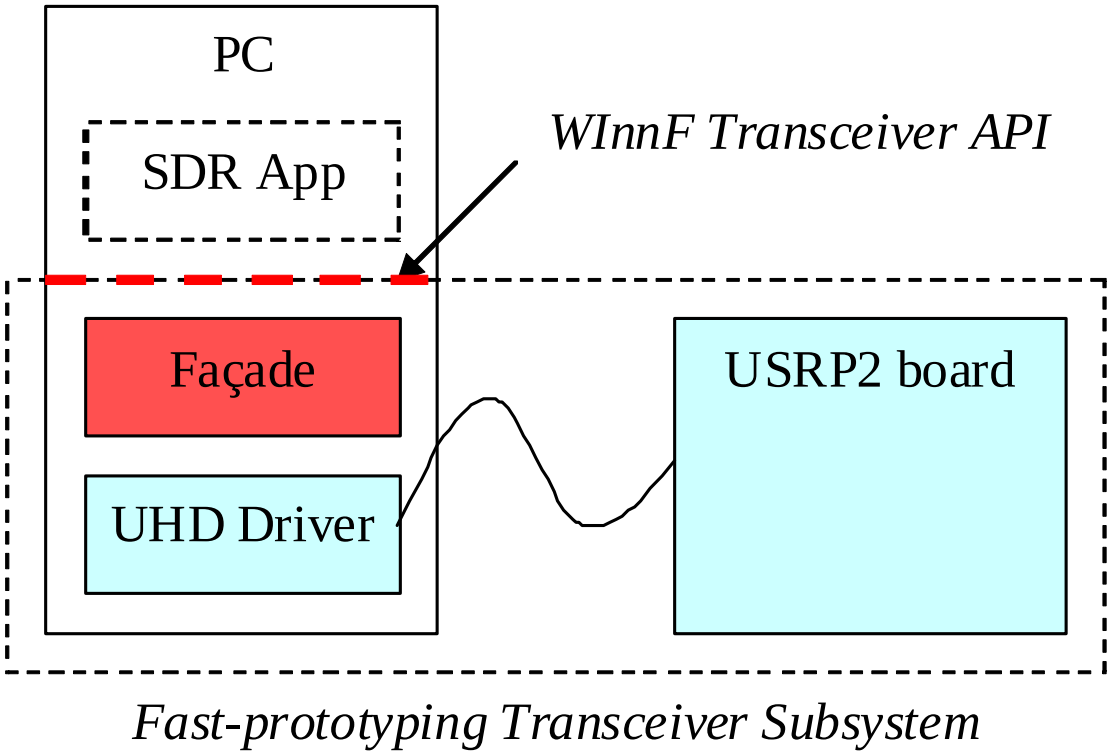


Project Name: Fast-prototyping with WInnF Transceiver Facility Implementation (Transceiver System Interface Task Group, TSI-TG)

What is understood by fast-prototyping? Solutions compliant with WInnF Transceiver Facility

THALES

Open source upgrade of USRP2 to WInnF Transceiver Facility



An Open source software, the "Façade", makes, in conjunction with a USRP2 board and its UHD driver, a Transceiver Subsystem compliant with the Transceiver Facility specification

- **Evolutions to the UHD will be introduced if required**

THALES

Outline

1) Used UHD Driver operations and modes

2) Architecture overview

3) Validation: Use Cases

Used UHD Driver operations and modes (1/3)

The transceiver configuration is conveyed on the PresetId as specified in the Facility specification document [1]

The contents of this preset are undefined and left open for each implementation. For the purposes of USRP2 implementation the Preset is simplified:

```
typedef struct PresetIdStruct
{
    // Rx configuration parameters
    std::string rx_antenna;
    double rx_gain;
    double rx_bandwidth;
    double rx_rate;
    // Tx configuration parameters
    std::string tx_antenna;
    double tx_gain;
    double tx_bandwidth;
    double tx_rate;
} PresetId;
```

[1] E. Nicollet, S. Pothin and A. Sanchez, *Transceiver Facility Specification*, Wireless Innovation Forum (WInnF), 2 February 2009, "*SDRF-08-S-0008-V1_0_0_Transceiver_Facility_Specification.pdf*". [Online]. Available: "http://groups.winnforum.org/p/cm/ld/fid=85"

THALES

Used UHD Driver operations and modes (2/3)

The programming of the Preset is straightforward thanks to the operations of the UHD driver:

```
void DeviceImp::configurePresetTx(PresetId inPreset)
{
        platform->set_tx_antenna(inPreset.tx_antenna);
        platform->set_tx_rate(inPreset.tx_rate);
        platform->set_tx_bandwidth(inPreset.tx_bandwidth);
        platform->set_tx_gain(inPreset.tx_gain);Default Gain 0
#ifdef DEBUG_ON
        std::cout << boost::format("New TX Antenna: %s") % (inPreset.tx_antenna) << std::endl;
        std::cout << boost::format("New TX Rate: %f Msps") % (inPreset.tx_rate/MHz) << std::endl;
#endif
}

void DeviceImp::configurePresetRx(PresetId inPreset)
{
        platform->set_rx_antenna(inPreset.rx_antenna);
        platform->set_rx_rate(inPreset.rx_rate);
        platform->set_rx_bandwidth(inPreset.rx_bandwidth);
        platform->set_rx_gain(inPreset.rx_gain); Default Gain 0
#ifdef DEBUG_ON
        std::cout << boost::format("New RX Antenna: %s") % (inPreset.rx_antenna) << std::endl;
        std::cout << boost::format("New RX Rate: %f Msps") % (inPreset.rx_rate/MHz) << std::endl;
#endif
}
```

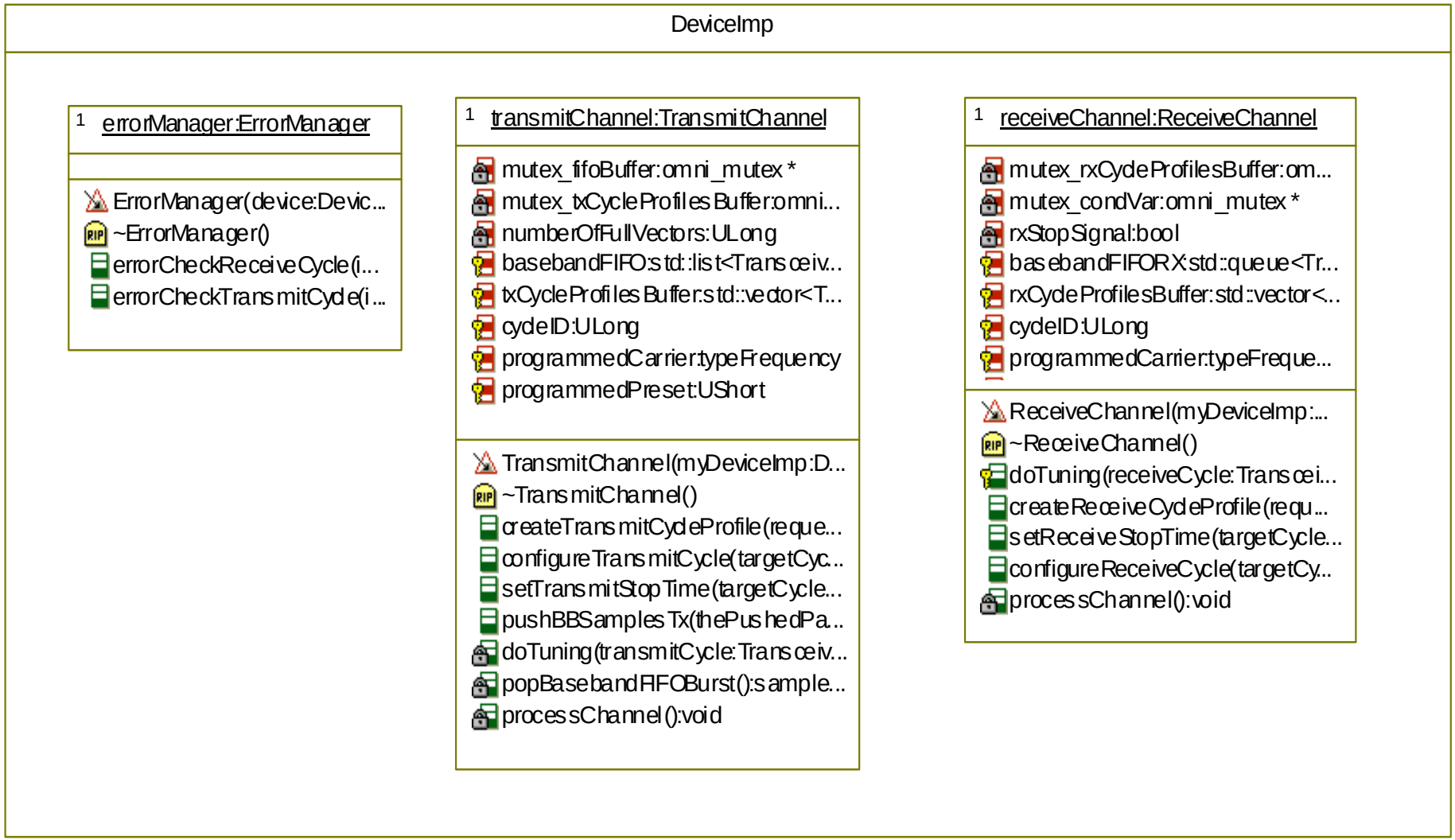
Used UHD Driver operations and modes (3/3)

The Transceiver Implementation (DeviceImp) creates Rx and Tx streamers:

(except from DeviceImp constructor code)

```
// Create Tx streamer
uhd::stream_args_t stream_args_tx("fc32", "sc16");
for(size_t chan = 0; chan < platform->get_tx_num_channels(); chan++)
    stream_args_tx.channels.push_back(chan); // linear mapping
tx_stream = platform->get_tx_stream(stream_args_tx);
// Create Rx streamer
uhd::stream_args_t stream_args_rx("fc32", "sc16"); // complex floats
for(size_t chan = 0; chan < platform->get_rx_num_channels(); chan++)
    stream_args_rx.channels.push_back(chan); // linear mapping
rx_stream = platform->get_rx_stream(stream_args_rx);
```

Architecture (1/8)



DeviceImp class and attributes

Architecture (2/8)

The two key design elements of the current implementation are:

- The time management
- The data exchange

The ways the Transceiver API and the UHD driver are used for tackling these two fundamental features are quite different:

- Transceiver Facility separates time management (time to send/receive) from data transmission and receive (samples exchange)
- UHD driver couples both in one single simple operation featuring a number of parameters for mode selection and control

Architecture (3/8)

For Transmission the send() operation is used:

- The key element is the uhd::tx_metadata_t md values setting:

```
md.start_of_burst = true;
md.end_of_burst   = true;
md.has_time_spec  = true;
md.time_spec = uhd::time_spec_t(time_to_send.secondCount,
    (double)time_to_send.nanosecondCount / NANOSECONDSTOSECONDS);
```

***time_to_send** is calculated previously depending upon the selected time mode. For instance immediateTime and eventBasedTime are supported.*

The approach is simple:

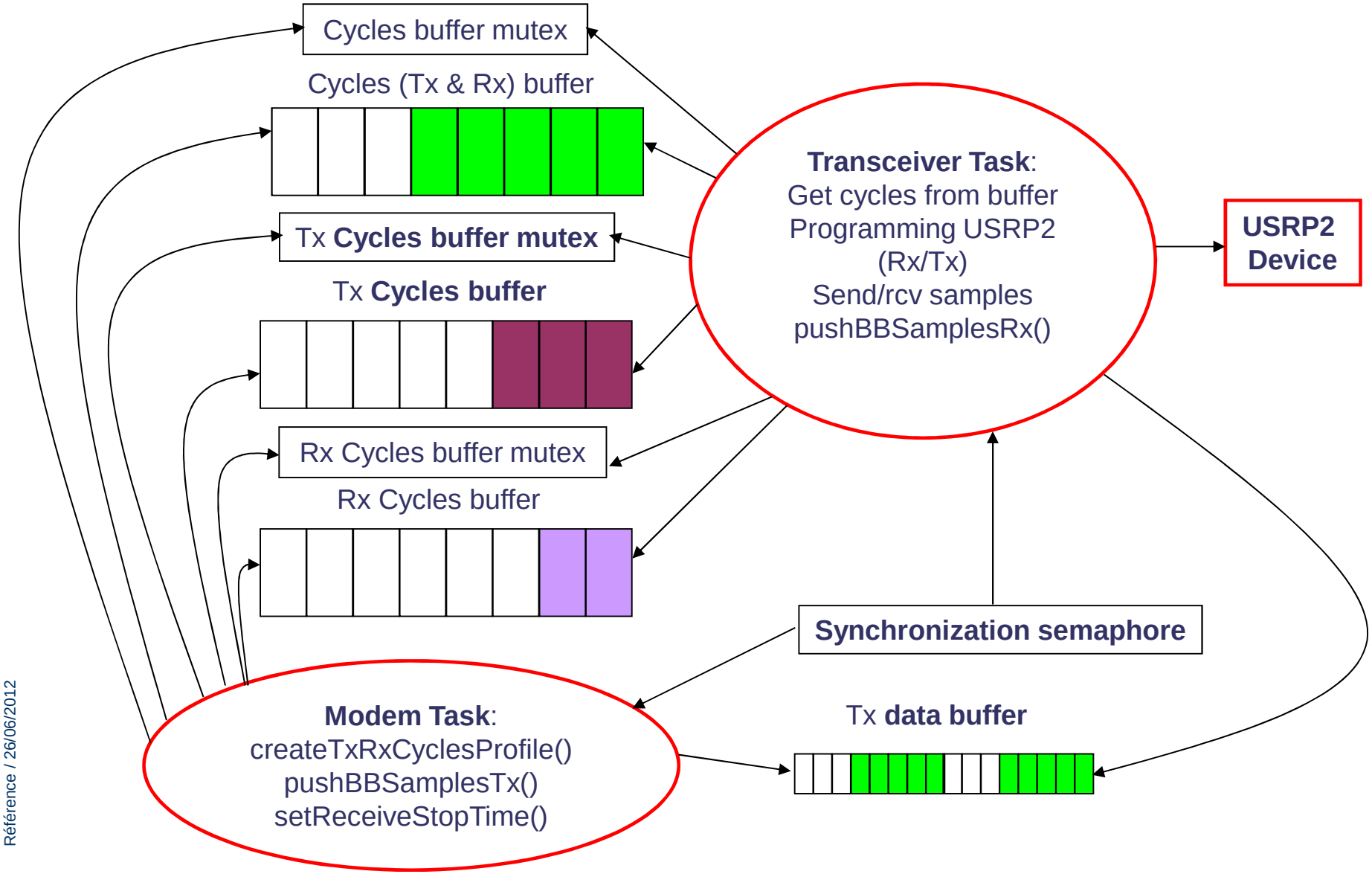
Basically the Transceiver (DeviceImp) keeps the latest TransmitStart and ReceiveStart Times and uses them as event sources for eventBased mode. The target times are added to these references.

Architecture (4/8)

The USRP2 board clock is started for the first activation (either Tx or Rx)

- **platform->set_time_now(uhd::time_spec_t(0.0));**
- **In fact the modem on the other side of the API does not know actually the time within the board. It operates in relative timing for eventBased mode (first eventBased request and subsequent Cycles are relative thus to the first activation)**
  - No **eventBased activation** is possible if no previous **immediate** one was executed
  - **Absolute mode** is **not supported**: synchronization between board and PC is quite cumbersome. A previous implementation tried to follow such an approach in which the **board clock** was read before each activation but the reactivity to retrieve the time and then act, introduced huge **delays** (around 6 ms worst)
  - **immediate mode** is made possible by keeping a margin
  - The key design principle is the **anticipation** of the **activations**

Simple Real-Time Architecture (5/8)



Tx Processing (6/8)

Really straighforward!

- Retrieve Tx tuning profile from Tx Cycles buffer

- Program USRP2 with the Tx tuning profile

- Retrieve Rx samples from Tx buffer

- Process Time Profile of Tx cycle

- **immediate Time → add margin to the last activation and set time_to_send to that value**
- **eventBased Time → add timeShift to the last activation time of the event and set time_to_send to that value**

- send()

Rx Processing (1/2) – Architecture (7/8)

Similar to the Tx processing, Undefined stop mode support is added (for common synchronization use cases)

- Retrieve Rx tuning profile from Rx Cycles buffer

- Program USRP2 with the Rx tuning profile

- Process Rx cycle Time profile

- **immediate Time → add margin to the last activation and set time_to_send to that value**
- **eventBased Time → add timeShift to the last activation time of the event and set time_to_send to that value**

.../...

Rx Processing (2/2) – Architecture (8/8)

- The UHD driver recv() command offers different stop criteria options. Depending on the stop time we may choose between:

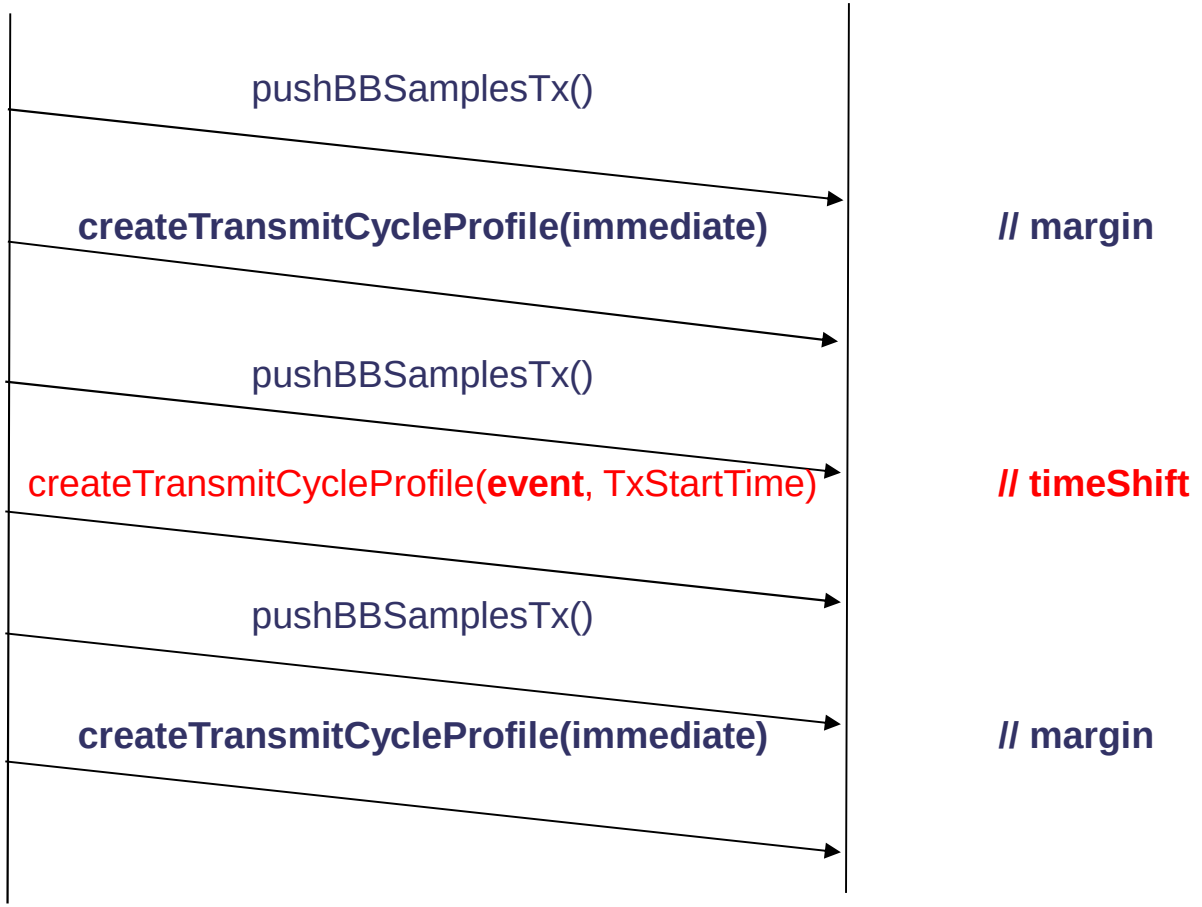
- **Undefined stop**: **stop time to be issued later on by the modem → option STREAM_MODE_START_CONTINUOUS**
- **Defined stop**: **for UHD the easiest way is to use the packetSize as the implicit stop time indicator. The driver will fragmentate data in UDP packets if the packetSize exceeds ethernet driver packet size. → option STREAM_MODE_NUM_SAMPS_AND_DONE**

- After recv() operation data is retrieved in a non blocking call (timeout parameter = 0), poll until data is received

- pushBBSamplesRx() is called by the Transceiver (implemented by the modem side of the API)

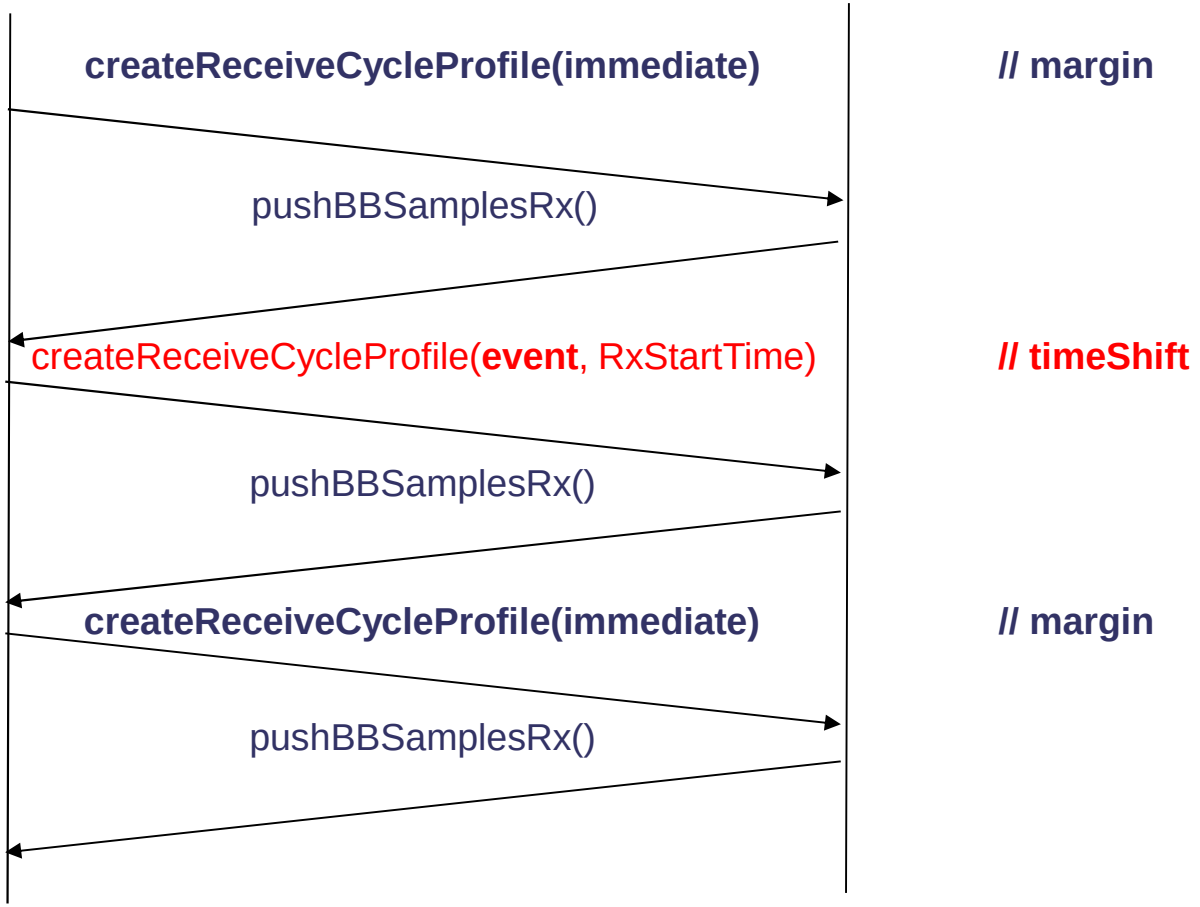
Validated use cases (1/4)

Transmitting (Tx) loop



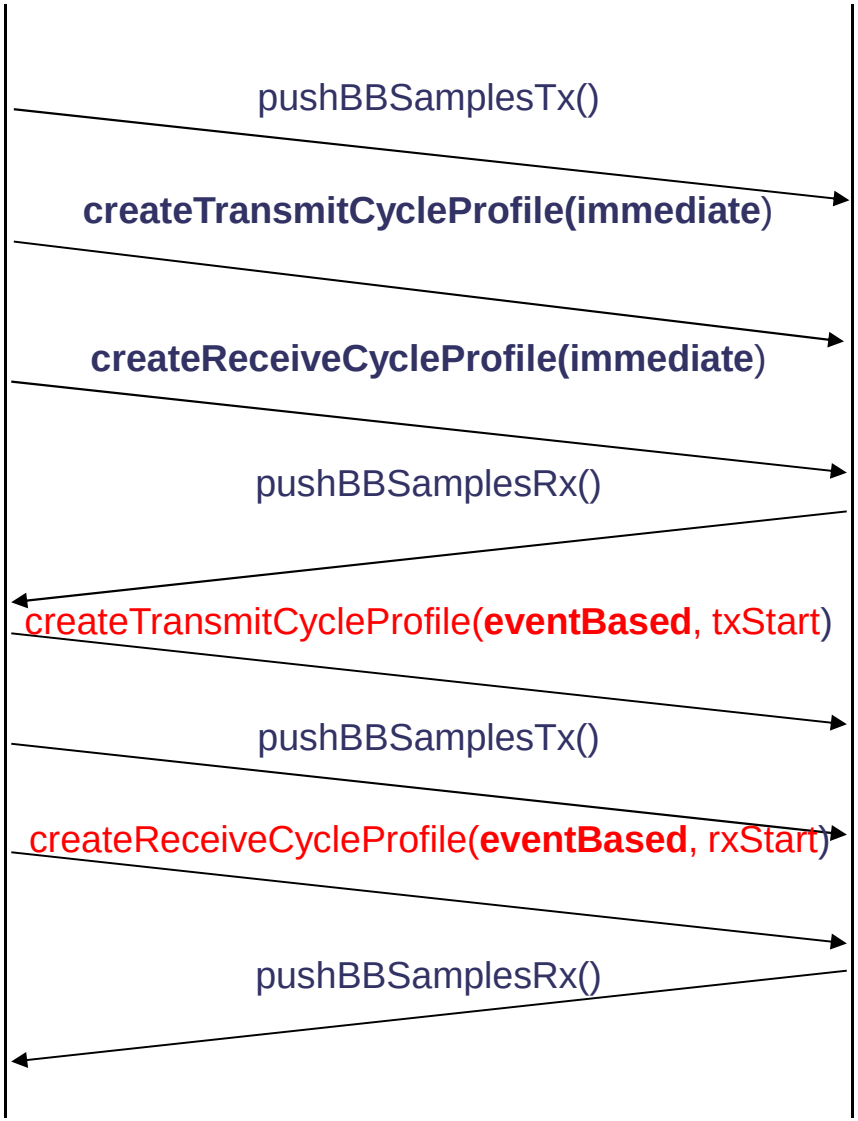
Validated use cases (2/4)

Receiving (Rx) loop



Validated use cases (3/4)

Transmitting (Tx)/Receiving (Rx) duplex loop



Validated use cases (4/4)

Typical synchronization (Rx) use case: undefined stop