

Interference mitigation in HetNet systems

From theory to practice



**Centre
Tecnològic
de Telecomunicacions
de Catalunya**

**Oriol Font-Bach
(Researcher)**

Outline

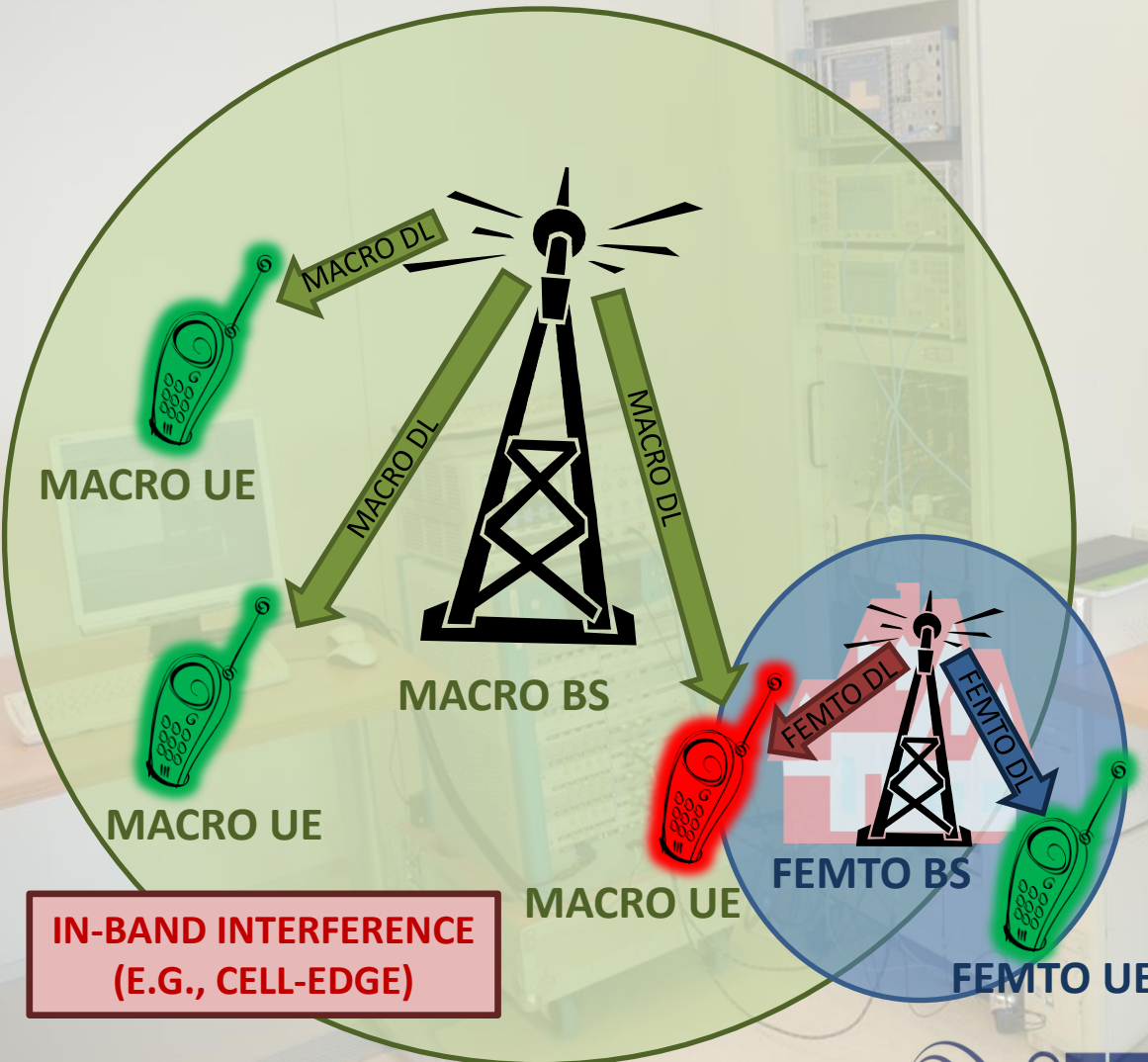
1. Introduction
2. Motivation
3. Digital design tendencies
4. Choosing a digital design flow for processing demanding systems
5. Development flow and prototyping platform
6. Interference management in HetNets
7. RTL design
8. Validation and results using the GEDOMIS[®] testbed

1. Introduction

Opportunistic spectrum reuse

- Evolution of wireless communication systems needs to address many issues
 - Congested RF spectrum → opportunistic reuse: i.e., objective of **CR**
 - **Problem:** in-band interference → secondary communication degrades QoS perceived by primary users
 - **Interference management** solutions are **required!**
 - Combined with high performance and demanding operating conditions → advanced PHY-layer schemes (e.g., MIMO-OFDM(A), closed-loop, wide bandwidth).

Spectrum-reuse example: femtocells



- address indoor topology losses
- spectrum reuse (e.g., same band & bandwidth)
- co-channel interference

Interference management

- Inter-Cell Interference Coordination (ICIC) schemes for HetNet (e.g., Macro/Femto)
 - Interference avoidance
 - E.g., spectrum sensing → allocate unused bands
 - Interference mitigation
 - E.g., interference-detection & adaptation of opportunistic transmission
 - required to enable frequency reuse → same band is used by different users among adjacent (heterogeneous) cells

2. Motivation



What motivates this tutorial?

- Two main factors:

1) Practical need for real-time implementations

- a. PHY-layer of a BWA system (e.g., LTE) featuring an interference management scheme
- b. Utilization of a heterogeneous prototyping platform (e.g., FPGA-based, using COTS RF + channel emulation)

2) Need to employ innovating digital design techniques to fulfill 1)

- a. Efficient utilization of baseband processor capacity
- b. Address implementation challenges posed by real-time DSP, wide bandwidth & complex baseband algorithmic

Why real-time PHY prototyping? (I)

I can rapidly model my algorithm/system with floating point code in a computer-based simulation and simulate as many scenarios as I wish...

If I'm struggling with extremely long simulation times, I can always make system-wide simplifications

No worries for hardware specifications, implementation cost, undesired operating or signal conditions ...

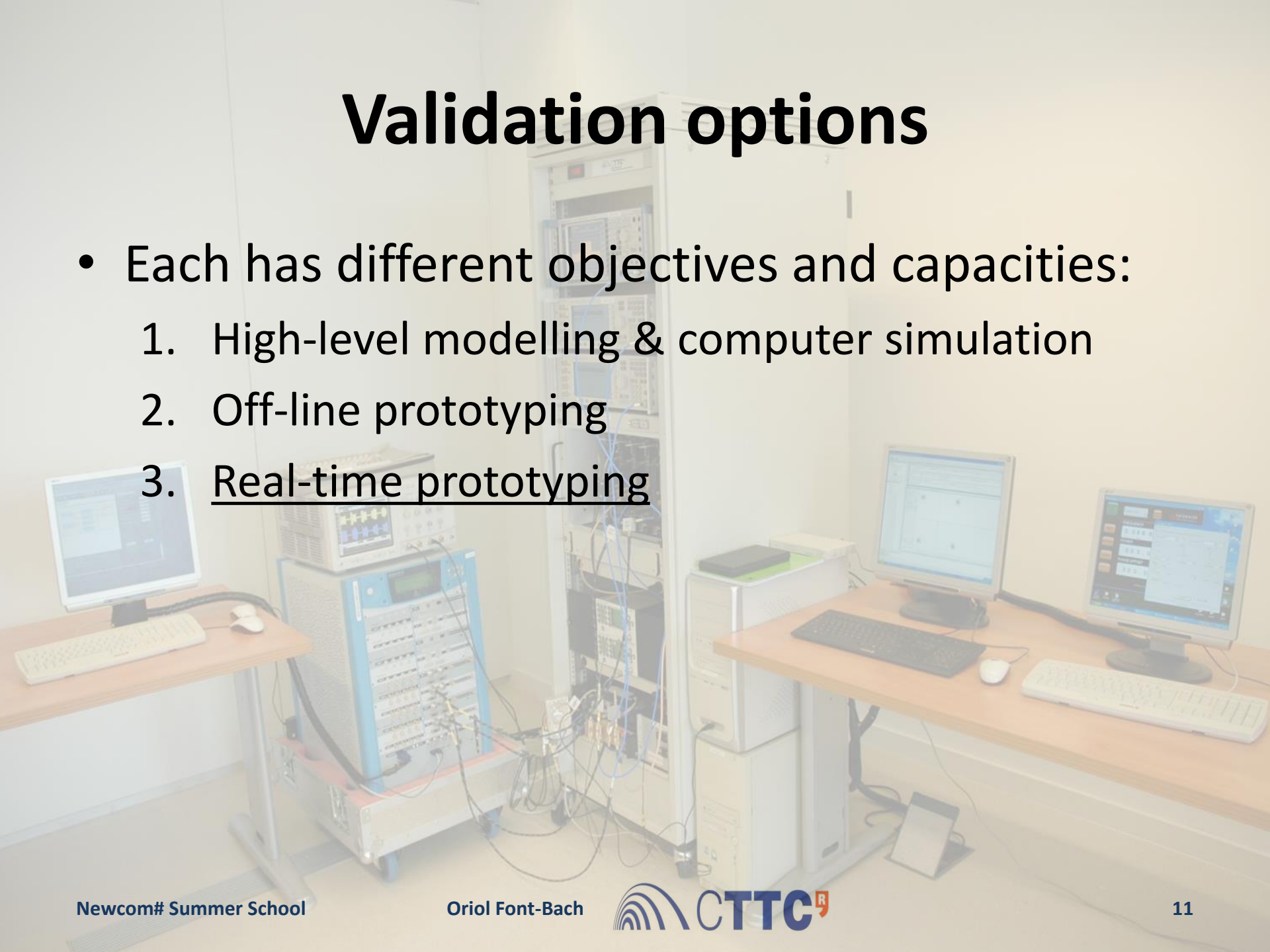
Why do I have to bother about implementation?

Why real-time PHY prototyping? (II)

- **Objective** = realistic validation of a (high-performance) Macro/Femto interference-management scheme
- **What affects the performance of the PHY-layer scheme under analysis?**
 - Low-level HW-specifications, limitations & impairments introduced to the signal
 - Realistic signal propagation conditions (including mobile channels, noise and interference)
 - Capacity of the target processing solution

Validation options

- Each has different objectives and capacities:
 1. High-level modelling & computer simulation
 2. Off-line prototyping
 3. Real-time prototyping

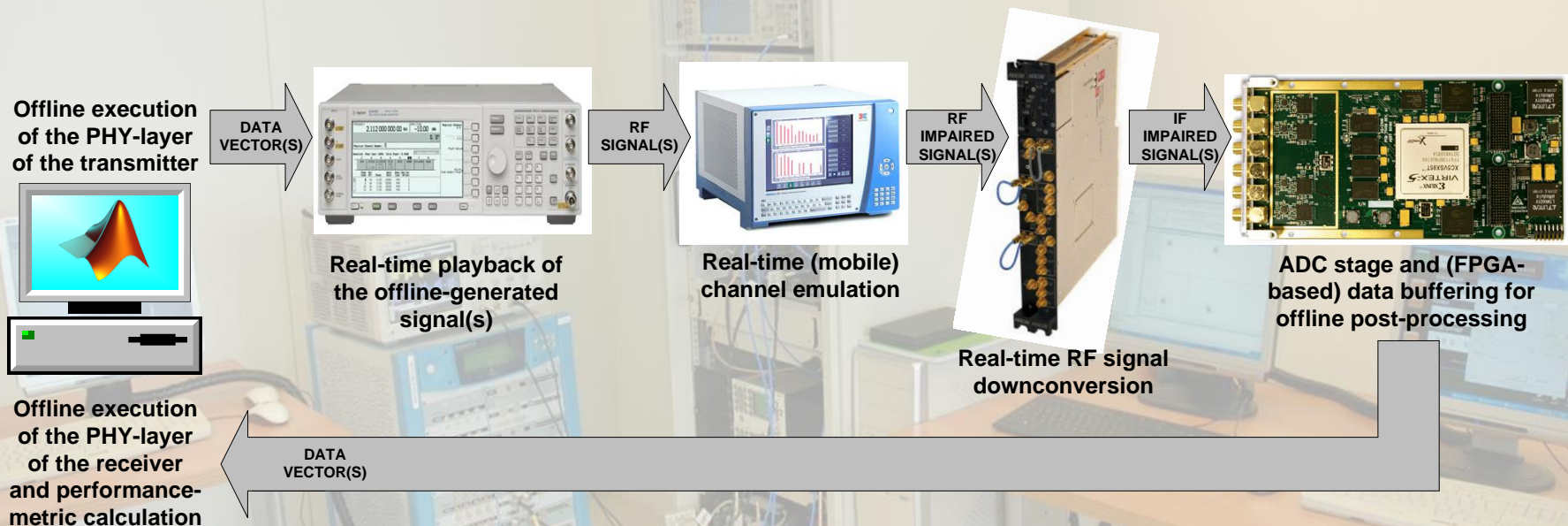


HLPL-based PHY-layer modelling

- Natural starting point of DSP research (e.g., MATLAB)
 - Pros: flexible, low cost (time and money), rapid evaluation of innovative techniques
 - Cons: limited capacity to...
 - deal with computational intensive data processing (e.g., wide bandwidth, MIMO)
 - model or account for realistic signal conditions and hardware-introduced impairments (e.g., mobile channel)
 - reproduce dynamic behavior at run-time (e.g., closed-loop)

Common assumptions and simplifications: idealized channel conditions, perfect synchronization, perfect CSI at Tx, ignores implementation cost, unlimited numerical precision... ➔ **IMPACTS PERFORMANCE ASSESSMENT**

Off-line PHY-layer prototyping (I)



Off-line PHY-layer prototyping (II)

- Combination of HLPL-based PHY-layer with COTS RF + over-the-air/channel emulation
 - Pros: keeps flexibility and low-development cost of software-based PHY modelling, considers realistic signal conditions (including HW-introduced impairments)
 - Cons: equipment cost (& stability of setup), still features limited capacity to...
 - deal with computational intensive data processing
 - reproduce dynamic behavior at run-time

An improved step towards realistic validation, but it is still common for modelled PHY to feature assumptions and simplifications: perfect CSI at Tx, ignore implementation cost... → IMPACTS PERFORMANCE ASSESSMENT

Real-time PHY-layer prototyping

- COTS RF + over-the-air/channel emulation + real-time DSP implementation (e.g., FPGA)
 - Pros: enables bit-intensive (adaptive) DSP, allows realistic validation by considering: close to real-life operating and signal conditions & HW limitations and implementation cost.
 - Cons: development bounded by...
 - long cycle → design, implementation and verification requires a lot of effort (time!)
 - elevated hardware cost
 - HW-specifications (finite resources & dynamic range)

Limits range of scenarios and PHY-layer schemes that can be considered (e.g., number of users, number of antennas...) → proof-of-concept

Why is it required innovative digital design? (I)

Ok, let's consider the real-time implementation of the proposed PHY-layer schemes...

What does it make their development so demanding?

Why is it required innovative digital design? (II)

- Design complexity increases because of:
 - Bandwidth
 - 4G BWA → up to 100 MHz!
 - Number of antennas
 - Real-time operation
 - Requiring parallelism + large storage capacity → complicated control-plane
 - Run-time adaptivity
 - Feedback generation, transmission, reception and reconfiguration of the PHY-layer
 - Realistic signal impairments
 - DFE, channel estimation...
 - Depending on the application, due to the required intelligent-utilization of the provided FPGA-resources

Why is it required innovative digital design? (III)

Conclusion: the design complexity motivates the inclusion of critical novelties, which are not directly related to the proposed DSP algorithmic, but to its actual implementation in a dedicated processing architecture

... plus modern FPGAs are offering unprecedented processing capacity

But nowadays there are plenty of vendor-provided tools to convert my HLPL-based model to a fully working FPGA implementation, right?

3. Digital design tendencies

General overview

- Focusing on FPGA-based developments
- Main HDL design approaches
 - Automated HDL generation
 - HLPL-to-RTL
 - Schematic-entry to HDL
 - Custom HDL with 3rd-party IPs
 - Full-custom HDL (i.e., gate-level design)

HLPL-to-RTL (I)

- Growing (EDA industry) interest in higher level design methodologies
 - System level tools/design methodologies are being explored.
- Motivation #1 → getting to a broader audience
 - No requirement for HDL or digital design skills
- Motivation #2 → IP reuse
 - Marketing & commercial tool for FPGA manufacturers
- Motivation #3 → need for High-Level Design
 - Higher level of abstraction → ever-increasing design complexity
 - **Reduce design efforts**
 - Fast development time
 - Technology independence → no need to consider low-level architecture of target FPGA device (?)
 - Ease of HW/SW partitioning

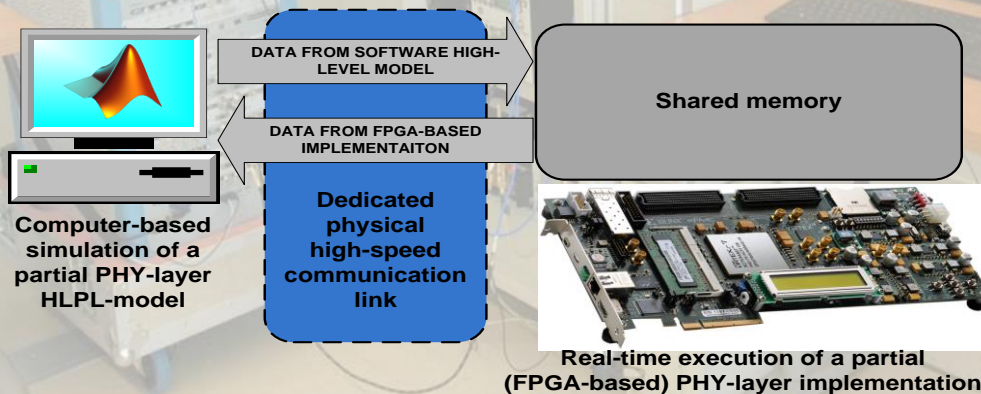
HLPL-to-RTL (II)

- Multitude of solutions today
 - C-based: SystemC, Simulink Coder, Synphony C Compiler, Catapult HLS, Xilinx Vivado...
 - Matlab-based: Mathworks HDL Coder, AccelChip DSP, System Generator for DSP...
 - Java-based: Forge, JHDL
 - Python-to-HDL: MyHDL

Schematic-entry to HDL (I)

Case study - Matlab-Simulink + System Generator for DSP

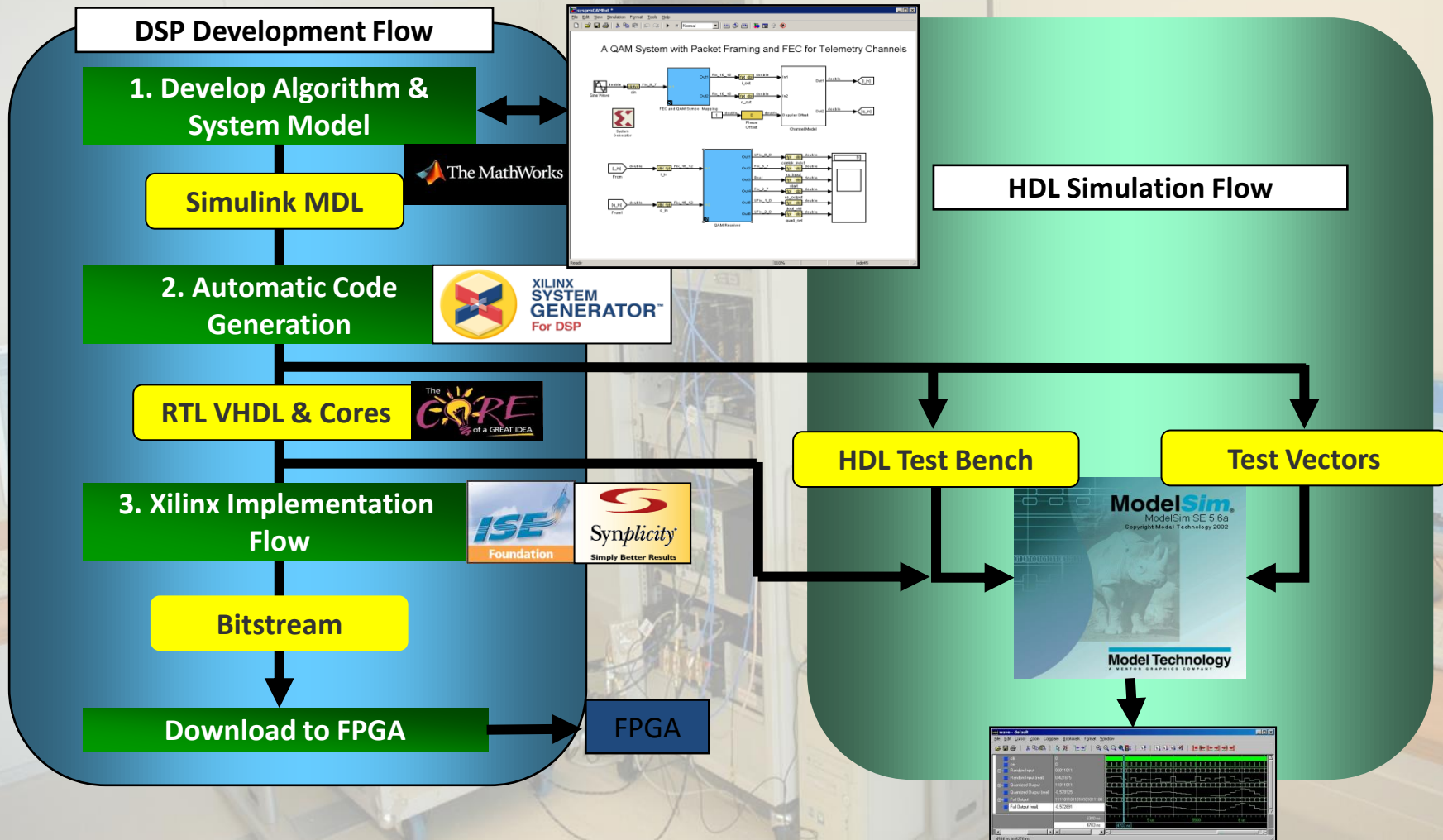
- Model-based design entry
 - Drag n' drop processing blocks + interconnect them
- Provides SW design utilities & precompiled mathematical functions
 - Many signal processing or specialized toolboxes included
- Includes optimized RTL IP libraries → System Generator for DSP
 - Xilinx offers a limited subset of the Core Generator IP cores
- Computer based simulation + automatic HDL generation
- Allows combination with other HDL coding approaches:
 - HLPL-to-RTL: user can include custom Matlab (M-code blocks)/C code
 - Custom HDL: user can instantiate it using the “black box primitive”
- Offers a hardware-software co-simulation environment → e.g., HIL



A nice way to
accelerate
simulations!

Schematic-entry to HDL (II)

Matlab-Simulink + System Generator for DSP development flow



Automated HDL (I)

That is a great step forward!



However... extracting all the concurrency from a sequential HLPL description is not an easy problem

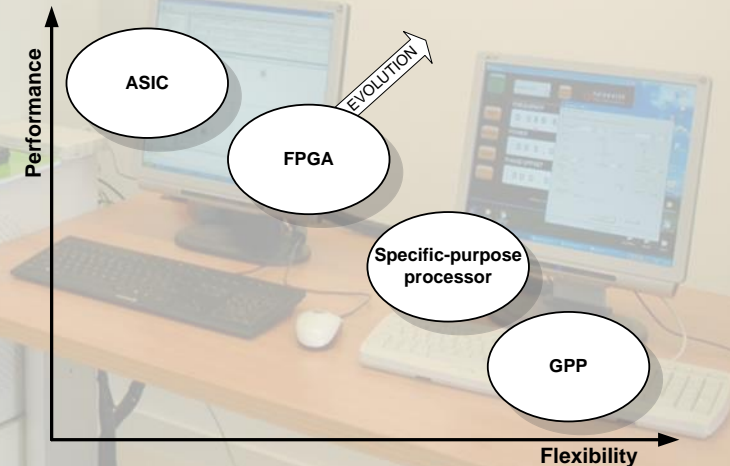
Automated HDL(II)

The downsides...

- HLS are inevitably less efficient (than custom RTL design)
 - Problematic for complex designs requiring an elevated amount of FPGA resources → cannot meet the required timing, area, or performance
 - Also... limited access to low-level implementation options of EDA tools
 - E.g., in C-to-RTL efficiency might be increased by introducing specialized FPGA-constructs (to force the utilization of specific embedded resources) → increases design time & complexity
- Coding limitations → HLPLs may...
 - Permit a certain subset of known commands
 - Require a specific source-code syntax
 - Impose/require certain code optimizations/restrictions
 - Constraint the maximum achievable performance
- Requirements for parallelism → high performance computing
 - Makes tougher to code with HLPLs

Special focus on the Vivado IDE (I)

- Xilinx promotes HW/SW co-design
 - Vivado is centred around high-level design
 - IP re-use + HLS
 - Zynq devices
 - Co-processing architecture
 - FPGA + dual-core ARM processor
 - Flexibility + performance
 - Wider range of end applications & customers



Special focus on the Vivado IDE (II)

- Pros: Flexibility complements the traditional parallelism offered by programmable logic
- Cons: HW/SW co-design and use of HLS is not trivial, although specialized SW tools and IP cores are being made available
 - E.g: Vivado HLS → specialized C-code including “FPGA-pragmas” and requiring several refinement iterations → development cycle time and design complexity comparable to that of custom HDL code generation

Metric	RTL expert	AutoPilot expert	AutoPilot expert
Dev. time (man-weeks)	4.5	3	5
LUTs	5,082	6,344	3,862
Registers	5,699	5,692	4,931
DSP48 s	30	46	30
18 K BRAMs	19	19	19

Implementation results for a 8x8 MIMO sphere decoder

(Note that Xilinx bought the AutoPilot HLS tool from UCLA and incorporated it into Vivado)

J. Noguera, S. Neuendorffer, S. V. Haastregt, J. Barba, K. Vissers, and C. Dick, “Implementation of Sphere Decoder for MIMO-OFDM on FPGAs Using High-level Synthesis Tools,” *Analog Integrated Circuits and Signal Processing*, vol. 69, no. 3, pp. 119–129, Sep. 2011.

Custom HDL coding (I)

- Custom HDL is hard to deliver and very costly in time but it will always be necessary...
 - Lack of pre-verified IP cores
 - Dense designs
 - Whenever an optimum HDL implementation is the goal
 - ...

... even if is only utilized on small portions of the design

Custom HDL coding (II)

- Provides the means to control every important aspect of the design
 - Low-level definition of a dedicated RTL architecture → optimized for performance, minimized resource utilization...
 - Benefits from utilization of **3rd-party IP cores**
 - Optimized for target FPGA device: e.g., Xilinx Core Generator (FFT, FIR...)
- Efficient design requires in-depth knowledge of target FPGA architecture & associated EDA tools

Custom HDL coding (III)

Example - Three steps to boost performance

1. Utilize embedded (dedicated) resources

- DSP slice, block RAM, ISERDES, OSERDES, EMAC, and MGT
- Dedicated hardware block timing is correct by construction
- Not dependent on programmable routing
- Offers as much as 3x the performance of soft implementations

2. Write code for performance

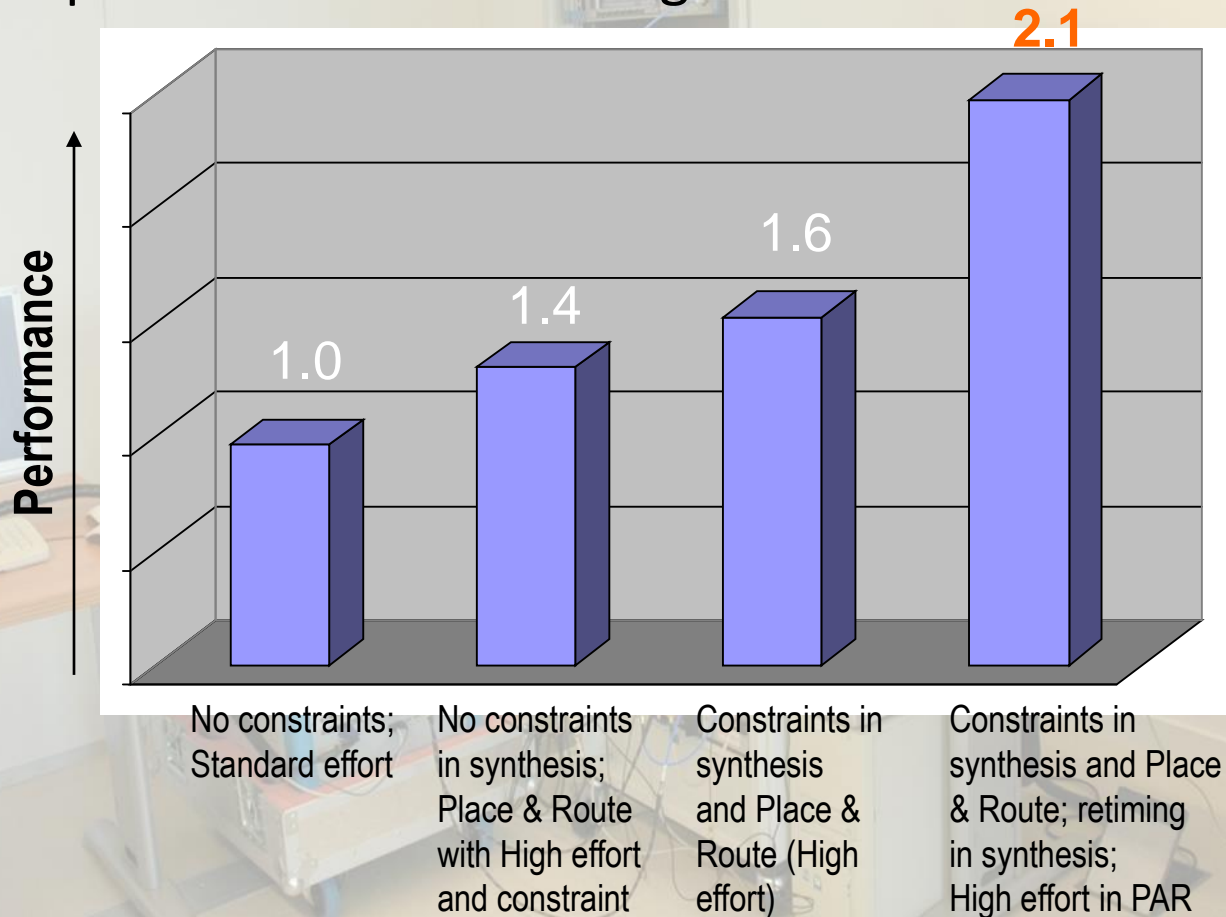
- Use pipeline stages—more bandwidth
- Use synchronous reset—better system control
- Use Finite State Machine (FSM) optimizations
- Use inferable resources (e.g. MUX, Shift Register LUT (SRL), BRAMs, Cascade DSP)
- Think about the levels of logic required for the logic you are building
- Be aware of the inferred circuits & the expected combinatorial complexity

3. Drive your synthesis and Place & Route tools

- Try different synthesis optimization techniques
- Add critical timing constraints in synthesis
- Preserve hierarchy
- Apply full and correct constraints
- Use High effort

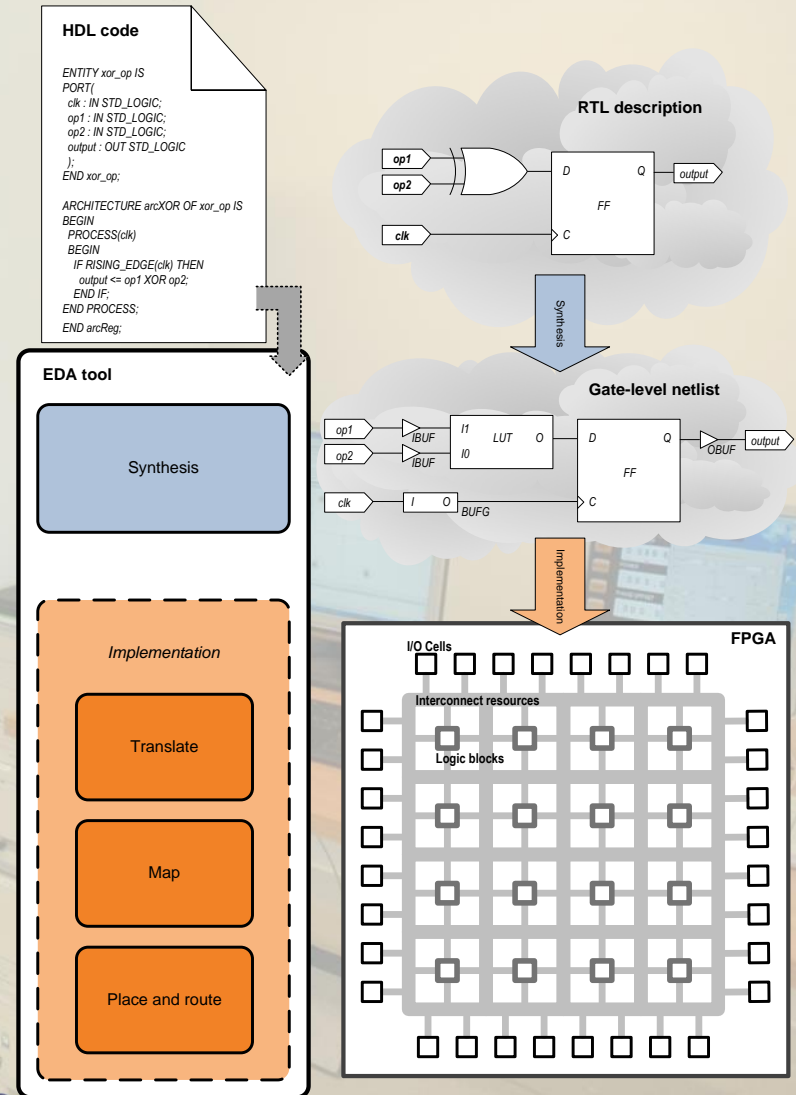
Example of the impact of constraints in EDA tools

- Example Reed-Solomon design



Full-custom HDL (I)

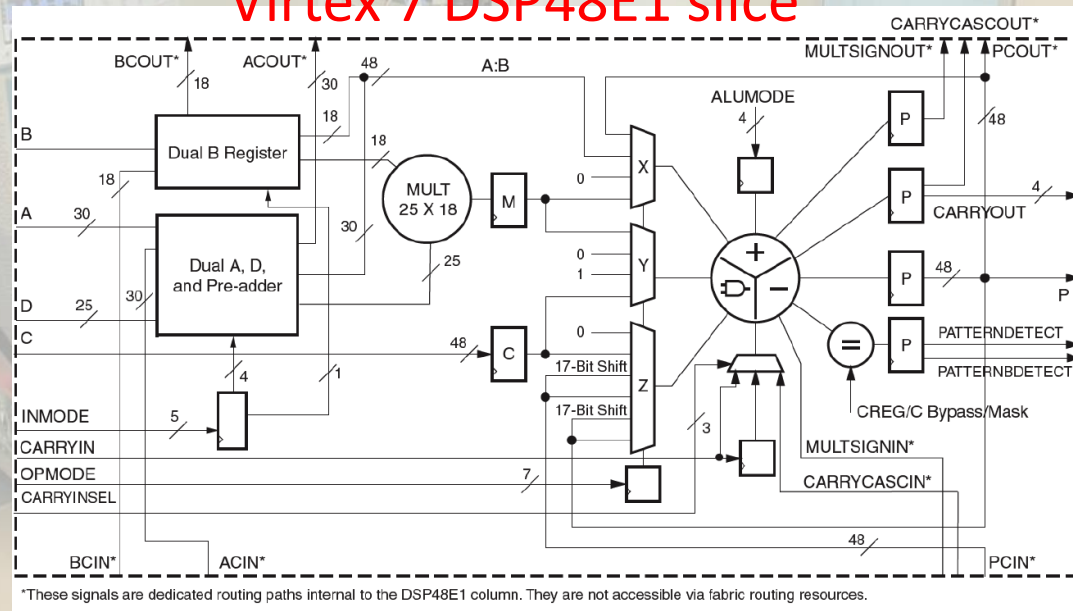
- *HDL design* = top-down methodology
- Code is translated (in various phases) to a low-level description of the circuit
 - Very abstract design description yields poor results
 - Detailed description drives the decisions of the translation process



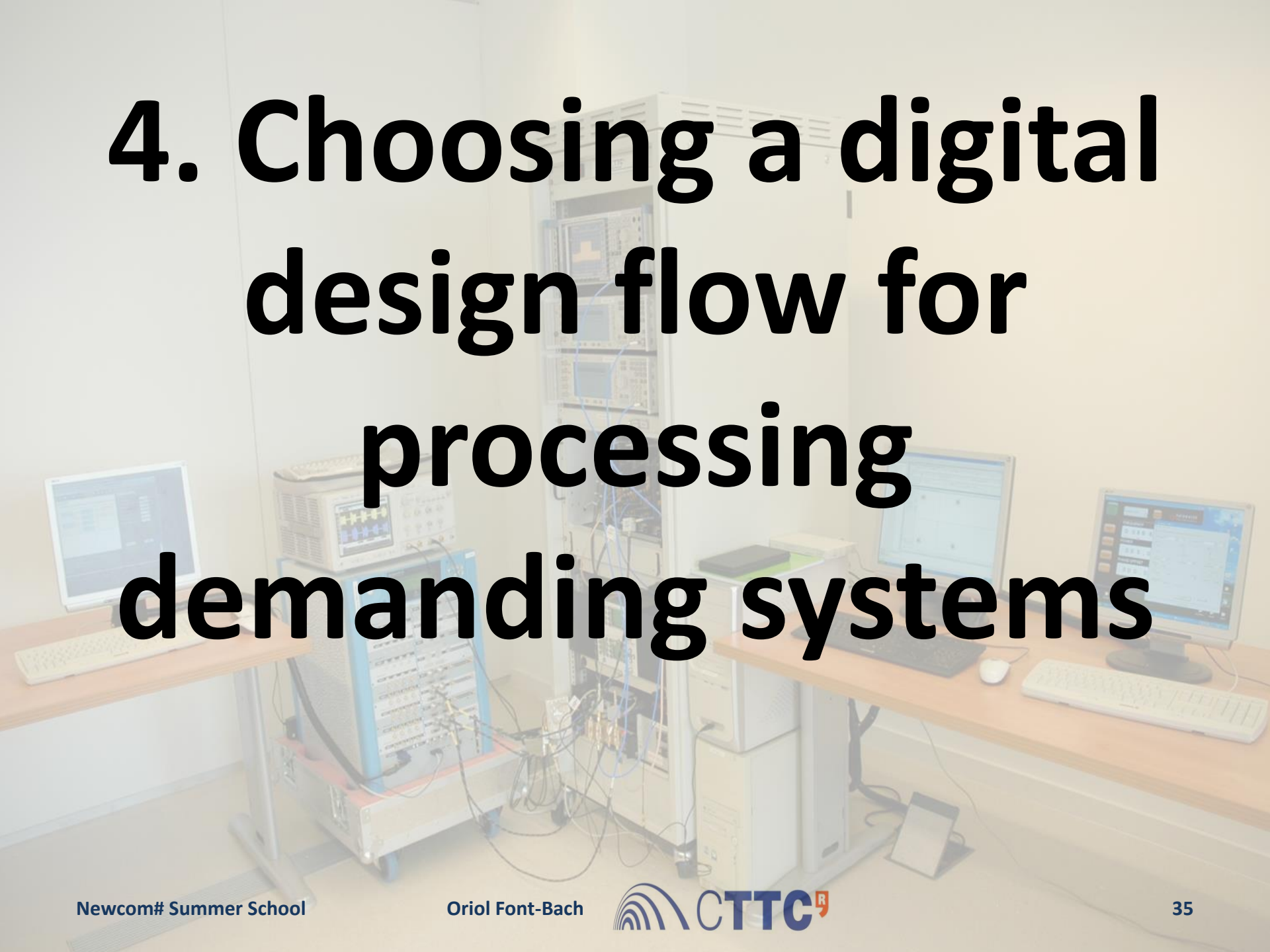
Full-custom HDL (II)

- **Gate-level design** → force utilization of the instantiated primitives (*avoid automatic inference*)
- Fully-optimized design → ASIC prototyping
 - Area, performance, consumption...
- Requires full knowledge of low-level architecture of the target FPGA

Virtex 7 DSP48E1 slice



4. Choosing a digital design flow for processing demanding systems

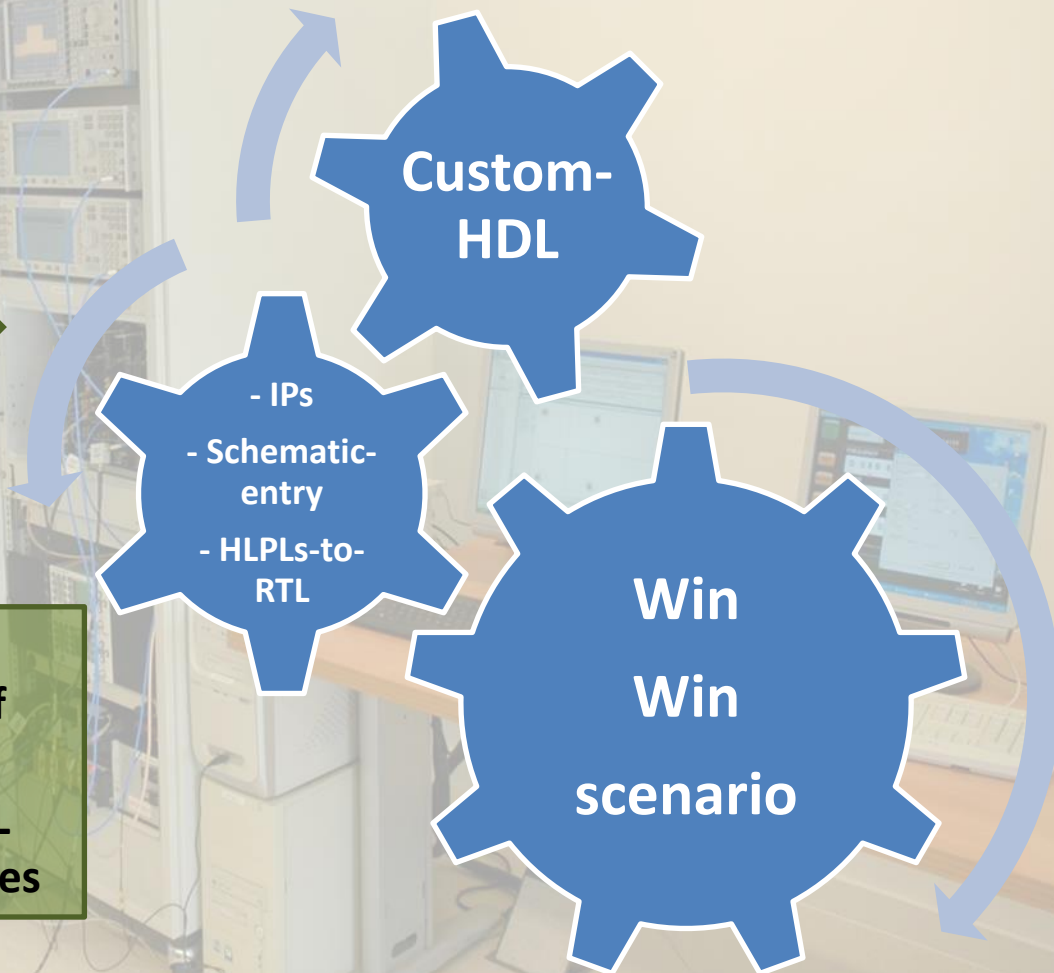
The background image shows a laboratory or classroom setting. In the center, there is a tall rack of electronic equipment, including what appears to be a signal generator or a logic analyzer, with various cables connected to it. On either side of the rack, there are computer workstations. The workstation on the left has a monitor displaying a software interface, a keyboard, and a mouse. The workstation on the right also has a monitor, keyboard, and mouse. The overall scene suggests a technical environment for digital design and system processing.

Different system-design cases require different design solutions!

1. Investigate thoroughly your system design requirements
2. Select the most appropriate development flow

- a) HDL coding approach
- b) IDE solution/target technology
- c) Validation strategy

Example:
combination of
custom and
automated HDL
coding approaches



How to select an appropriate design methodology?

- Parameters to consider:
 - Target use → experimental prototype, product...
 - Scope of application defines fundamental specifications → BWA, power-line communications, space, medical...
 - Cost! → budget for HW, SW, PMs...
 - Design objectives → performance, low-power, area... trade-off?
 - Operation mode defines design constraints & HW complexity → real-time, off-line?
 - Technical skills of the team + available HW/SW → mature processing technology, pre-verified IPs...

Use case: designing a processing demanding PHY-layer scheme (I)

Analysis of the presented use case

Target use	Experimental prototype using COTS HW
Application: general scope	Macro/femto interference-mitigation scheme for BWA systems
Application: low-level scope	High performance adaptive DSP, baseline standard compliance (3GPP LTE Rel. 9)
Cost	HW/SW budget defined, PM defined
Design objectives	Performance, Portability, Extendibility
Operation mode	Real-time
Existing programming & design expertise	VHDL, Matlab, C, Java
Existing SW tools and HW equipment	Prototyping boards, pre-verified IPs, licenses for SW design packages
Capacity for real-life system validation	Signal generation/acquisition HW, system-wide testing & debugging using various equipment, board-level code integration

Use case: designing a processing demanding PHY-layer scheme (II)

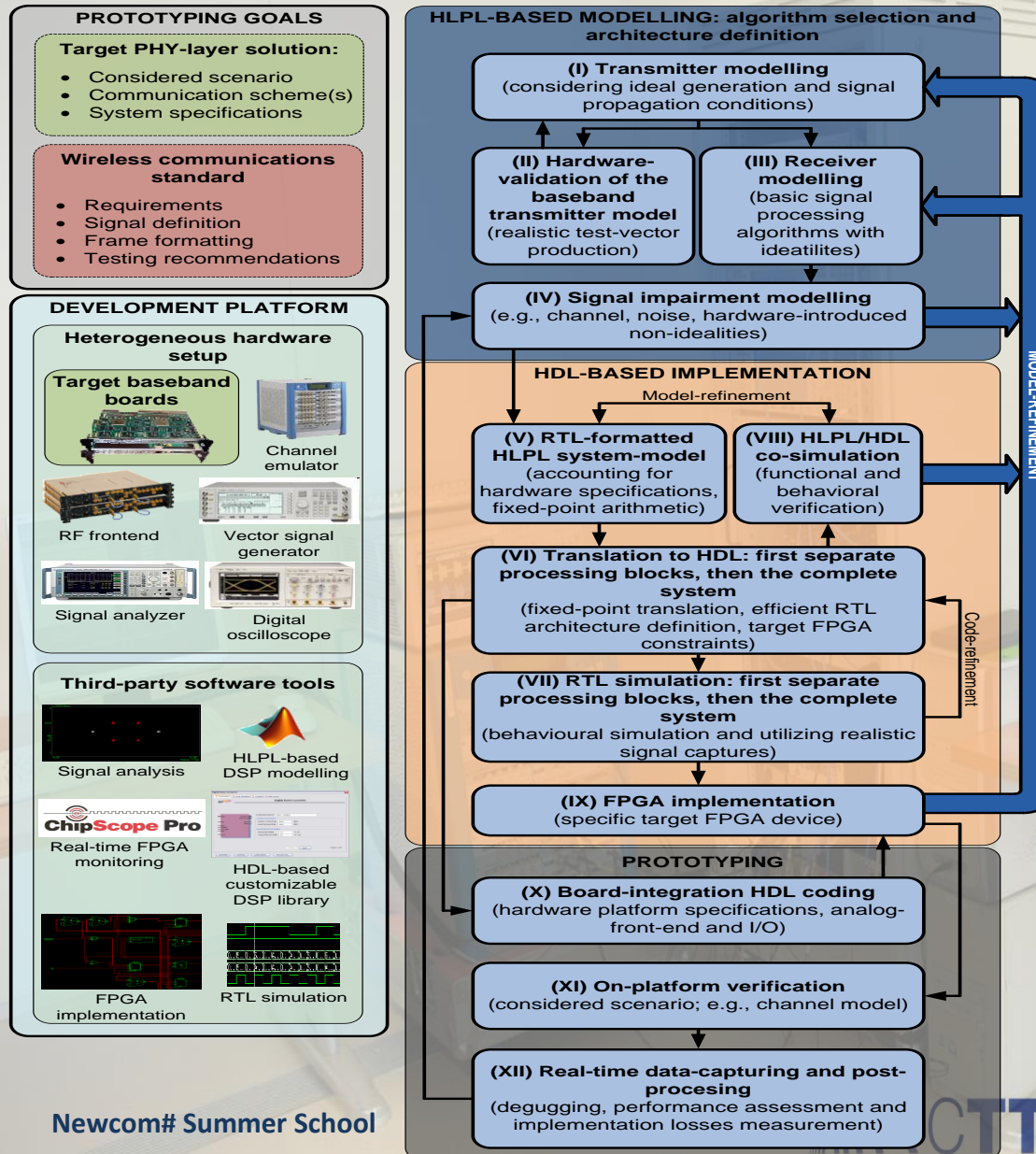
- Given the previous analysis and the described motivation it has been selected a...

**CUSTOM HDL CODING APPROACH
RELYING ON 3rd-party IP CORES**

(more details on the full development flow and target prototyping platform follow)

5. Development flow and prototyping platform

Proposed incremental development flow



1. Idealized HLPL model
 - algorithm selection
 - implementation cost
2. Off-line Tx prototyping
 - hardware-validation of Tx
 - experimental captures
3. HLPL-model refinement
 - realistic signal
 - RTL-awareness
4. RTL design (custom HDL)
 - co-simulation (IP config.)
 - test DFE in HW → back to 3
5. FPGA implementation
 - platform integration
6. On-lab validation
 - debugging → chipscope + equipment
 - real-time data captures
7. Performance assessment
 - post-processing → metrics

Development challenges

- Heterogeneous prototyping platform
 - Characterization → early identification of performance bottlenecks
 - Stability → E.g., intermediate signal-powers
 - Hardware-originated signal impairments
- Channel and mobility effects
- Translation to RTL → E.g., fixed-point
- FPGA-design partitioning
- Design and implementation software tools → recall previous example!

The GEDOMIS[®] testbed (I)



The GEDOMIS[®] testbed (II)


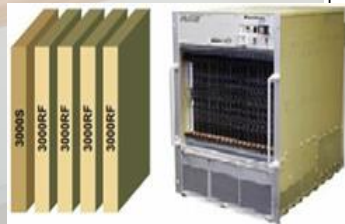
- Signal conversion and baseband processing
 - Lyrtech ADP



VHS-ADC	VHS-DAC	SMQUAD-4	DRC
ADCs: <ul style="list-style-type: none"> ▪ AD6645 (8x) sampling rate 105 MSPS, 14-bit resolution Control & pre-processing: <ul style="list-style-type: none"> ▪ Virtex-4 XC4VLX160 FPGA ▪ 128-MB SDRAM Off-board I/O: <ul style="list-style-type: none"> ▪ RapidCHANNEL TX & RX, 1 GBps, full-duplex 	DACs: <ul style="list-style-type: none"> ▪ DAC5687 (4x) sampling rate 480 MSPS (14-bit resolution) Control & pre-processing: <ul style="list-style-type: none"> ▪ Virtex-4 XC4VLX160 FPGA ▪ 128-MB SDRAM Off-board I/O: <ul style="list-style-type: none"> ▪ RapidCHANNEL TX & RX, 1 GBps, full-duplex 	FPGA devices: <ul style="list-style-type: none"> ▪ 2 Xilinx Virtex-4 XC4VLX160 DSP microprocessors: <ul style="list-style-type: none"> ▪ 4 TMS320C6416 DSPs SDRAM memories: <ul style="list-style-type: none"> ▪ 128MB per DSP/FPGA Off-board I/O: <ul style="list-style-type: none"> ▪ RapidCHANNEL TX & RX, 1 GBps, full-duplex On-board inter-FPGA bus: <ul style="list-style-type: none"> ▪ LYRIO 1-GBps (1 RX , 1 TX) 	FPGA device: <ul style="list-style-type: none"> ▪ Xilinx Virtex-4 XC4VSX35 ▪ Onboard flash PROM Off-board I/O: <ul style="list-style-type: none"> ▪ RapidCHANNEL TX & RX, 1 GBps, full-duplex On-board inter-FPGA bus: <ul style="list-style-type: none"> ▪ LYRIO 1-GBps (1 RX , 1 TX)

The GEDOMIS[®] testbed (III)

- RF section
 - Upconversion → Agilent E4438C ESG VSG
 - Also off-line prototyping (arbitrary waveform generator)
 - Downconversion → MCS RF 3000T (4 channels)

Equipment	Main specifications
 Agilent ESG4438C VSG	250 kHz to 6 GHz 80 MHz bandwidth +17 dBm output power <-134 dBc phase noise at 20 kHz offset ±1 ppm internal reference accuracy
 MCS Echotek Series RF 3000T Tuners	20 MHz to 3 GHz 65 MHz bandwidth Manual gain control 85 dB <-115 dBc phase noise at 10 kHz offset ±0.5 ppm internal reference accuracy

The GEDOMIS[®] testbed (IV)

- Provision of realistic signal conditions
 - EB Propsim C8 Channel Emulator
 - Real-time standard/custom channels, up to 4x4 MIMO
 - AI (extremely flat) AWGN generators
 - E.g., BER vs SNR testing



Equipment	Main specifications
EB Propsim C8 Channel Emulator	350 MHz to 6 GHz 70 MHz bandwidth Up to 48 fading paths per channel Propagation delay up to 6.4 ms Mobile speed up to 40,000 km/h
AI NS-3 RF Noise Source	5 MHz to 2.15 GHz 30 dB range with 0.1 dB steps -90 dBm/Hz maximum output power ± 2.0 dB flatness over full operating range

The GEDOMIS[®] testbed (V)

- Other specialized equipment
 - Clock generation → Holzworth microwave sources



Equipment	Main specifications
HSC1001A RF synthesizer	8 MHz to 1 GHz 0.001 Hz resolution -110 to +15 dBm output power range <-131 dBc phase noise at 10 kHz offset Internal reference 100 MHz ±1 ppb internal reference accuracy
HSM1001A RF synthesizer	250 kHz to 1 GHz 0.001 Hz resolution -70 to +10 dBm output power range <-133 dBc phase noise at 10 kHz offset Internal reference 100 MHz ±1 ppb internal reference accuracy External reference input 10/100 MHz



- Signal analysis → oscilloscope & spectrum analyzer



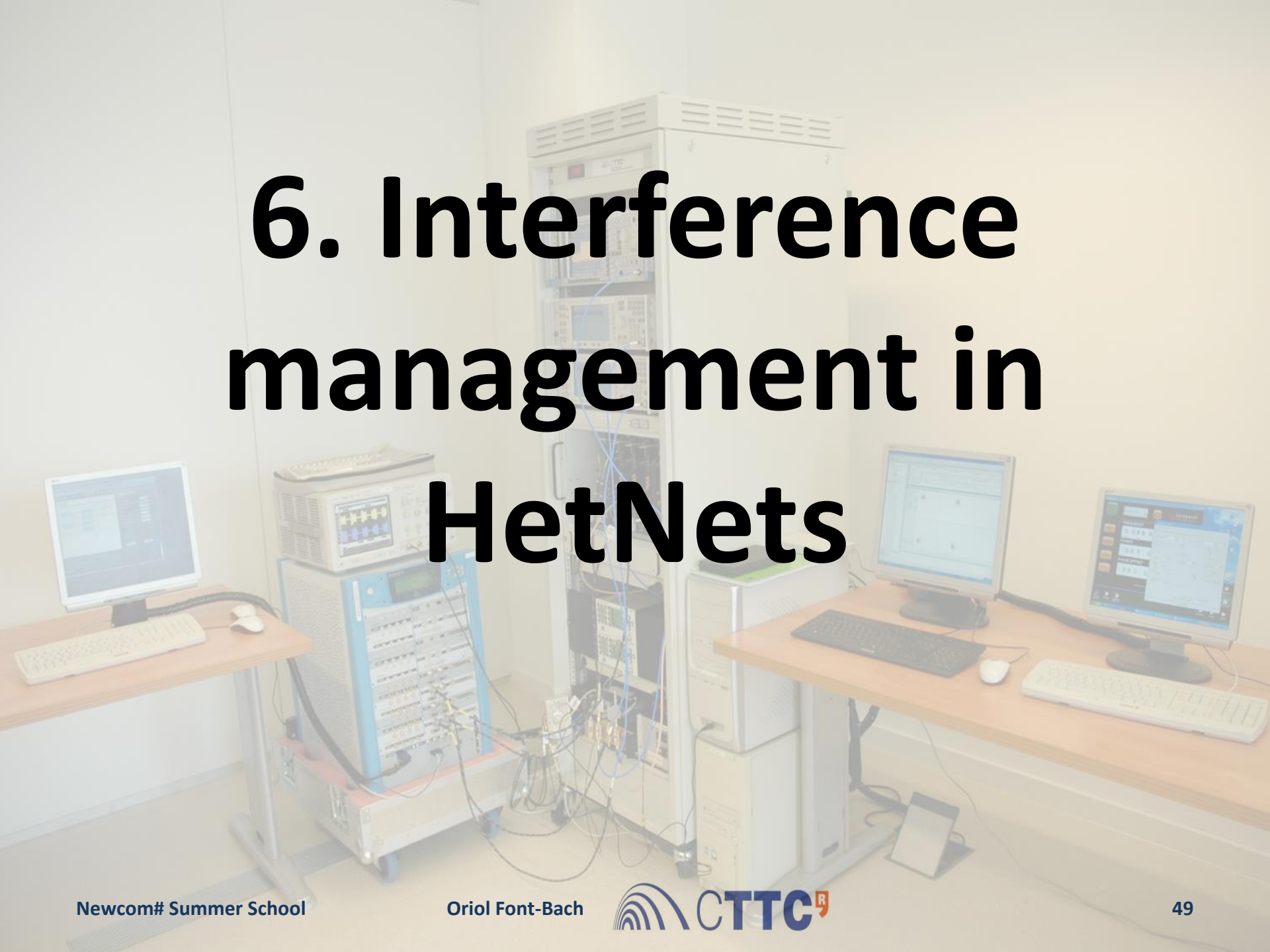
Equipment	Main specifications
R&S FSQ Signal Analyzer	20 Hz to 26.5 GHz 120 MHz bandwidth -173 dBm displayed average noise level 235 MSa I/Q memory <-133 dBc phase noise at 10 kHz offset
Agilent DSO80804B Infiniium Oscilloscope	4 analog channels 10 GHz bandwidth 40 GSaPS Noise floor 294 μ V (5 mV/div)



Signal impairments resulting from the utilization of GEDOMIS[®]

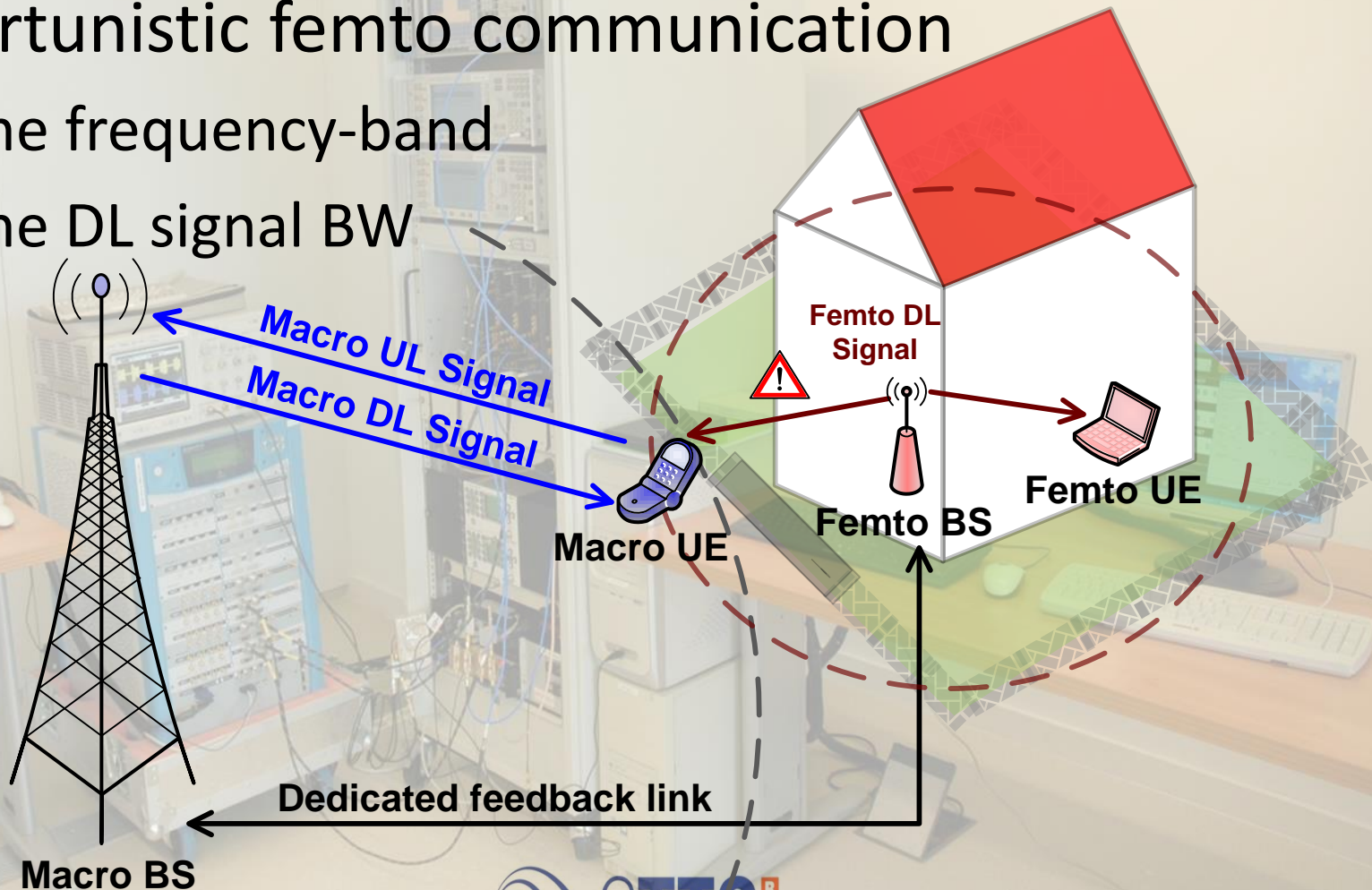
- High-end RF equipment:
 - Negligible I/Q phase/gain imbalances
 - CFO can be accurately generated
- High precision clock synthesis equipment:
 - It can be safely ignored: effects of inaccuracy between sampling clock at Tx/Rx in respect to exact sampling frequency, LO drifts/instability
 - LO coupling at RF transmitter still needs to be accounted → it is converted to an in-band sinusoid
- Extremely flat AWGN generator:
 - Precise control of noise level
- The chassis of the ADP introduces a DC signal:
 - Out-of-band signal which needs to be filtered in the digital domain
- Channel emulator:
 - Allows the reproduction of standard and custom channels (e.g., mobility conditions, interference)

6. Interference management in HetNets



Scenario definition

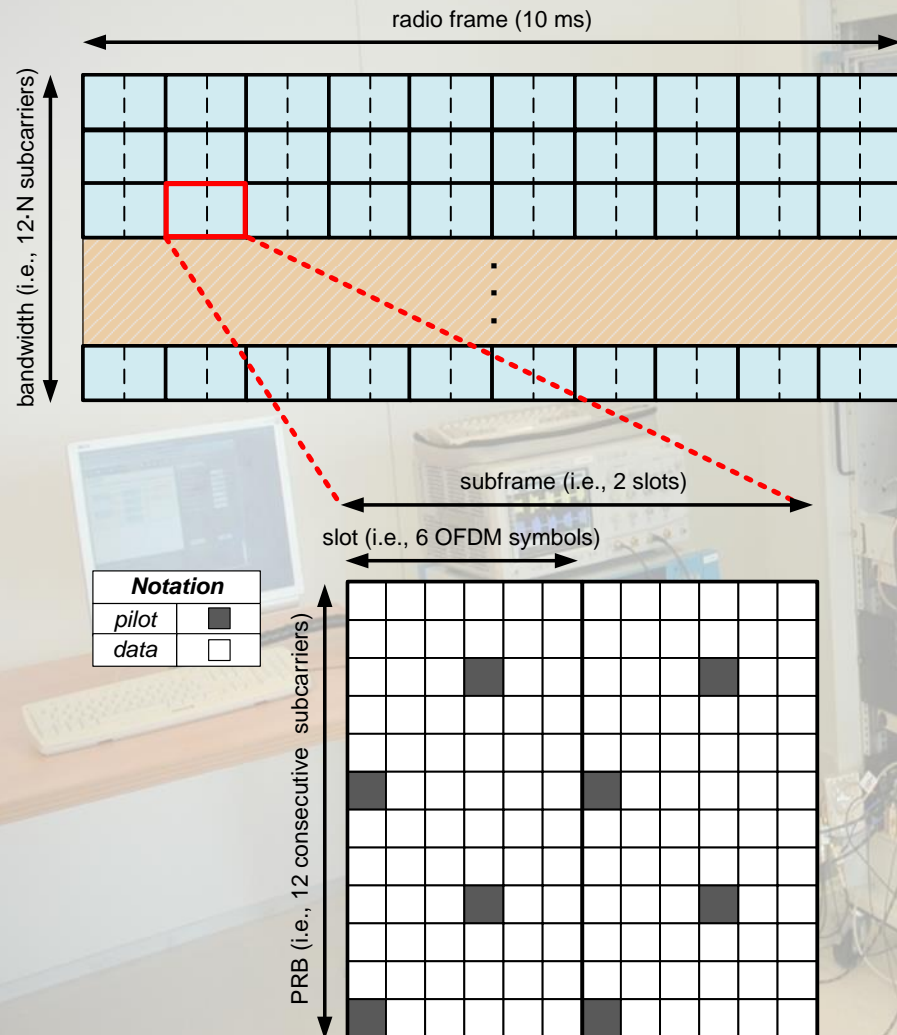
- Opportunistic femto communication
 - Same frequency-band
 - Same DL signal BW



System specifications

Parameter	Value
Wireless telecommunication standard	3GPP LTE (Rel. 9)
Antenna scheme: SISO	1x1
Channel bandwidth (MHz)	20
Cyclic prefix (samples)	512 (1/4 of the symbol)
Modulation type	QPSK
Duplex mode	FDD
Active subcarriers per OFDM symbol	1200
Null subcarriers per OFDM symbol	848
FFT size	2048
OFDM symbols per frame: total active	120 83
Closed-loop transmission scheme	Interference-aware PRB allocation
ADC sampling frequency (MHz)	61.44
Baseband sampling frequency (MHz)	30.72
RF band (GHz)	2.6
IF (MHz)	46.08
Tested channel model	ITU Ped. B (up to 3 km/h)

3GPP LTE (Rel. 9; FDD)



- OFDM symbols are organized in Physical Resource Blocks (**PRBs**)
 - Nº of PRBs depends on BW
 $\rightarrow 20 \text{ MHz} = 100 \text{ PRBs}$
- Slot = 6 OFDM symbols
- Subframe = 2 slots
- Frame = 10 subframes
- RSs found in one of every 3 OFDM symbols
 - 4 predefined values
 $\rightarrow \pm \frac{1}{\sqrt{2}} \pm \frac{1}{\sqrt{2}}j$

Considered SIR values

- 3GPP suburban deployment of LTE femtocells → pathloss modelling of 3 DL signals:

(1) macro BS → macro UE

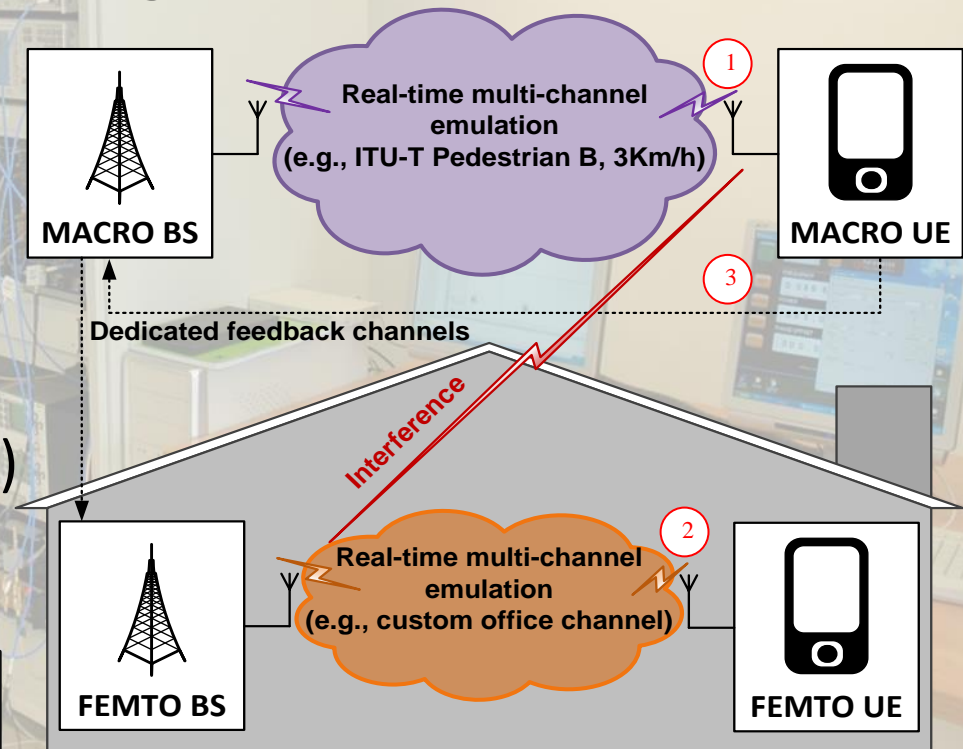
(2) femto BS → femto UE

(3) femto BS → macro UE

- SIR

- ratio between (1) and (3)
- range from 12 to 20 dB

Simulation assumptions and parameters for FDD HeNB RF requirements, 3GPP TSG RAN WG4 R4-092042.



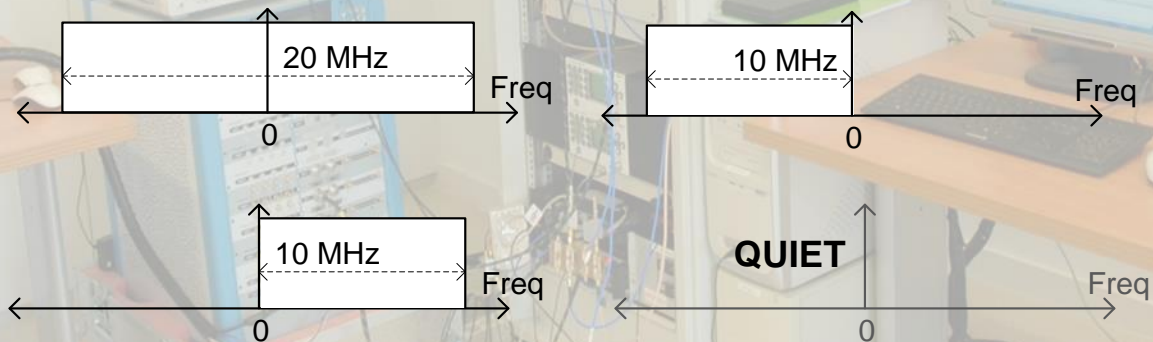
Interference-management algorithm

- *Distributed ICIC algorithm* → Victim User Aware Soft Frequency Reuse in macrocell/femtocell HetNets
 - Available BW is divided in N sub-bands
 - Instantaneous channel conditions of macro UE are exploited to adapt femto DL transmission
 - Objective: avoid interfering primary communication , while deactivating least #sub-bands in secondary DL communication

M. Shariat, A. u. Quddus, M. Bennis, Z. Bharucha, M. Lalam, M. Maqbool, S. Mayrargue, C. Kosta, A. De Domenico, E. Calvanese-Strinati, R. Mahapatra, C. H. M. de Lima and S. Uygungelen, "Promising Interference and Radio Management Techniques for Indoor Standalone Femtocells", *Deliverable D3.2, ICT 248523 FP7 Broadband Evolved FEMTO Network (BeFEMTO) Project*, Jun. 2012.

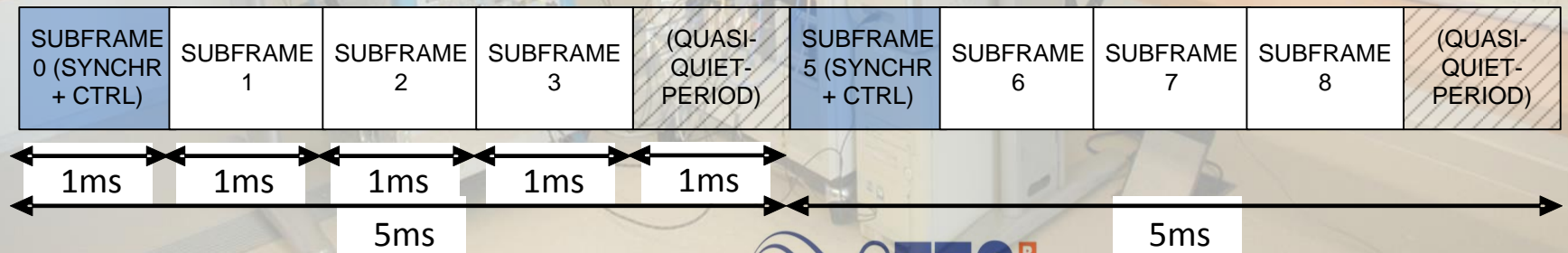
Scaling the scenario to fit the proof-of-concept (I)

- Baseline interference-management algorithm
 - 1 macro BS-UE pair & 1 femto BS-UE pair
 - 20 MHz BW → two 10 MHz bands
 - 4 pre-defined femto PRB allocation cases

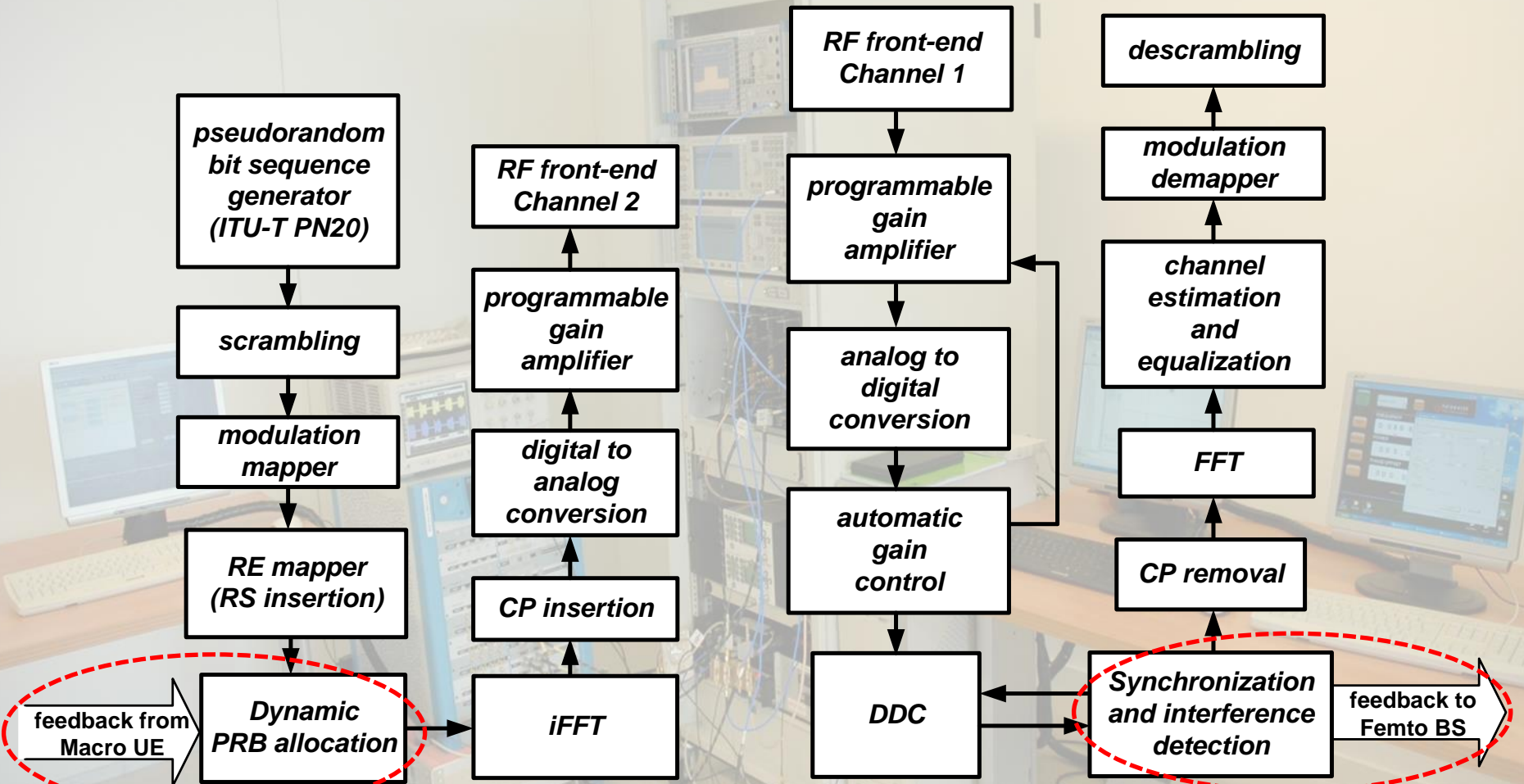


Scaling the scenario to fit the proof-of-concept (II)

- PHY-layer specifications
 - Point-to-point DL communication
 - Emulated UL → real-time intra-FPGA link
 - Fixed frame format
 - 10 ms radio frame divided in two 5-ms, separated by quasi-quiet periods (i.e., no data, only RSs)
 - facilitate vital DFE processes → gain adjustment, CFO correction



System modelling (I)



macro BS

femto UE **Joint design of synchrhonization & interference-detection techniques**

System modelling (II)

- Received signal model (considering the utilization of GEDOMIS®):

$$c(t) = \Re\{x(t) \cdot e^{j2\pi(f_{IF} + \Delta f)t}\} + \Re\{u(t) \cdot e^{j2\pi(f_{IF} + \Delta f_u)t}\} + A + B \cdot \cos(2\pi(f_{IF} + \Delta f)t + \varphi) + w(t),$$

- $x(t)$: useful part of received baseband signal
- $u(t)$: (asynchronous) interference signal
- f_{IF} : IF (46.08 MHz) / Δf : CFO / Δf_u : CFO interf.
- A : DC level introduced by baseband boards
- $B \cdot \cos(2\pi(f_{IF} + \Delta f)t + \phi)$: unwanted in-band residual carrier → LO coupling
- $w(t)$: Gaussian noise

Synchronization/interference-detection techniques (I)

- CP-based synchronization → cross-correlation exploiting the self-similarity of the received OFDM symbols due to CP:
 1. Far less complex implementation than technique based on PSS/SSS
 2. Cross-correlation values can be opportunistically reused to detect interference → degradation directly related to SIR
 3. Design favouring resource-reuse → required for its FPGA implementation!

Synchronization/interference-detection techniques (II)

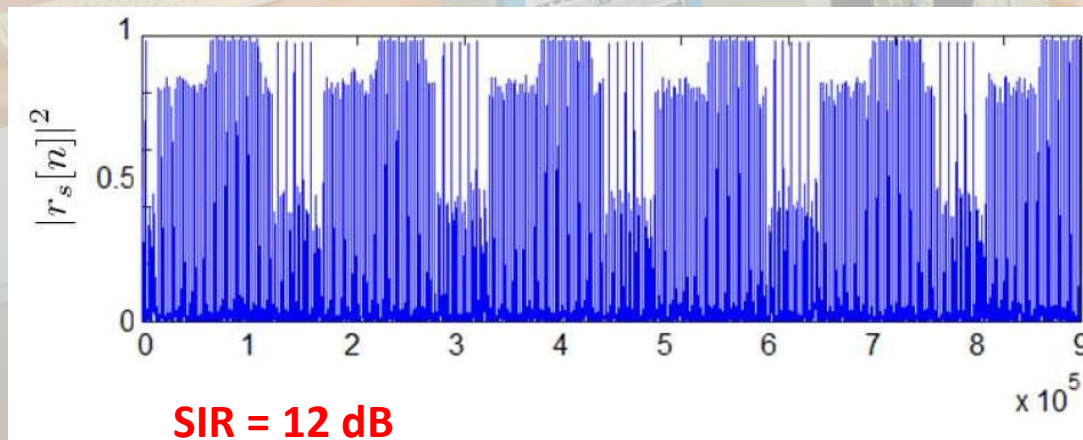
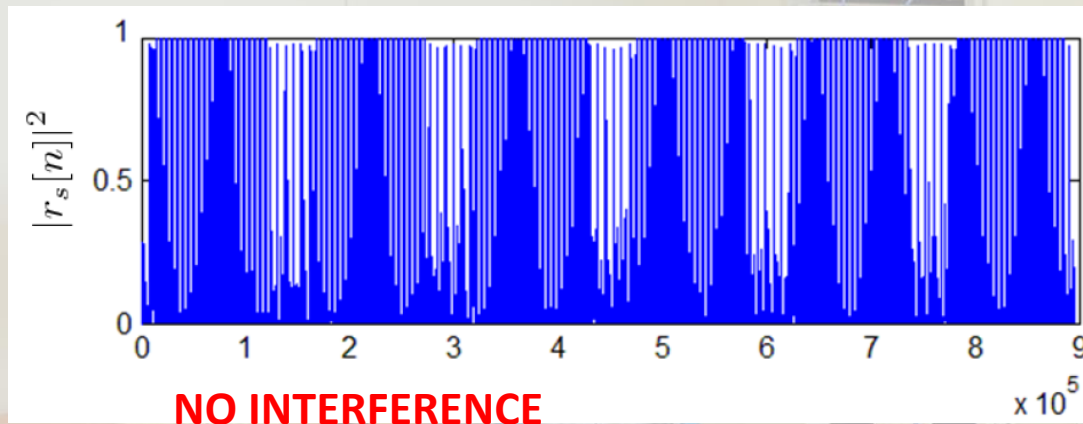
- ITU pedestrian B channel → cross-correlation using a 2048+467 sample-window:

$$|r_s[n]|^2 = \frac{|\sum_{l=0}^{466} s^*[n+l] \cdot s[n+l+2048]|^2}{(\sum_{l=0}^{466} |s[n+l]|^2) \cdot (\sum_{l=0}^{466} |s[n+l+2048]|^2)}$$

- Peak of $|r_s[n]|^2$ indicates position of CP → location of FFT-window
- Phase of $r_s[n]$ can be used to estimate the phase shift of the received signal in the presence of CFO

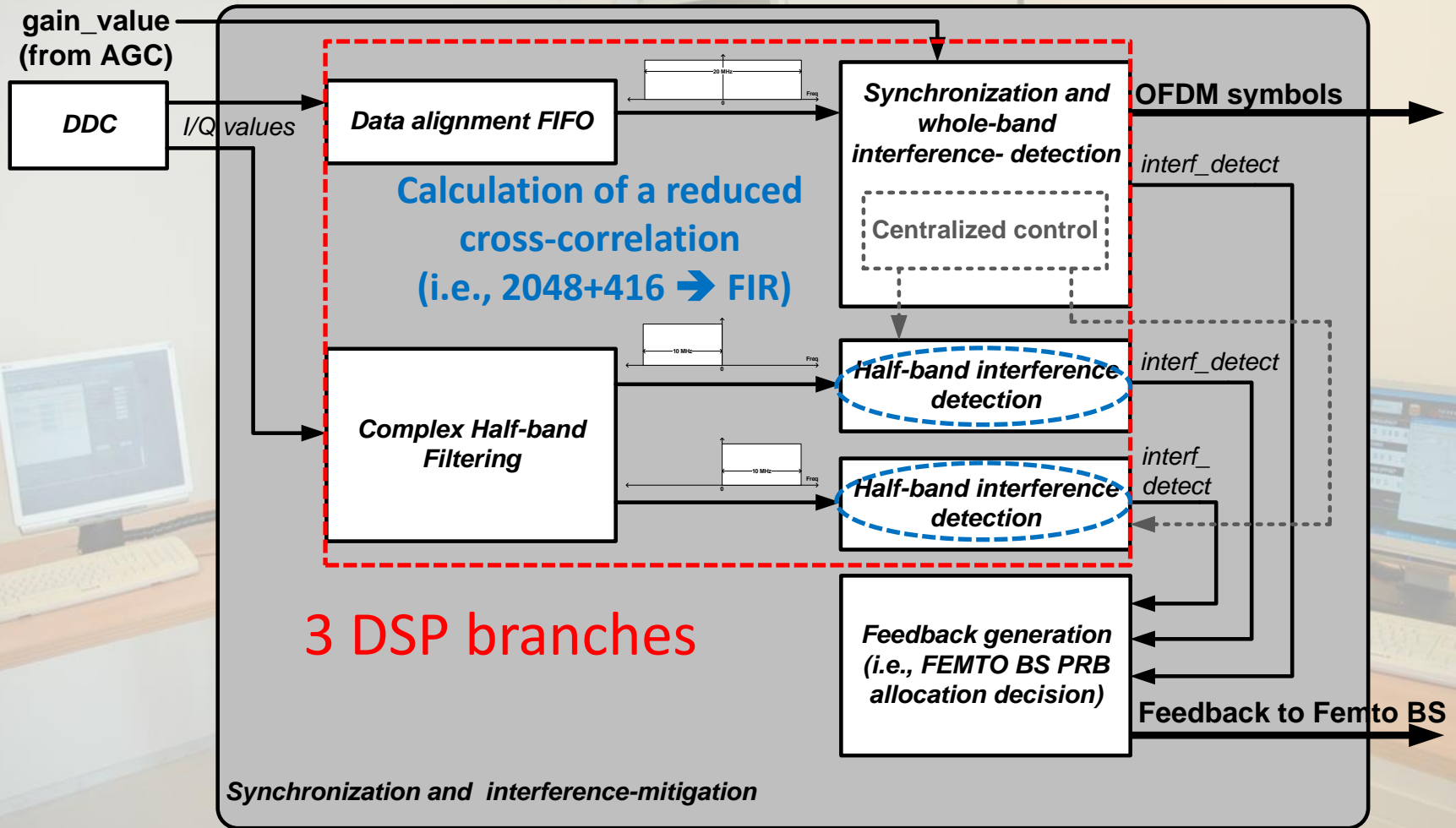
Synchronization/interference-detection techniques (III)

- Ideally (i.e., no noise and no interference) → peak-amplitude of $|r_s[n]|^2 = 1$, but...



... the cross-correlation profile is degraded in the presence of noise and interference.

General DFE architecture



Interference-detection algorithm

- Algorithm applied to each 5-ms frame → decides which band(s) are interfered

Algorithm 1

```
if wholeband_detection == 0 then
    decision = no interference;
else
    if low_10MHz_band_detection == 1 and high_10MHz_band_detection == 0 then
        decision = interference detected in the low 10 MHz band;
    else if low_10MHz_band_detection == 0 and high_10MHz_band_detection == 1 then
        decision = interference detected in the high 10 MHz band;
    else
        decision = interference detected in the entire bandwidth;
    end if
end if
```

How is interference detected?

- Amount of degradation is directly related to power of received interference → presence of interference can be detected by defining a threshold (i.e., peak-value below threshold = interference)
- **Threshold** definition aims at **fulfilling a KPI**:
 - Probability that raw/uncoded BER is below 10^{-2} must be above 0.8 (conditioned on the fact that interference is detected)

Thresholds definition (I)

- Exhaustive MATLAB simulations

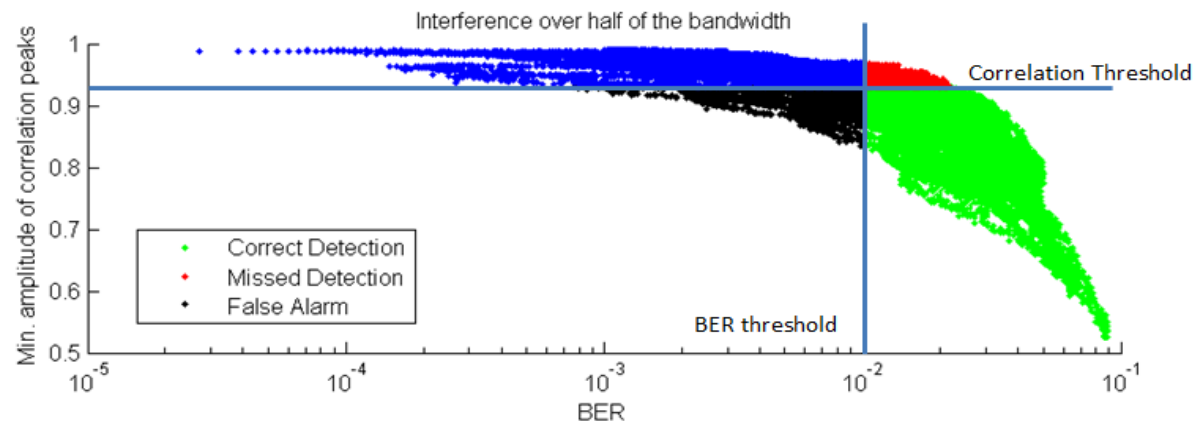
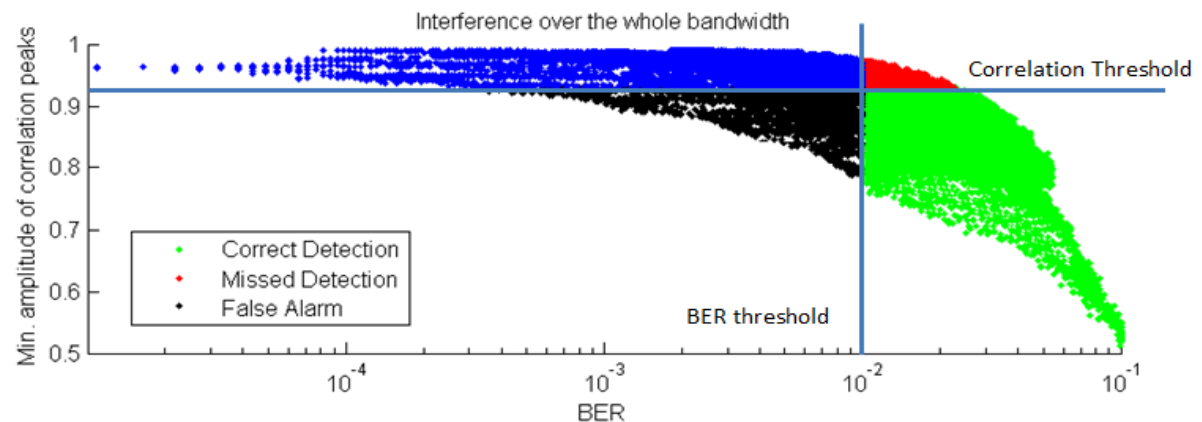
- Step 1)

- all-synthetic signals

- Step 2)

- data recorded using GEDOMIS

1. load MATLAB-generated BSs' I/Q vectors to VSGs
2. configuration of channel emulator
3. real-time signal reception & data capturing
4. off-line simulation of MATLAB UEs



Thresholds definition (II)

Threshold of main branch	Interference over the whole bandwidth		Interference over half of the bandwidth	
	Prob. of detection	Prob. of false alarm	Prob. of detection	Prob. of false alarm
0.88	0.46	0.04	0.48	0.02
0.89	0.53	0.05	0.56	0.03
0.90	0.60	0.07	0.63	0.04
0.91	0.68	0.09	0.71	0.06
0.92	0.77	0.12	0.81	0.09
0.93	0.87	0.17	0.89	0.15
0.94	0.93	0.26	0.95	0.25

Threshold of secondary branch	Interference over the whole bandwidth		Interference over half of the bandwidth	
	Prob. of detection	Prob. of false alarm	Prob. of detection	Prob. of false alarm
0.88	0.37	0.04	0.73	0.11
0.89	0.43	0.05	0.78	0.13
0.90	0.49	0.06	0.83	0.18
0.91	0.56	0.08	0.88	0.24
0.92	0.64	0.10	0.92	0.31
0.93	0.73	0.14	0.95	0.39
0.94	0.82	0.19	0.98	0.49

7. RTL design



Extended DFE

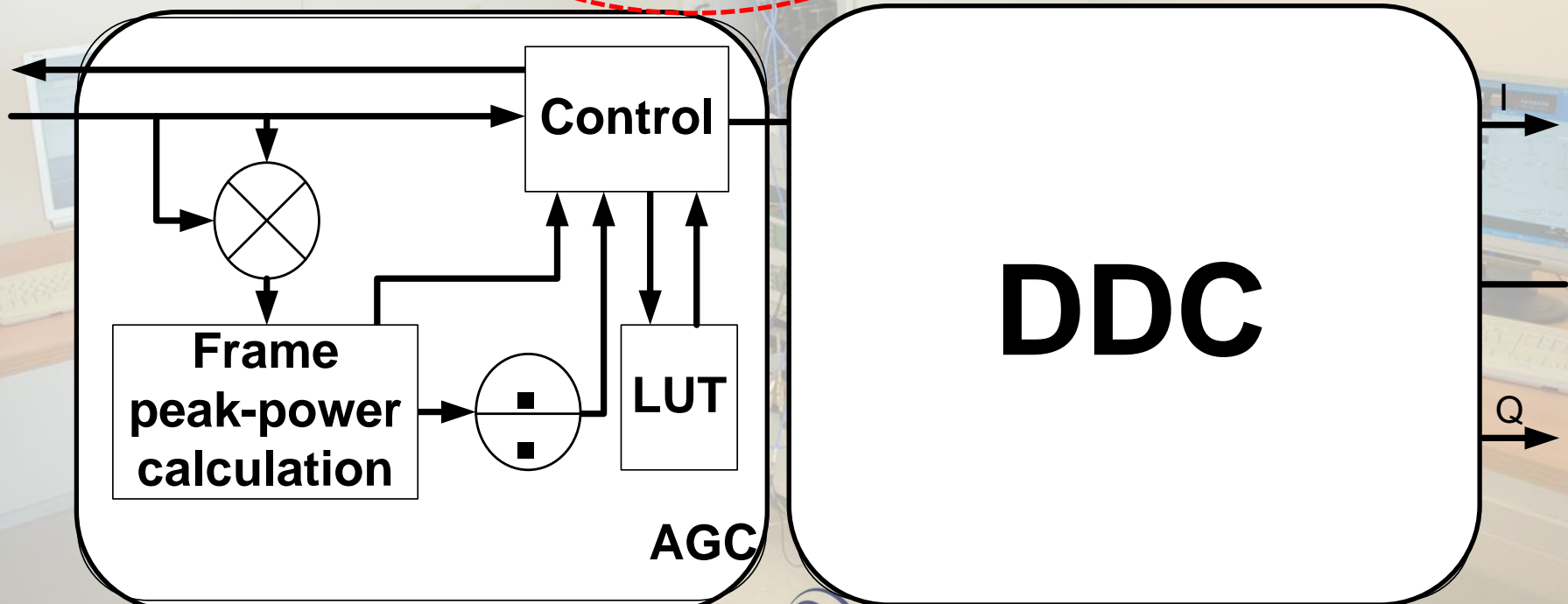
- The focus is set on the interference-aware DFE of the macro UE
 - It is one of the most complex processing blocks in the PHY-layer of the presented system
 - It has a critical impact on the performance of the whole interference-management scheme

AGC and DDC blocks (I)

- AGC configures PGA →
 - 15 gain steps, GS (1.5 dB)

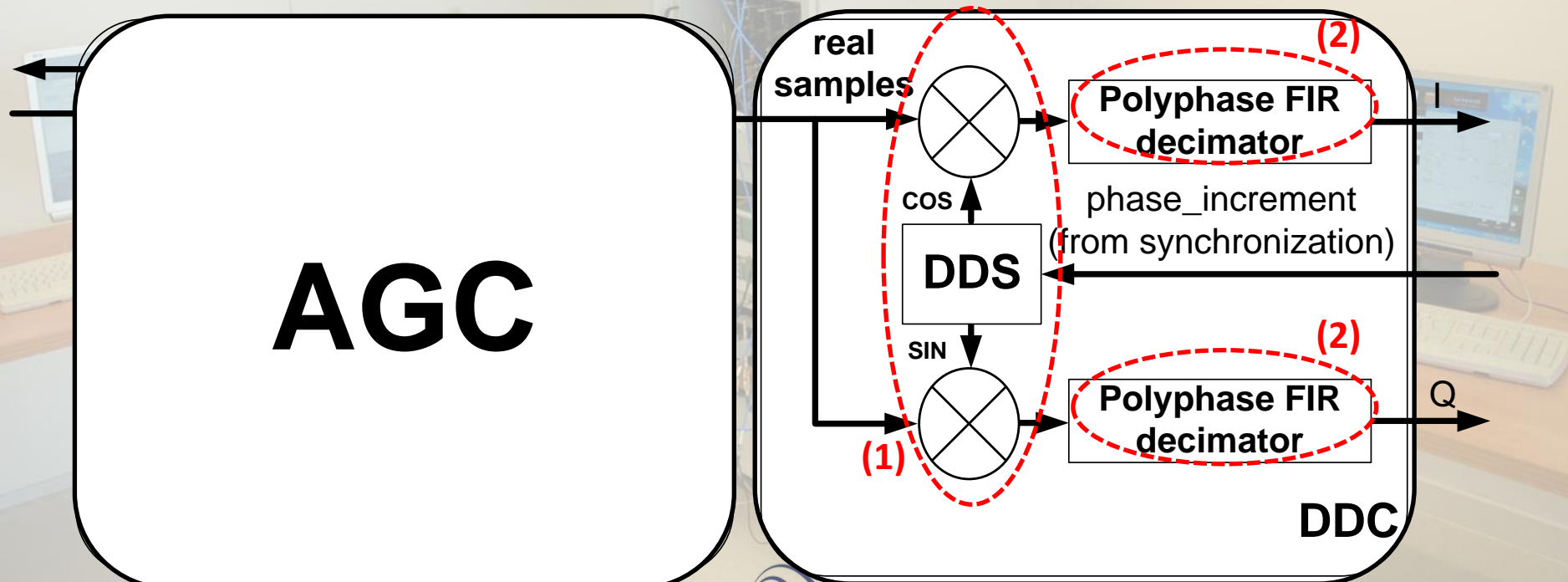
$$G = \begin{cases} G_{max}, & \text{if } G_{prev} + n \cdot GS > G_{max} \\ G_{prev} + n \cdot GS, & \text{if } G_{min} \leq G_{prev} + n \cdot GS \leq G_{max} \\ G_{min}, & \text{if } G_{prev} + n \cdot GS < G_{min} \end{cases}$$

$$n \cdot GS = 10 \cdot \log_{10} \left(\frac{\frac{Q_{2.14}}{backoff}}{frame_peak_power} \right) \rightarrow \text{LUT relates } n \cdot GS \text{ and } g$$



AGC and DDC blocks (II)

- DDC (using various Xilinx IP cores):
 - (1) frequency translation
 - (2) I/Q components extraction + decimation
 - MATLAB FDA tool



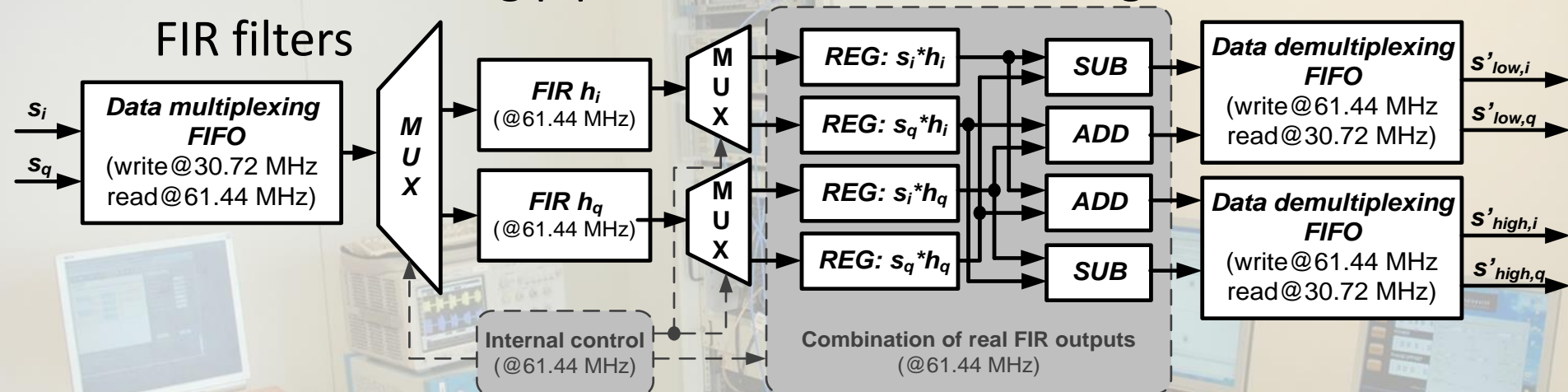
Hardware-efficient filtering stage (I)

- Xilinx FIR filter IP core
 - Direct link to MATLAB Fdatool → 51 18-bit complex-valued symmetric coefficients
 - ... but only accepts real-valued coefficients!
 - A single filter requires a large amount of DSP and regular FPGA slices... we would need 4!
- Design exploits fact that the coefficients of the required filters are the complex conjugate of each other:

$$\begin{aligned}h_{\text{low}}[n] &= h_i[n] + j \cdot h_q[n] \\h_{\text{high}}[n] &= h_i[n] - j \cdot h_q[n]\end{aligned}$$

Hardware-efficient filtering stage (II)

- Resource-sharing pipelined architecture, using two 2-channel FIR filters



- 1) A new I/Q value is received each 32.55 ns $\rightarrow s[n] = s_i[n] + j * s_q[n]$
- 2) multiplex/demultiplex \rightarrow FIFOs with independent R/W clocks
- 3) 32.55 ns time-slots are divided:
 - a) 1st half used to process $s_i[n]$
 - b) 2nd half used to process $s_q[n]$
- 4) Outputs are combined

$$\begin{aligned}
 s'_{\text{low},i}[n] &= s_i[n] * h_i[n] - s_q[n] * h_q[n] \\
 s'_{\text{low},q}[n] &= s_q[n] * h_i[n] + s_i[n] * h_q[n] \\
 s'_{\text{high},i}[n] &= s_i[n] * h_i[n] + s_q[n] * h_q[n] \\
 s'_{\text{high},q}[n] &= s_q[n] * h_i[n] - s_i[n] * h_q[n]
 \end{aligned}$$

Joint synchroization/interference-detection (I)

- RTL-optimized calculation of cross-correlation

$$|r_s[n]|^2 = \frac{|dn[n]|^2}{ds0[n] \cdot ds1[n]}$$

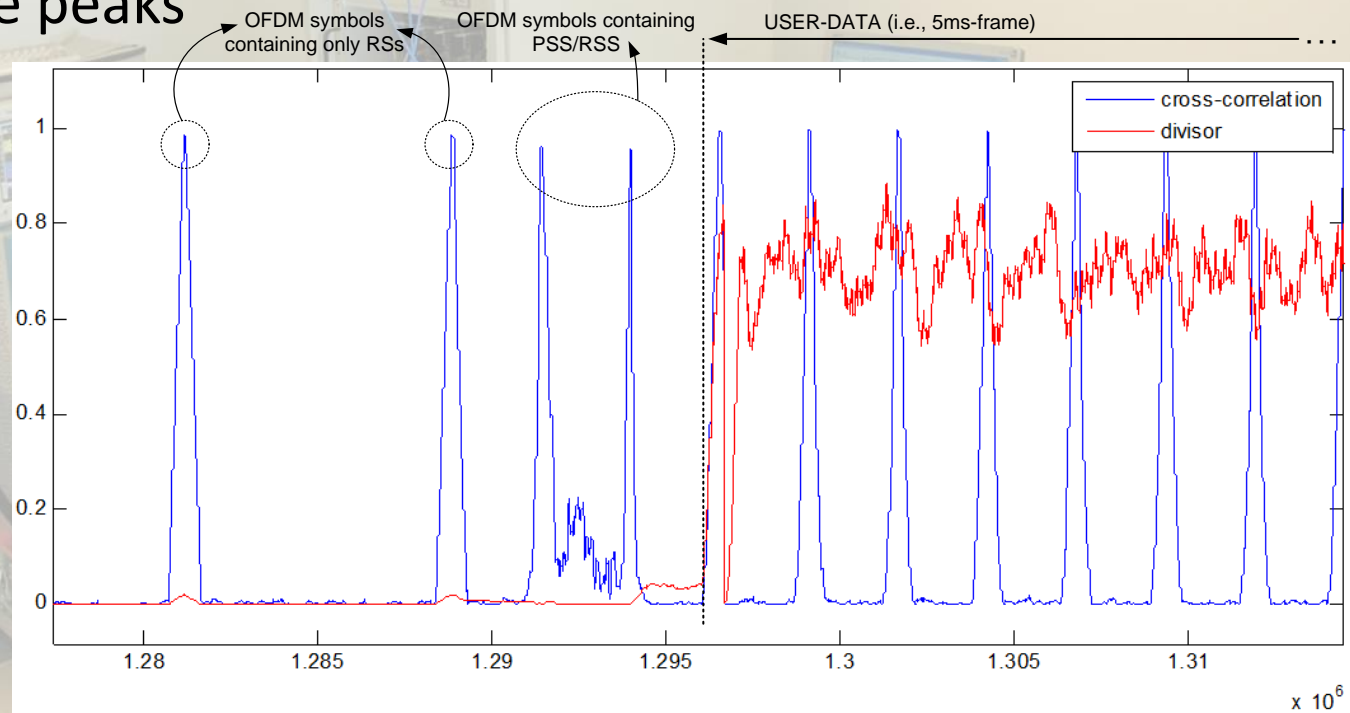


$$dn[n+1] = \begin{cases} dn[n] + s^*[n+467] \cdot s[n+2048+467] & \text{if } n \leq 467 \\ dn[n] - s^*[n] \cdot s[n+2048] \\ \quad + s^*[n+467] \cdot s[n+2048+467] & \text{if } n > 467, \end{cases}$$

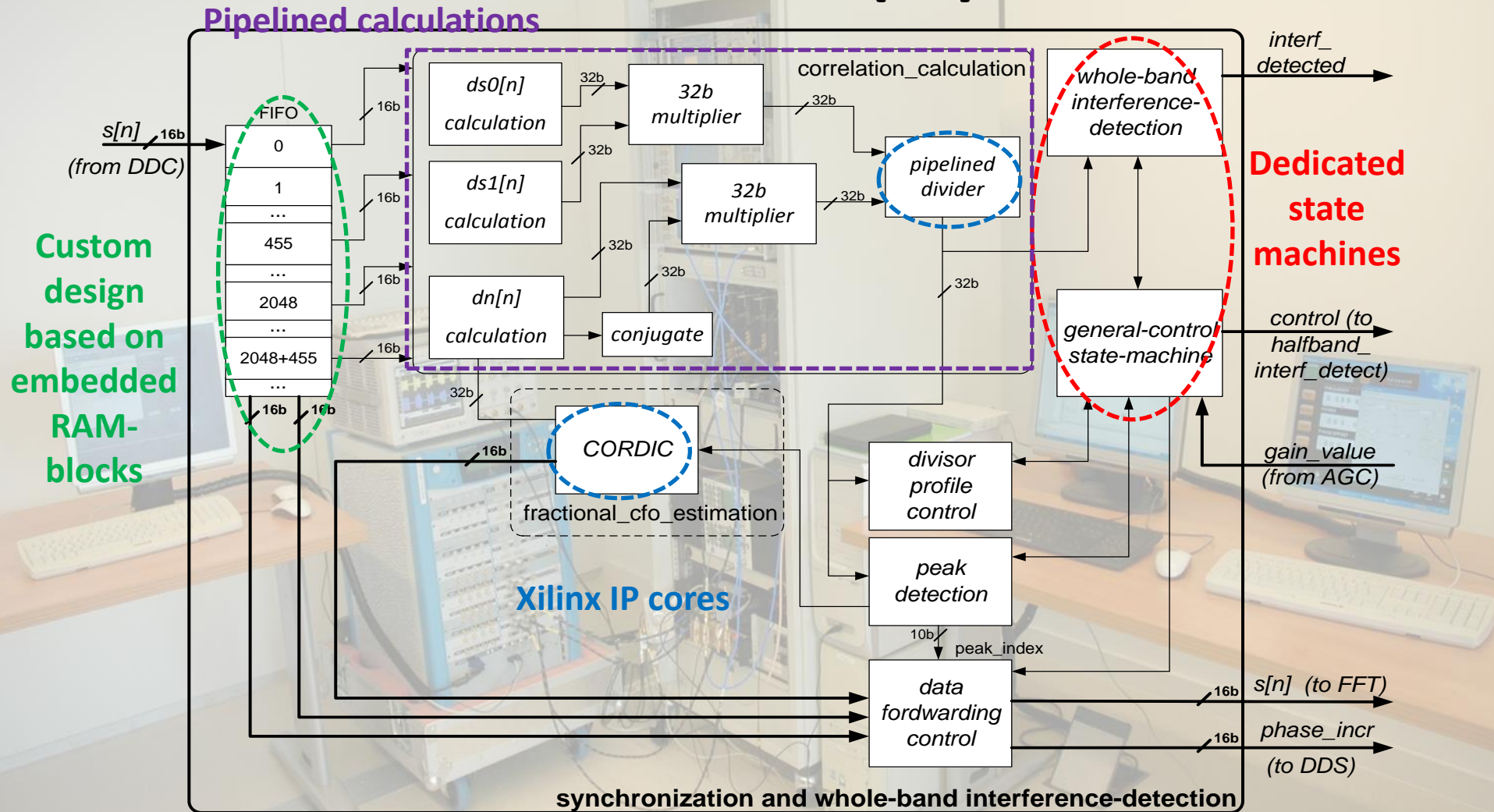
- Only four samples need to be introduced to the already calculated correlation, each clock cycle
- DSP48-slice savings!

Joint synchroization/interference-detection (II)

- Peak-detection based on triggering threshold
 - Because of RSs, peaks can be also found in the quasi-quiet periods → values of $ds0[n] \cdot ds1[n]$ are used to determine legitimate peaks

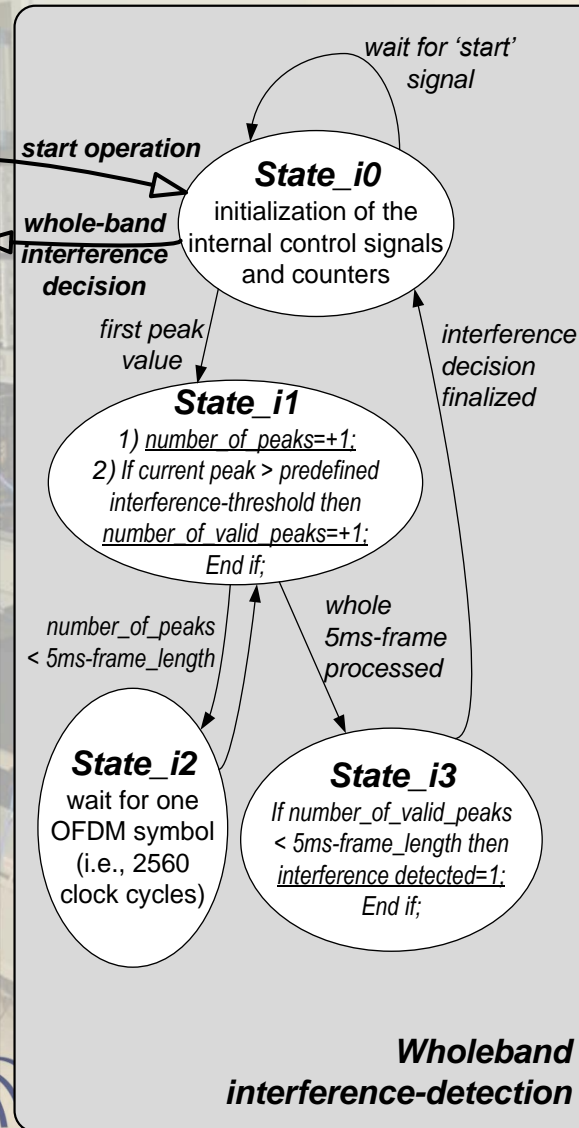
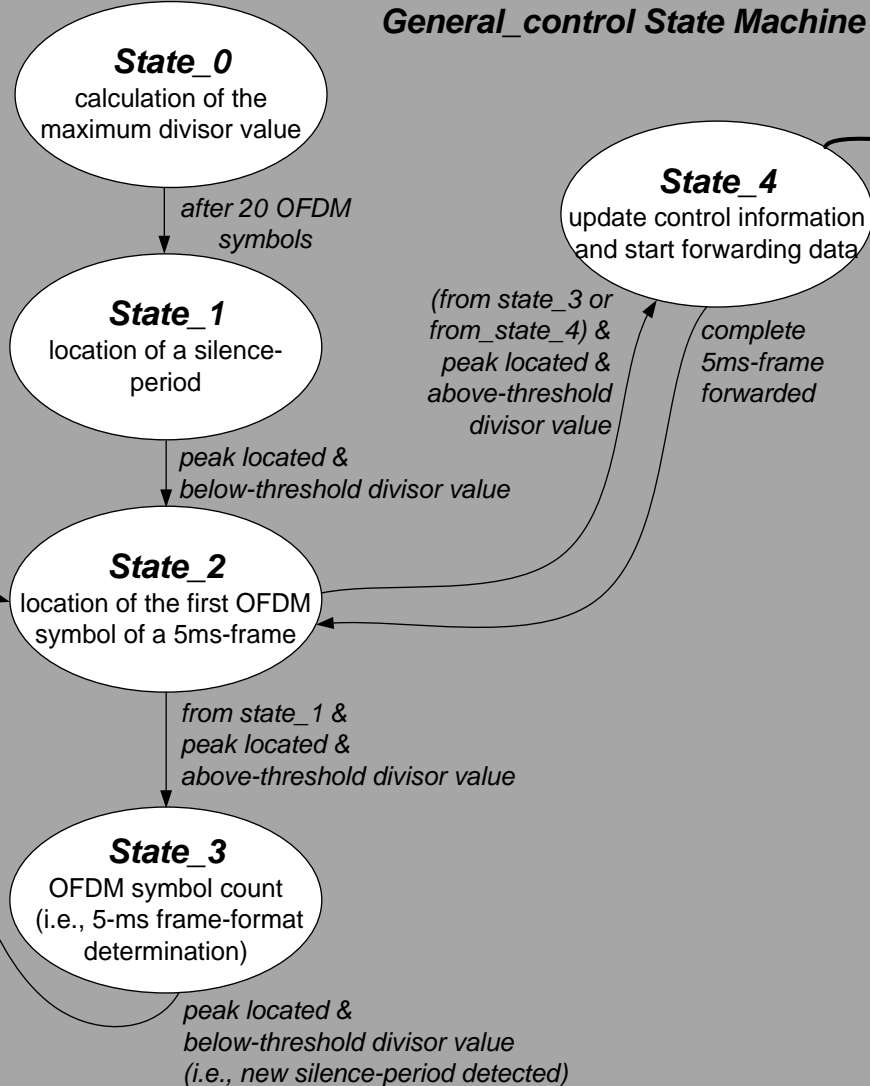


Joint synchroization/interference-detection (III)



Centralized control unit

General_control State Machine

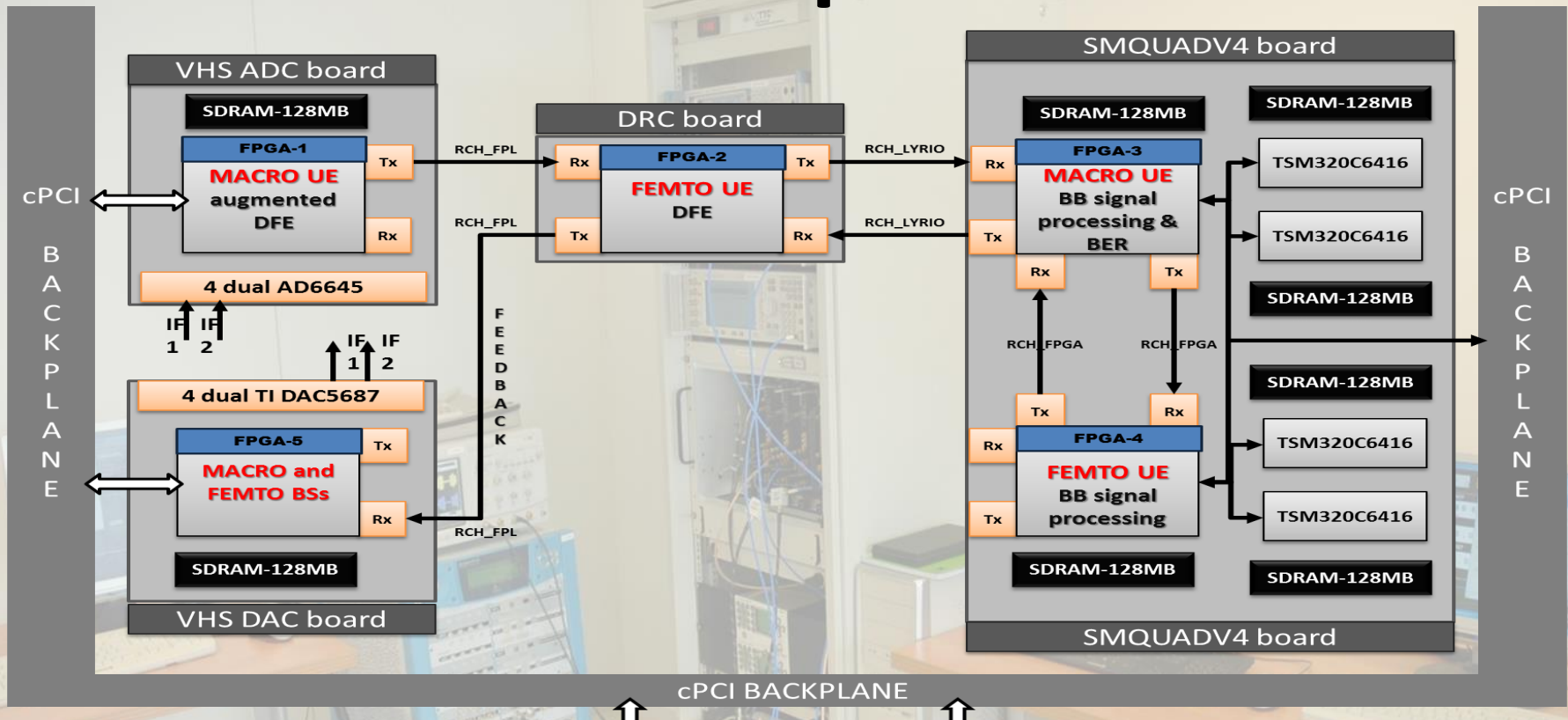


Resource-reuse → hierarchical structure of dedicated state machines

Not depicted, control of AGC (divisor values!)

8. Validation and results using the GEDOMIS[®] testbed

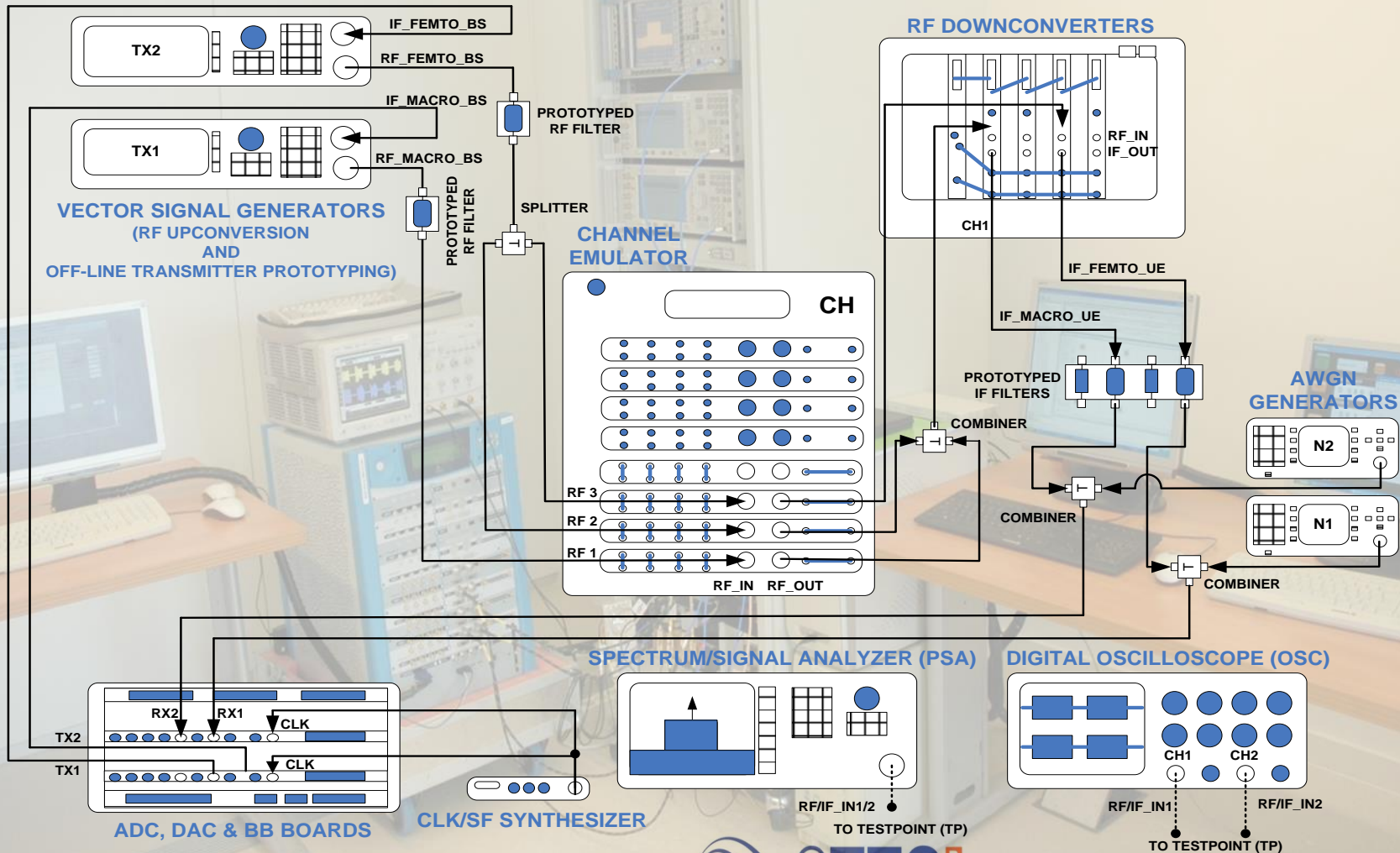
Multi-FPGA implementation



	FPGA-1	FPGA-2	FPGA-3	FPGA-4	FPGA-5
Device	<i>XC4VLX160</i>	<i>XC4VSX35</i>	<i>XC4VLX160</i>	<i>XC4VLX160</i>	<i>XC4VLX160</i>
Slices	67%	44%	33%	36%	27%
DSP48s	90%	23%	94%	52%	78%
RAMB16s	78%	35%	62%	78%	82%

Setup of GEDOMIS[®]

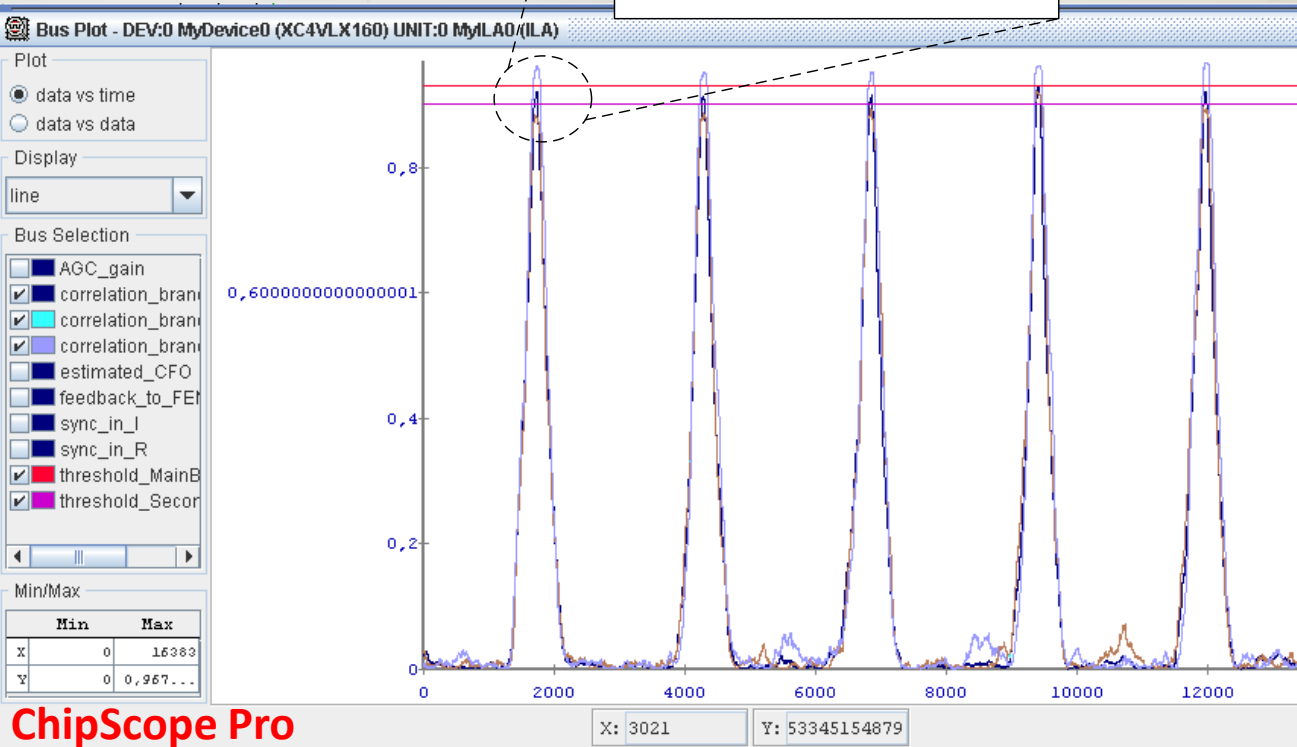
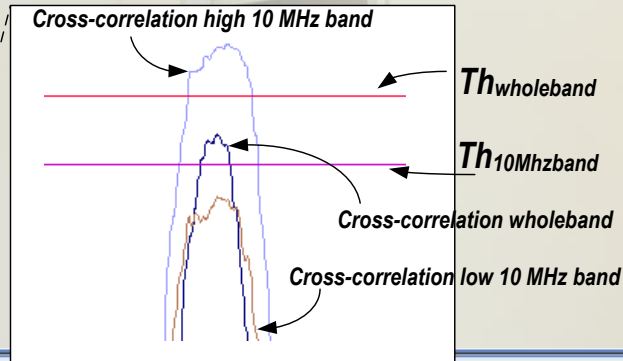
PC-BASED CONTROL & DEBUGGING PLANE



Visualization of the cross-correlation

- Interference in the low 10 MHz band
- SIR = 12 dB
- Static pedestrian B channel

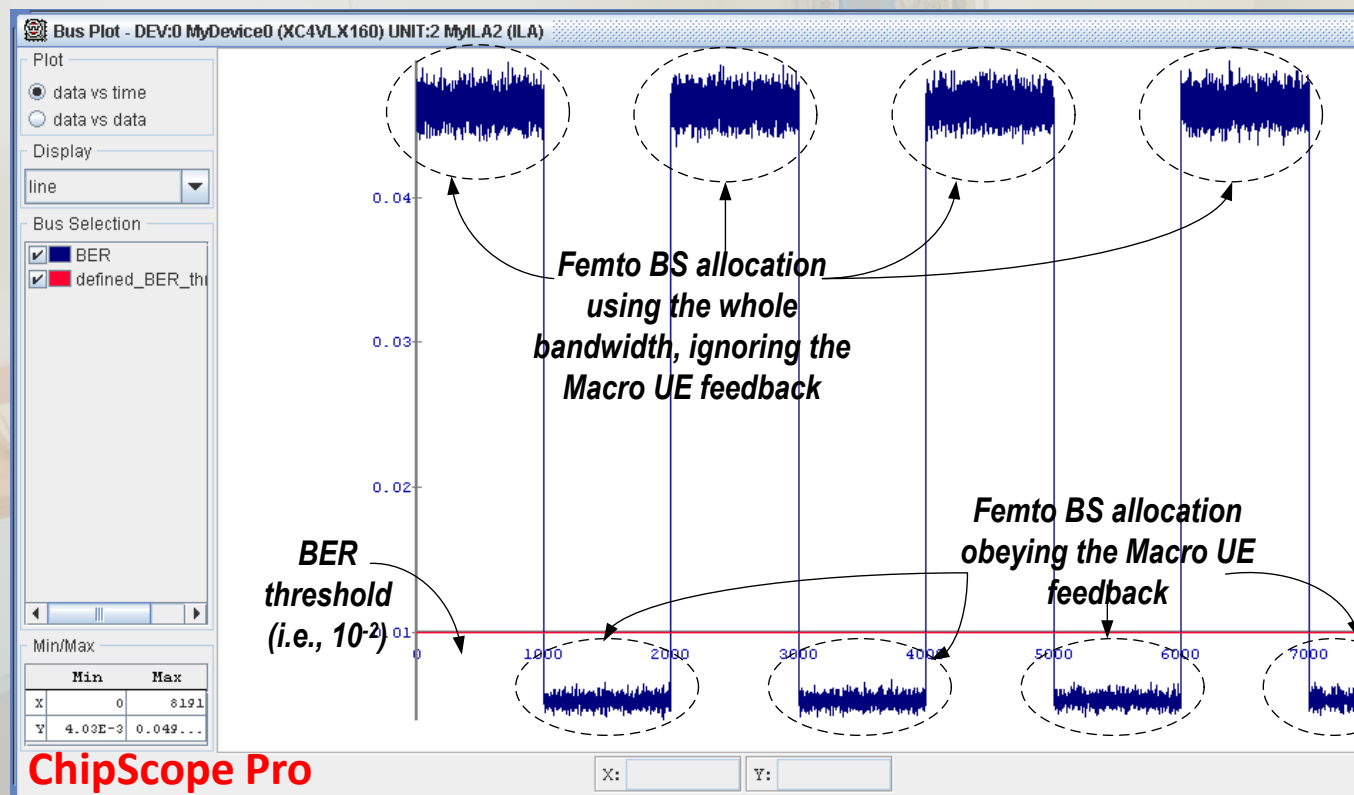
feedback_generation_inputs	
wholeband_detection=1	01 (interference detected in the low 10 MHz band)
low_10Mhz_band_detection=1	
high_10Mhz_band_detection=0	



ChipScope Pro

Visualization of the BER (I)

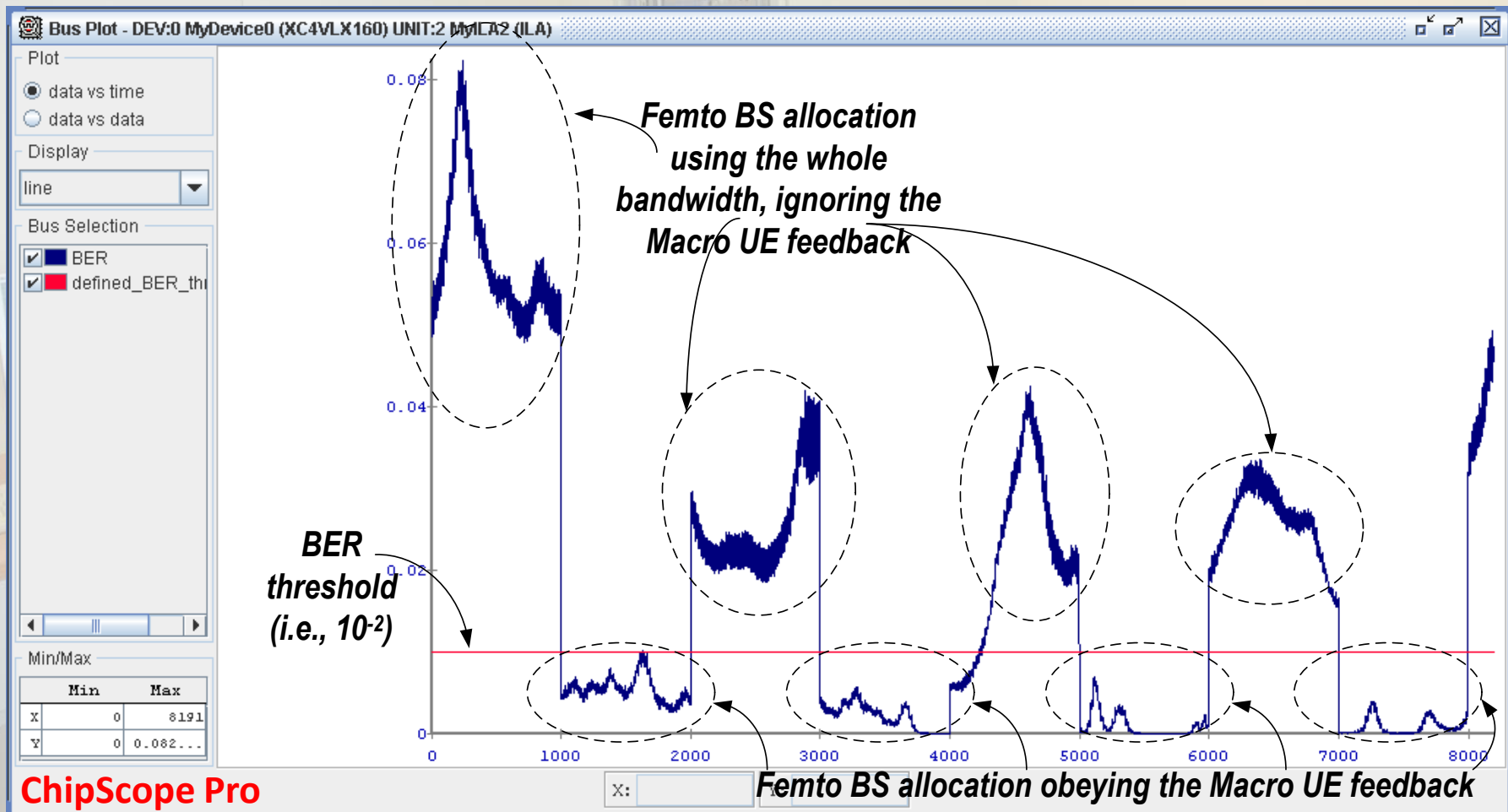
- Two transmission modes are defined for the Femto BS
 - According to feedback or ignoring it (i.e., whole 20 MHz band transmission)
 - Transmission mode changes every N seconds
- Real-time calculation of macro UE VER
 - Replication of macro BS' PRBS generator



- Interference in the whole 20 MHz band
- SIR = 10 dB
- static pedestrian B channel

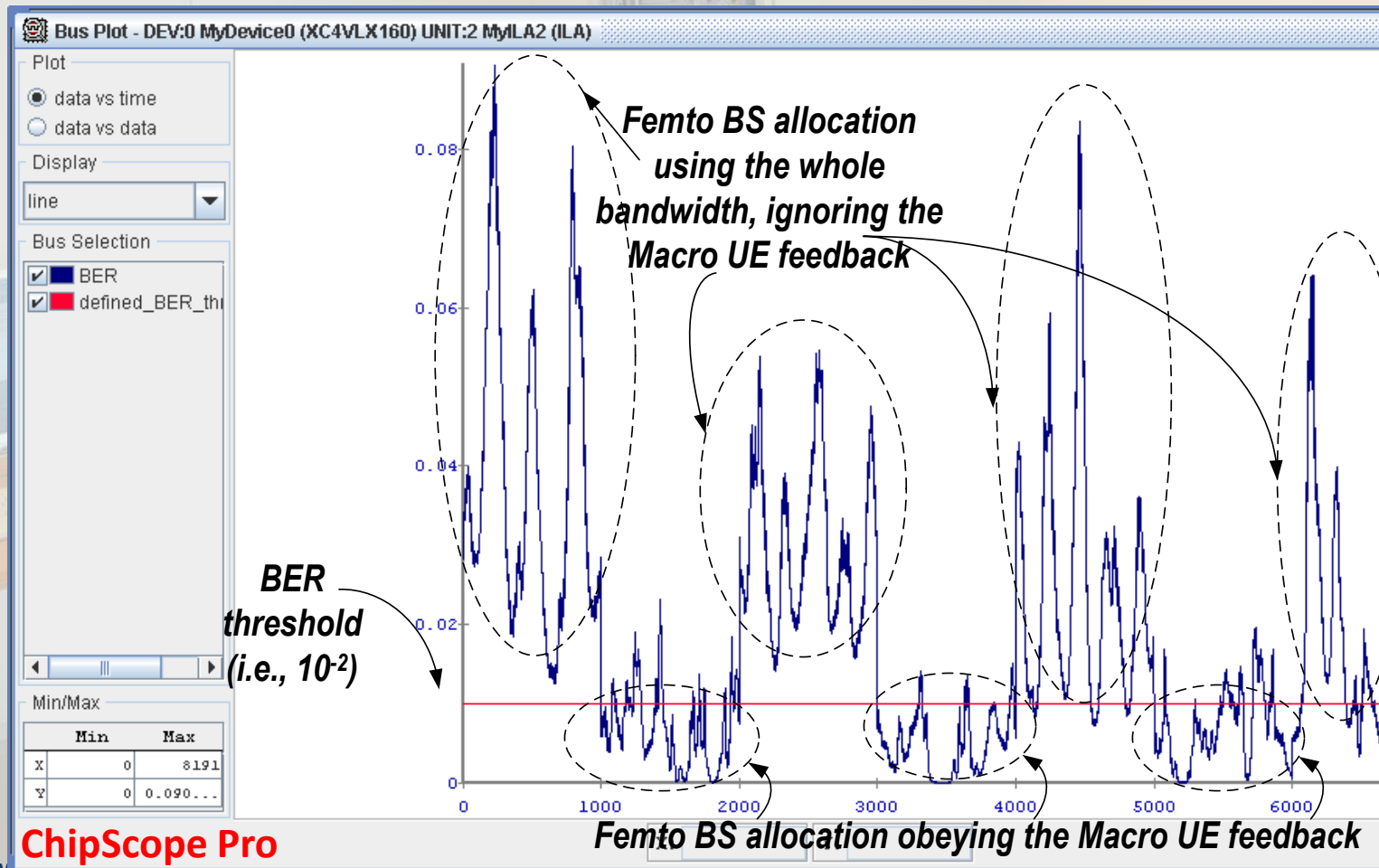
Visualization of the BER (II)

Interference in the low 10 MHz band, SIR = 12 dB, low mobility pedestrian B channel (i.e., 0.2 km/h)



Visualization of the BER (III)

Interference in the high 10 MHz band, SIR = 14 dB, mobile pedestrian B channel (i.e., 3 km/h)



ACK

Development team

- Signal processing and algorithmic
 - Antonio Pascual (UPC), Miquel Payaró (CTTC)
- High-level modelling and simulations
 - Luís Blanco & Jordi Serra (CTTC), Marc Molina (UPC)
- RTL design and VHDL coding
 - Pepe Rubio & Oriol Font (CTTC)
- Laboratory setup and debugging
 - Nikolaos Bartzoudis & David López (CTTC)

Questions?

Oriol Font-Bach

oriol.font@cttc.cat



**Centre
Tecnològic
de Telecomunicacions
de Catalunya**

