OpenAirInterface Overview and Lab Session

Florian Kaltenberger, Raymond Knopp
Eurecom

Newcom# Summer School May 2013

www.openairinterface.org

- **Provides open-source (hardware and software) wireless technology platforms**
  - target innovation in air-interface technologies through experimentation
- **We rely on the help of**
  - Publicly-funded research initiatives (ANR,ICT,CELTIC)
  - Direct contracts with industrial partners
  - Widespread collaboration with a network of partners using open-source development and tools
    - LINUX/RTAI based SW development for PCs
    - LEON3/GRLIB-based HW and eCos/MutexH-based SW development for FPGA targets
    - LINUX networking environment
  - Experimental Licenses from ARCEP (French Regulator) for medium-power outdoor network deployments
    - 1.9 GHz TDD, 5 MHz channel bandwidth
    - 2.6 GHz FDD (two channels), 20 MHz channel bandwidth
    - 800 MHz FDD (two channels) : 10 MHz channel bandwidth

OpenAirInterface Development Areas

Cognitive Technologies
Wideband RF, Agile Spectrum Management, Interference Management and Control, Distributed/Collaborative techniques, Spectrum Sensing, Cognitive and Flexible Radio Architectures, Ambient Networking

OPENAIR3 : Wireless Networking
All-IP, Mobility Management, 802.21, Cellular/Mesh Routing Protocols, Mesh Topology Management, Multimodal Radio Resource Management

OPENAIR2: Medium-Access Protocols
Cellular topologies, single-frequency resource allocation, cross-layer wideband scheduling, Mesh topologies, distributed resource control

OPENAIR1: Baseband/PHY
Advanced PHY (LTE), Propagation Measurement and Modelling, Sensing and Localization Techniques, PHY Modeling Tools

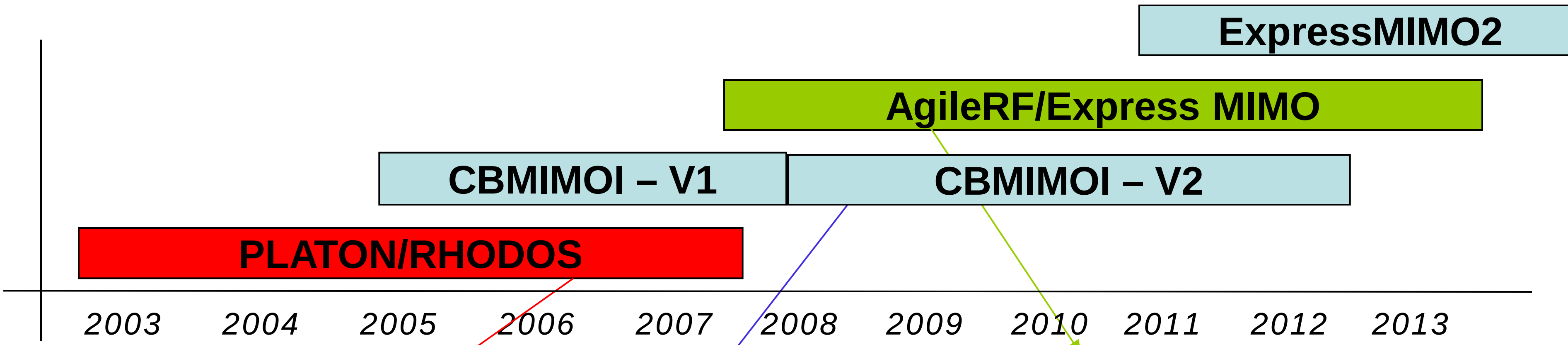
OPENAIR0: Wireless Embedded System Design
Agile RF design, Reconfigurable High-end Transceiver Architectures, FPGA prototyping, Simulation Methodologies, Software development tools, low-power chip design

Collaborative Web Tools

- **OpenAirInterface SVN Repositories**
  - All development is available through www.openairinterface.org's SVN repository (openair4G) containing
    - OPENAIR0 (open-source real-time HW/SW)
    - OPENAIR1 (open-source real-time and offline SW)
    - OPENAIR2 (open-source real-time and offline SW)
    - OPENAIR3 (open-source Linux SW suite for cellular and MESH networks)
    - TARGETS : different top-level target designs (emulator, RTAI, etc.)
  - Partners can access and contribute to our development
- **OpenAirInterface TWIKI**
  - A TWIKI site for quick access by partners to our development via a collaborative HOW-TO
- **Forum**
  - external support services (not currently used effectively)
- **Mailing list**
  - openair4G-devel@eurecom.fr

EQUIPMENT AND SW

Prototype Equipment Timeline



Cellular Systems

Pure Software Radio

WCDMA-TDD

All-IPv4/v6

AdHoc/Mesh and Cellular Topologies

FPGA-SoC (Virtex 2)

2x2 OFDM(A) @ 2 GHz, 5MHz channels

Cellular (towards LTE)

Cellular, AdHoc and P2MP Topologies

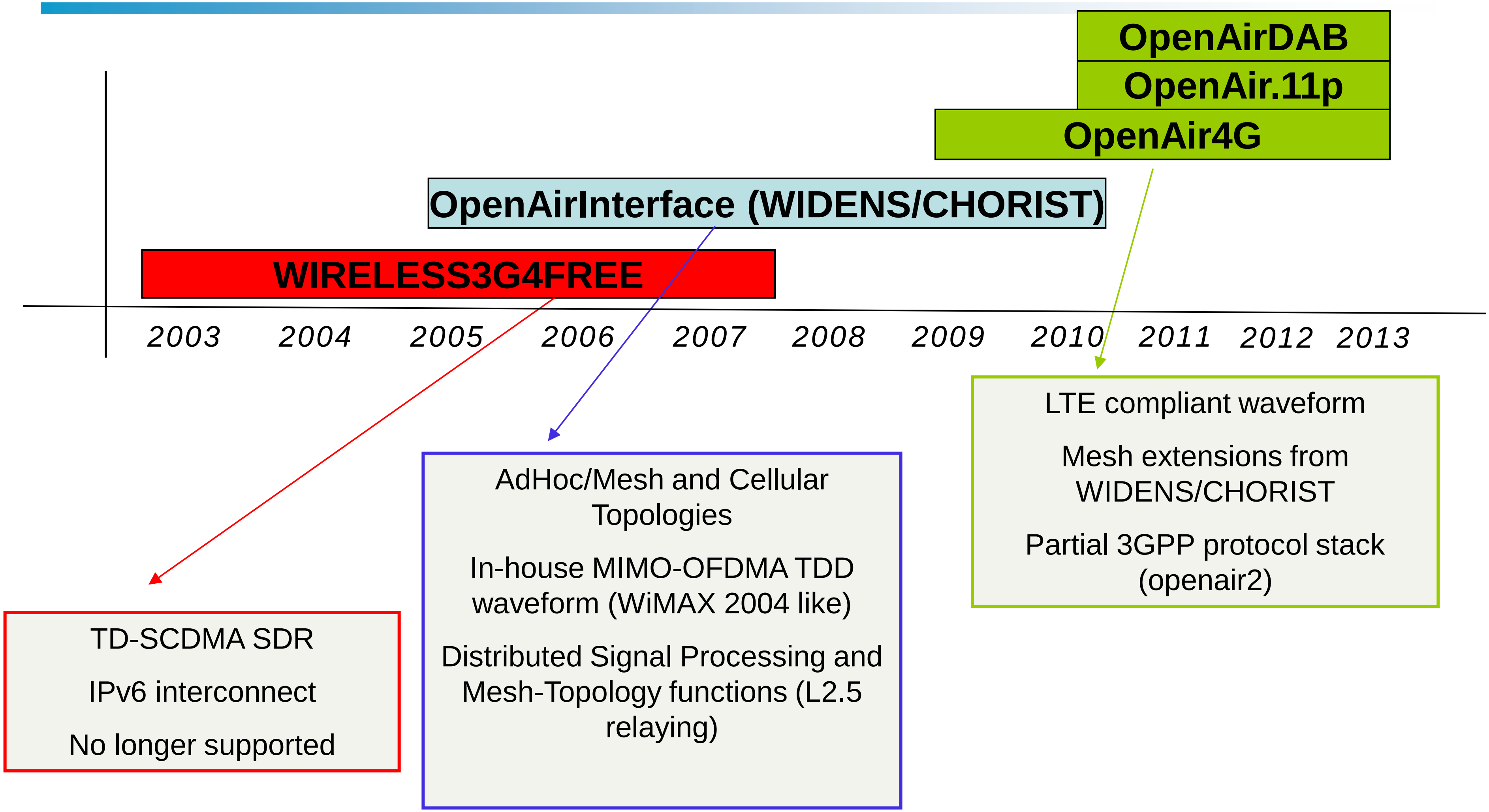
FPGA SoC (Virtex 5)+ Interface for partner Processing Engines

Agile Tuning module (0.2 – 7 GHz)

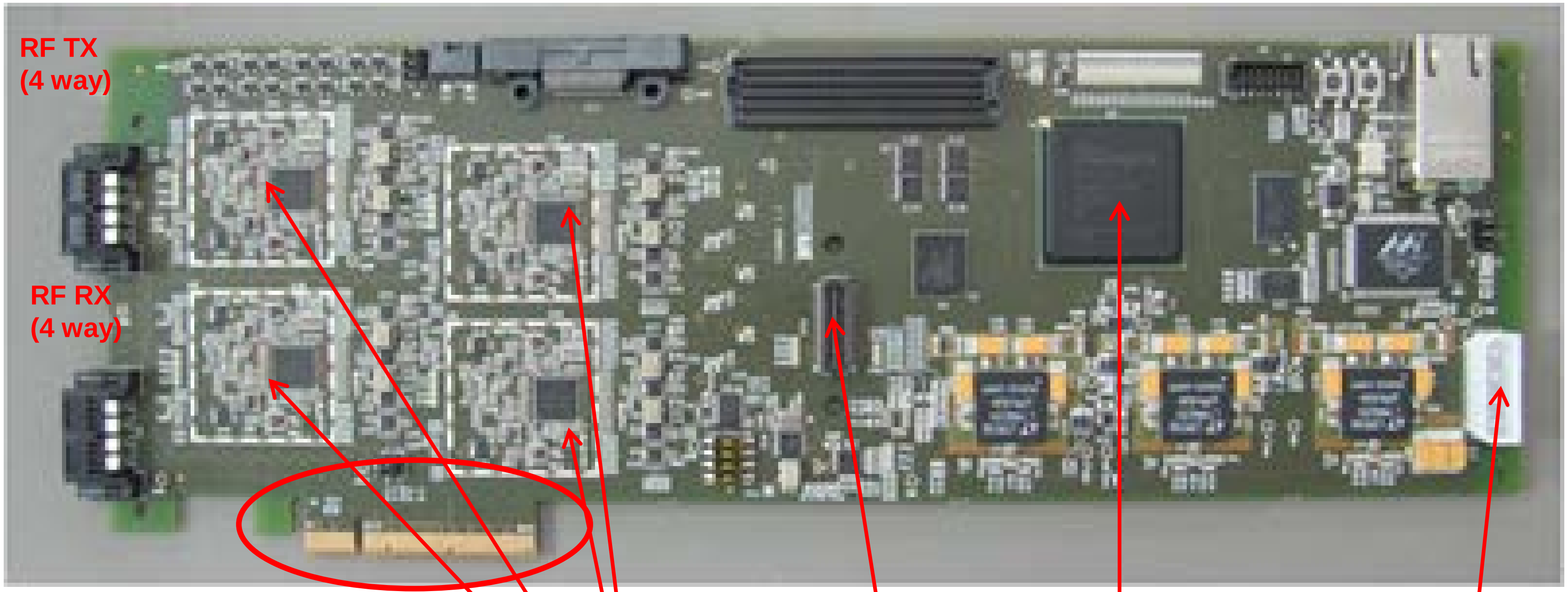
Maximum Channel BW 20 MHz

OFDM(A)/WCDMA

Software Roadmap



ExpressMIMO2



PCIexpress (1-way or 4-way)

Spartan 6 LX150T

GPIO for external RF control

12V from ATX power supply

4xLMS6002D RF ASICs
250 MHz – 3.8 GHz

ExpressMIMO2 key facts

- **Spartan 6 LX150T FPGA (PCIexpress like ExpressMIMO)**
  - Derived from Xilinx/Avnet evaluation board (but smaller, medium-sized PCIe format)
  - Used for FFT and Turbo/Viterbi decoders (key processing bottlenecks)
  - Control of RF and acquisition from converters
- **4 LIME Semiconductor zero-IF RF chipsets**
  - TX, RX and A/D, D/A on single-chip (1.5cm x 1.5cm)
  - 300 MHz – 3.8 GHz tuning bandwidth
  - FDD or TDD operation requires external RF
  - LTE UE, RN RF compliance (EVM)
  - 0 dBm output power, up to 30dBm with external RF

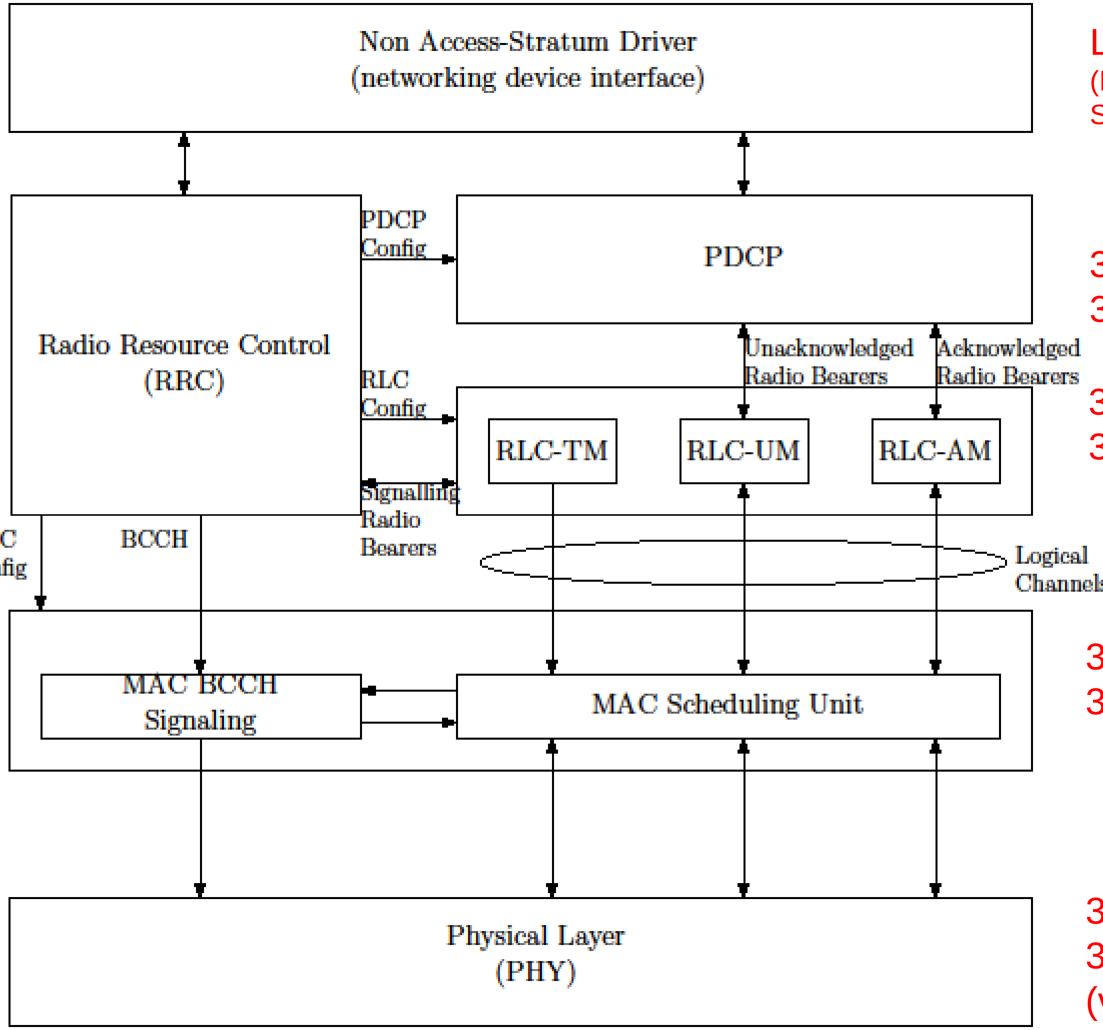
OPENAIR4G MODEM

Purpose

- **Develop an open-source baseband implementation of a subset of LTE Release-8/9 on top of OpenAirInterface.org SW architecture and HW demonstrators**
- **Goals**
  - Representative of LTE access-stratum
    - Full compliance of LTE frame (normal and extended prefix)
    - Full Downlink shared channel compliance
    - Support for a subset of transmission modes (2x2 operation)
      - Modes 1,2,4,5,6 (Mode 3 to be studied for inclusion)
    - Support for up to 3 sectors in eNB
  - Useful for measurement campaigns
  - Useful as starting point for research-oriented extensions (to justifiably claim potential impact on LTE-A)
  - Provide realistic (and rapid) LTE simulation environment for PHY/MAC

OpenAirLTE PHY/MAC Protocol Stack

3GPP Compliance
36-331 (v9.2)



Linux networking device
(IPv4/IPv6, classification/routing
Services for DRB)

3GPP Compliance
36-323 (v10.1)

3GPP Compliance
36-322 (v9.3)

3GPP Compliance
36-321 (v8.6)

3GPP Compliance
36-211, 36-212, 36-213
(v8.6)

Current Status (LTE/LTE-A)

- **PHY (36.211,36.212,36.213)**
  - LTE softmodem for 5 MHz (1.5, 10 + 20 too, but not completely functional yet)
    - Subset of 36-211,36-212 and 36-213 specifications
    - Mode 1, Mode 2, Mode 5 and Mode 6 support
    - Mode 4 under integration
  - Missing elements (the rest is largely supported)
    - User-selected and periodic feedback (not planned)
    - Modes 3,7 (not planned)
    - Rel-9/10 enhancements (Carrier Aggregation exists in branch)
- **MAC (36.321)**
  - Full random-access procedures
  - eNB scheduler for all transmission modes
  - UE Power headroom and BSR reporting
- **RLC (36.322)**
  - Complete UM/AM implementation, SRB interfaces with RRC for the moment

Current Status

- **PDCP (36.323)**
  - Currently just provides DRB interface for linux networking device
  - No security and compressions features
- **RRC (36.331)**
  - Two separate actions, RRC LITE and Cellular
  - LITE
    - is LTE only, with ASN.1 messages (asn1c C code generator) and subset of LTE RRC procedures (RRCConnectionRequest/Setup,ReconfigurationRequest)
    - Empty security context establishment will be added
    - Currently integrating measurement reporting and MobilityControlInfo (handover)
    - Extendable for Mesh networks (LOLA)
    - No SAE NAS support currently, but could be added …
  - Cellular
    - Inherits RRC from W3G4Free (IP/UMTS)
    - Automatic code generation using Esterel Studio
    - “hand”-compressed messages and research-oriented NAS extensions for IPv6 interconnect (QoS and mobility management)

OPENAIR4G LAB SESSION 1

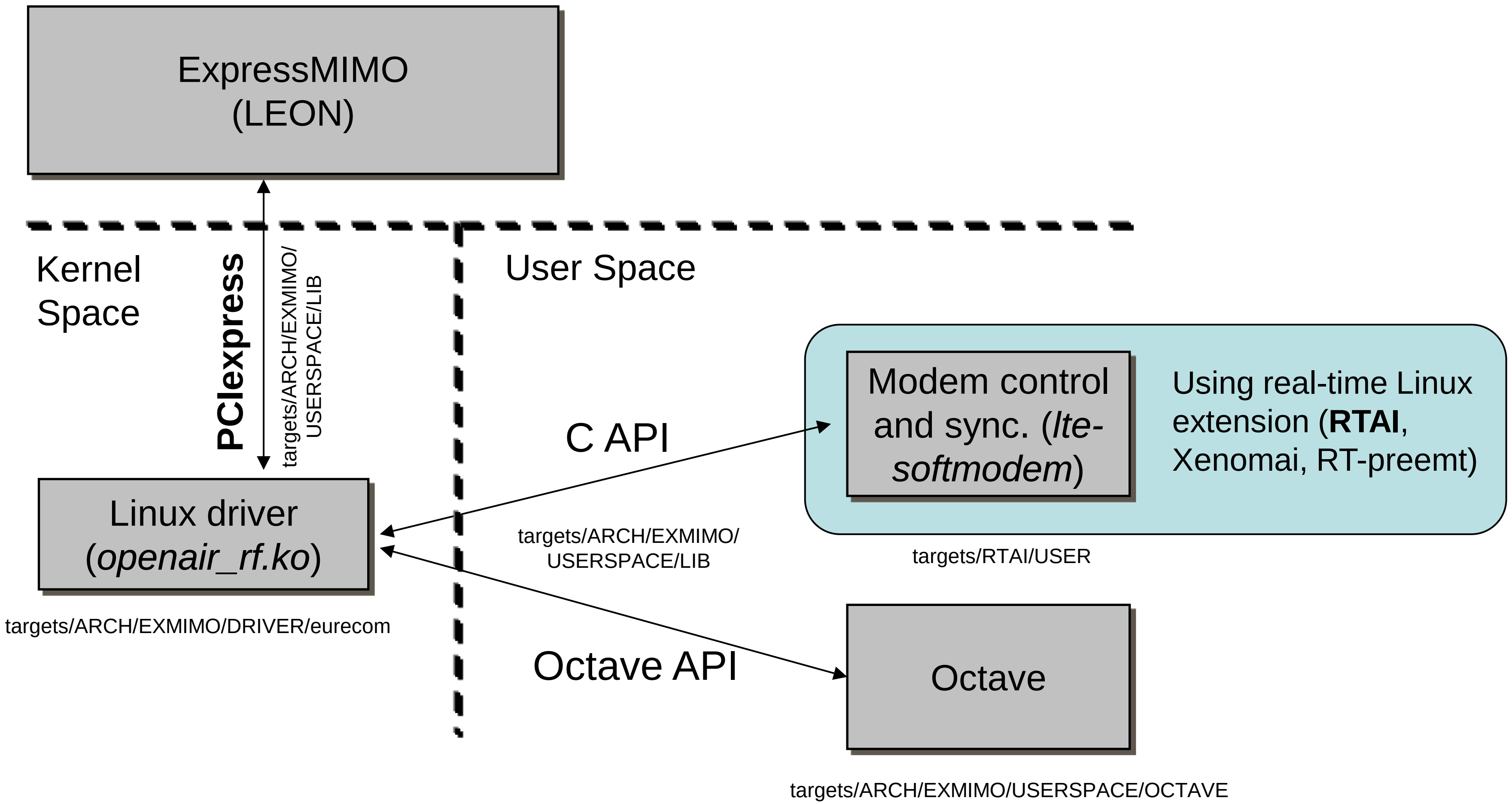
Objectives

- **Familiarization of OpenAir4G Development Environment through a simple example**
  - Insertion of kernel modules for CBMIMO1 hardware
  - Control of HW with OCTAVE (signal acquisition)
  - Basic DSP example
    - LTE Initial synchronization
  - Control of HW with user-space C programs (signal acquisition) using OpenAir4G x86-based DSP
  - Basic principles of Real-time operation under RTAI with CBMIMO1

Openair4G directories

- **Location**
  - http://svn.eurecom.fr/openair4G/trunk (read only)
  - http://svn.eurecom.fr/openairsvn/openair4G/trunk (read/write, requires account)
- **$OPENAIR_TARGETS**
  - Specific SW targets (SIMU,RTAI) for instantiating OpenAir4G components
- **$OPENAIR1_DIR**
  - Basic DSP routines for implementing subset of LTE specifications under x86 (36.211, 36.212, 36.213 3GPP specifications)
  - Channel simulation, sounding and PHY abstraction software,
- **$OPENAIR2_DIR (not for this lab session)**
  - MAC/RLC/PDCP/RRC
- **$OPENAIR3_DIR (not for this lab session)**
  - L3 IP-based Networking elements and applications

ExpressMIMO software architecture



Compiling and Loading the kernel modules

- **Start from** `$OPENAIR_TARGETS/RTAI/USER`
- **Compile kernel modules and firmware**
  - `make drivers`
- **This creates**
  - `$OPENAIR_TARGETS/ARCH/EXMIMO/DRIVER/eurecom/openair_rf.ko`
    - PCI/PCIe driver for ExpressMIMO
    - LINUX character device interfaces (open,close,ioctl,mmap)
  - `$OPENAIR_TARGETS/ARCH/EXMIMO/USERSPACE/OAI_FW_INIT/updatefw`
    - Tool to update firmware in ExpressMIMO
  - `$OPENAIR2_DIR/NAS/DRIVER/MESH/nasmesh.ko`
    - NAS driver providing Linux networking interface

Compiling and Loading the kernel modules

- **Identifying the HW**
  - To see that the HW is identified by Linux you can do `lspci` and you should see a device with the name “Xilinx Corporation …”
- **Loading drivers and firmware**
  - `sh init_exmimo2.sh`
  - Check that module is loaded using `lsmod`
  - Check that firmware is initialized using `dmesg`

```
[   78.055319] [LEON card0]: FWINIT: Will start execution @ 40000000, stack @ 43fffff0
[   78.744943] [LEON card0]: pcie_initialize_interface_bot(): firmware_block_ptr
15f00100, printk_buffer_ptr 15f40100, pci_interface_ptr 15f40500, exmimo_id_ptr
15f40700
[   78.745318] [LEON card0]: System Info:
[   78.746585] [LEON card0]: Bitstream: SVN Revision: 4855, Build date (GMT): Fri 2013-
03-15 10:31:49,  User ID: 0x0001
[   78.747824] [LEON card0]: Software:  SVN Revision: 4863, Build date (GMT): Thu 2013-
04-18 08:44:32
[   78.749034] [LEON card0]: ExpressMIMO-2 SDR! (Built on Apr 18 2013 10:44:33)
[   78.750261] [LEON card0]: Initialized LIME.
[   78.751496] [LEON card0]: Initializing RF Front end chain0 (to B19G_TDD).
[   78.751759] [LEON card0]: ready.
```

User-space applications

- **API to dialogue with driver**
  - `targets/ARCH/EXMIMO/USERSPACE/LIB/openair0_lib.h`
    - `int openair0_open(void);`
      - ☞ Initializes PCI interface `openair0_exmimo_pci` (see `targets/ARCH/EXMIMO/DEFS/pcie_interface.h`)
    - `int openair0_close(void);`
    - `int openair0_dump_config(int card);`
      - ☞ Dumps `openair0_exmimo_pci` to the card
    - `int openair0_get_frame(int card);`
    - `int openair0_start_rt_acquisition(int card);`
    - `int openair0_stop(int card);`

User-space applications

- **OCTAVE wrapper API**
  - `targets/ARCH/EXMIMO/USERSPACE/OCTAVE`
  - Gives access to API from OCTAVE
    - `oarf_config_exmimo` (see online help for parameters)
    - `sig = oarf_get_frame(card)`
    - `oarf_send_frame(card,sig,nbits)`
    - `oarf_stop(card)`
  - To compile the .cc to .oct files (note: octave-headers needs to be installed), do
    - `make clean`
    - `make oarf`
    - `make gpib` (if you want gpib)
  - Examine `rx_spec.m` as an example

OCTAVE example (rx_spec.m)

```
card=0;
limeparms;
rf_mode = (RXEN+TXLPFNORM+TXLPFEN+TXLPF25+RXLPFNORM+RXLPFEN...
           +RXLPF25+LNA1ON+LNAMax+RFBBNORM+DMAMODE_RX)*[1 1 0 0];
freq_rx = 1907600000*[1 1 1 1];
freq_tx = freq_rx;
tx_gain = 0*[1 1 1 1];
rx_gain = 30*[1 1 1 1];
rf_local= rf_local*[1 1 1 1];
rf_rxdc = rf_rxdc*[1 1 1 1];
rf_vcocal=rf_vcocal_19G*[1 1 1 1];
eNBflag = 0;
tdd_config = DUPLEXMODE_FDD + TXRXSWITCH_TESTRX;
syncmode = SYNCMODE_FREE;
rffe_rxg_low = 61*[1 1 1 1];
rffe_rxg_final = 61*[1 1 1 1];
rffe_band = B19G_TDD*[1 1 1 1];
```

Set parameters

```
oarf_config_exmimo(card,freq_rx,freq_tx,tdd_config,syncmode,...
           rx_gain,tx_gain,eNBflag,rf_mode,rf_rxdc,rf_local,...
           rf_vcocal,rffe_rxg_low,rffe_rxg_final,rffe_band)
```

Send parameters to card

```
s=oarf_get_frame(card);
```

Get 10ms of signal from RX chains

```
f = (7.68*(0:length(s(:,1))-1)/(length(s(:,1))))-3.84;
spec0 = 20*log10(abs(fftshift(fft(s(:,1)))));
spec1 = 20*log10(abs(fftshift(fft(s(:,2)))));
```

Compute spectrum

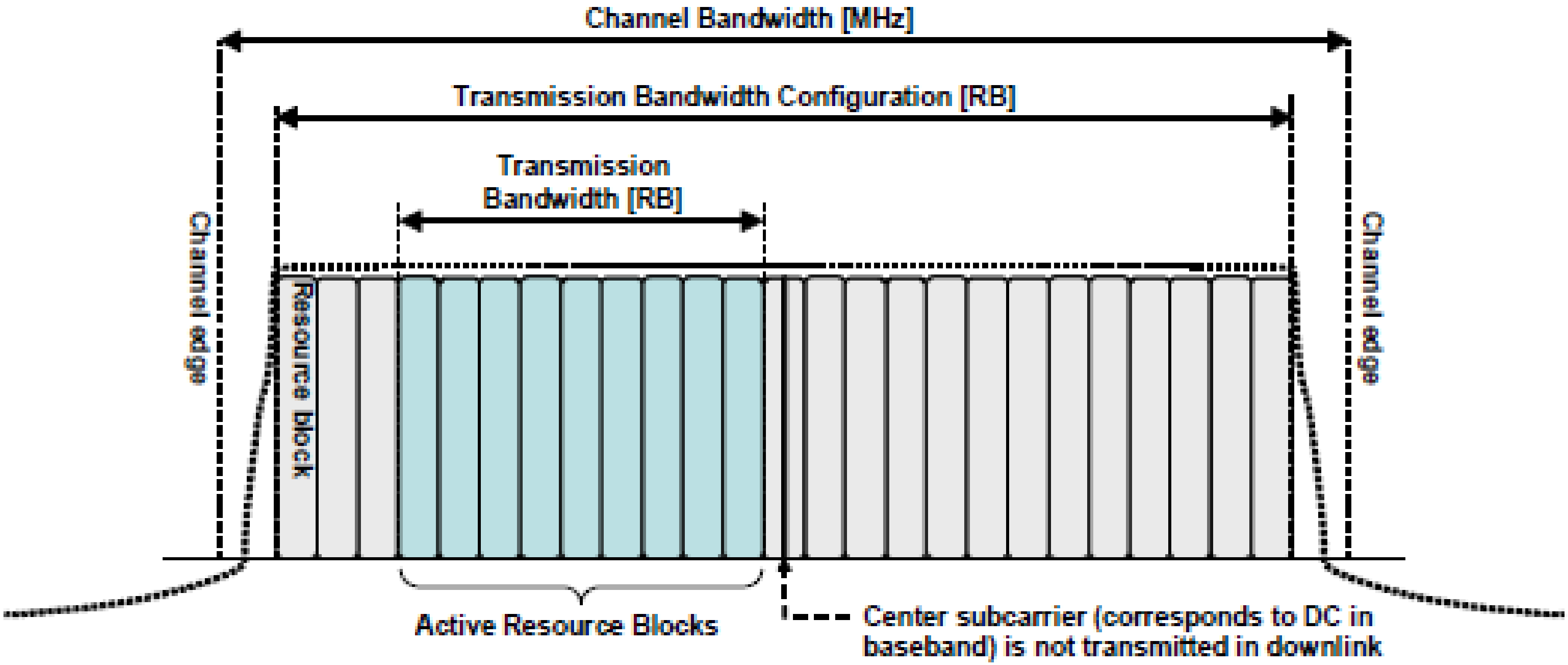
```
clf
plot(f',spec0,'r',f',spec1,'b')
axis([-3.84,3.84,40,160]);
legend('Antenna Port 0','Antenna Port 1');
grid
```

Plot

LTE Initial Synch Example

- **Need a few basics in LTE DL Transmission**
  - OFDM + QAM
  - Frame formats
  - Synchronization signals
    - Primary Synchronization Signal (PSS)
    - Secondary Synchronization Signal (SSS)
    - Physical Broadcast Channel (PBCH)
    - Cell-specific Reference Signals (CSRS)

Resource blocks



- LTE defines the notion of a <u>resource block</u> which represents the minimal scheduling resource for both uplink and downlink transmissions
- A <u>physical resource block</u>(PRB) corresponds to 180 kHz of spectrum

Common PRB Formats

Channel Bandwidth (MHz)	N_{RB}^{DL}/N_{RB}^{UL}	Typical IDFT size	Number of Non-Zero Sub-carriers (REs)
1.25	6	128	72
5	25	512	300
10	50	1024	600
15	75	1024 or 2048	900
20	100	2048	1200

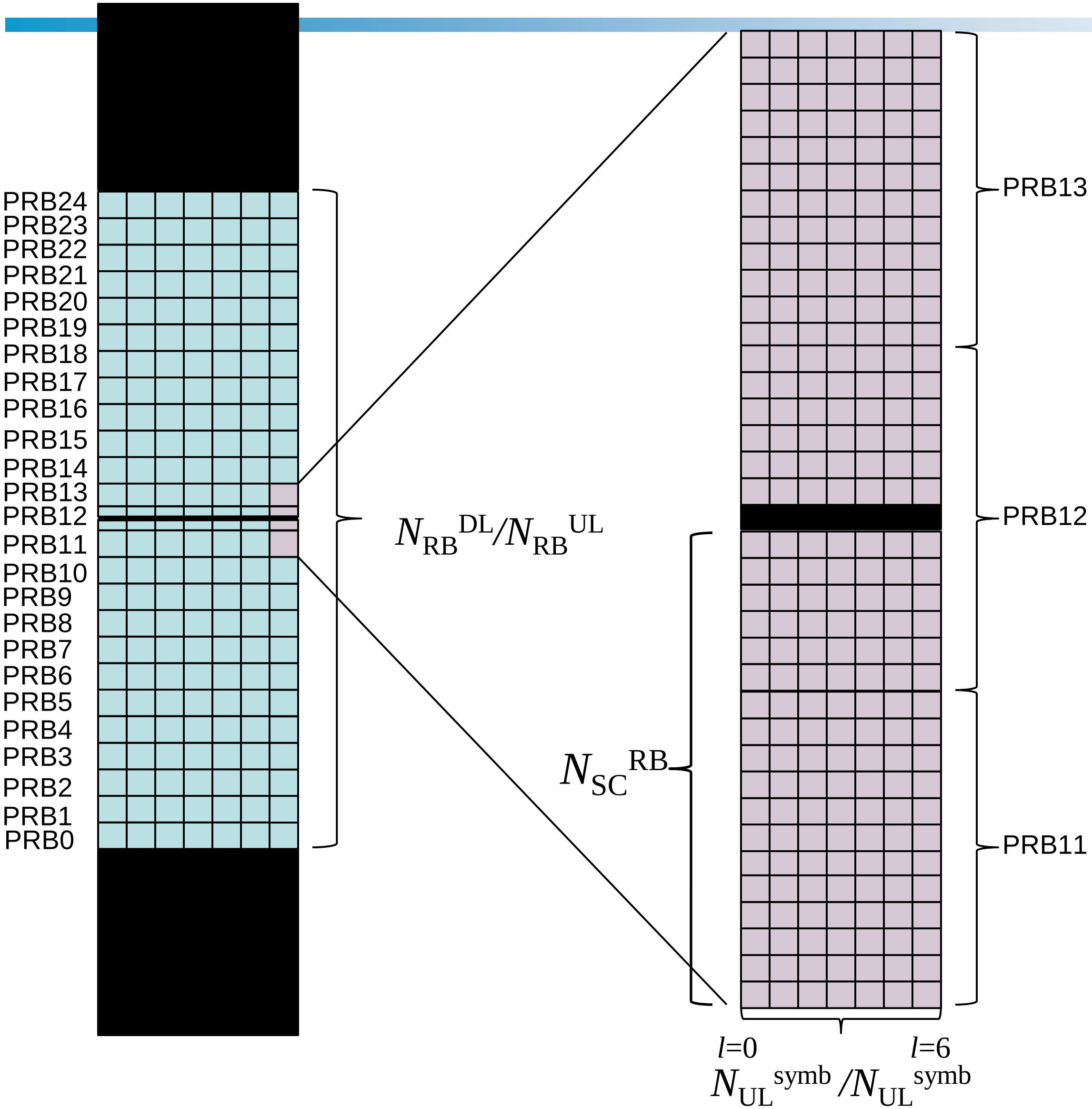
- **PRBs are mapped onto contiguous OFDMA/SC-FDMA symbols in the time-domain (6 or 7)**
- **Each PRB is chosen to be equivalent to 12 (15 kHz spacing) sub-carriers of an OFDMA symbol in the frequency-domain**
  - A 7.5kHz spacing version exists with 24 carriers per sub (insufficiently specified)
- **Because of a common PRB size over different channel bandwidths, the system scales naturally over different bandwidths**
  - UEs determines cell bandwidth during initial acquisition and can be any of above

OFDMA/SC-FDMA Mapping

- **OFDMA/SC-FDMA Sub-carriers are termed “Resource Elements” (RE)**
- **DC carrier (DL) and high-frequencies are nulled**
  - Spectral shaping and DC rejection for Zero-IF receivers
  - Half the bandwidth loss w.r.t. WCDMA (22%)

Channel Bandwidth (MHz)	N_{RB}^{DL}/N_{RB}^{UL}	Bandwidth Expansion
1.25	6	8%
5	25	11%
10	50	11%
15	75	11%
20	100	11%

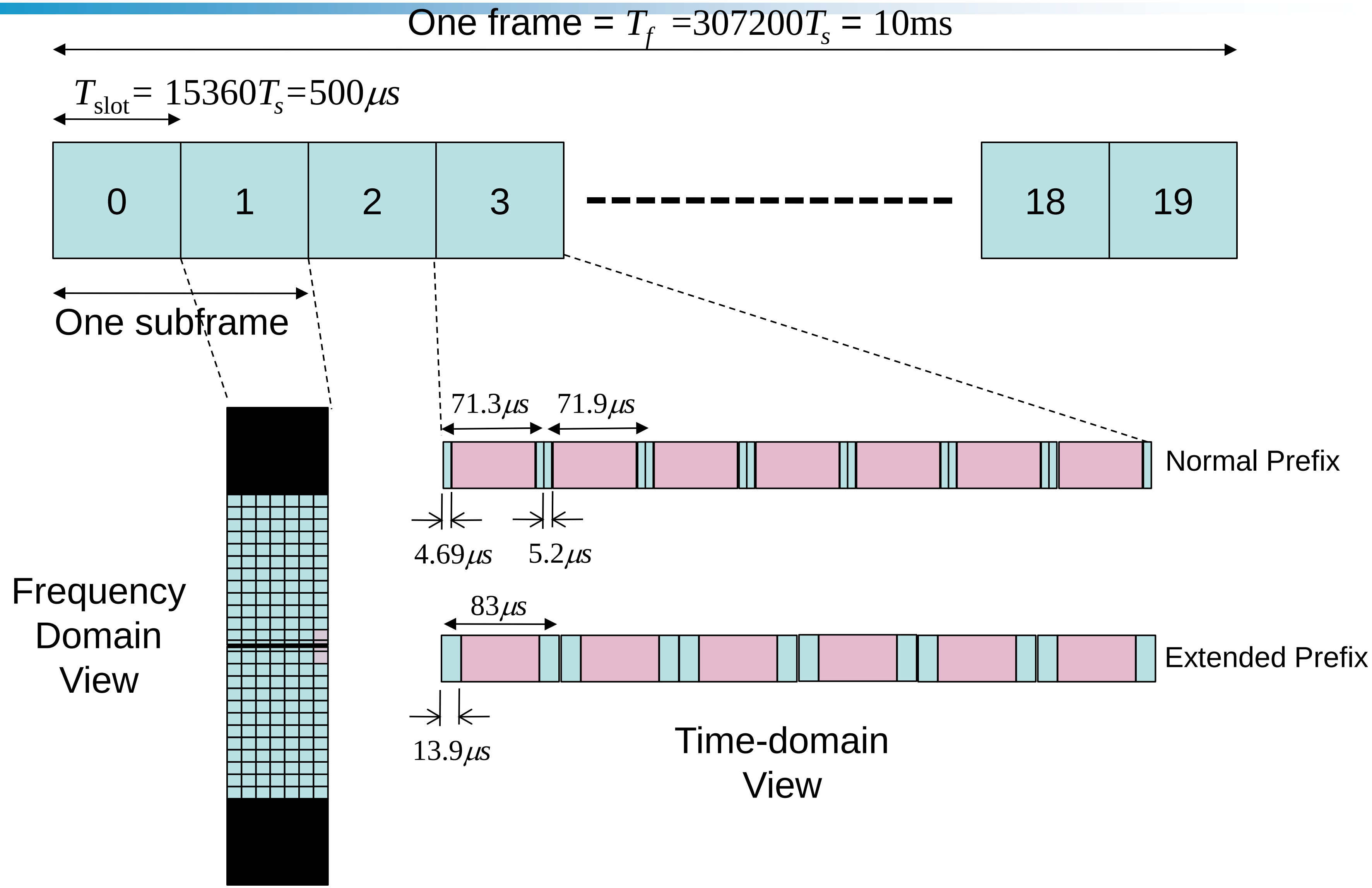
Example: 300 REs, 25 RBs (5 MHz channel)



"Normal" Cyclic Prefix Mode (7 symbols)

"Extended" Cyclic Prefix Mode (6 symbols)

Sub-frame and Frame

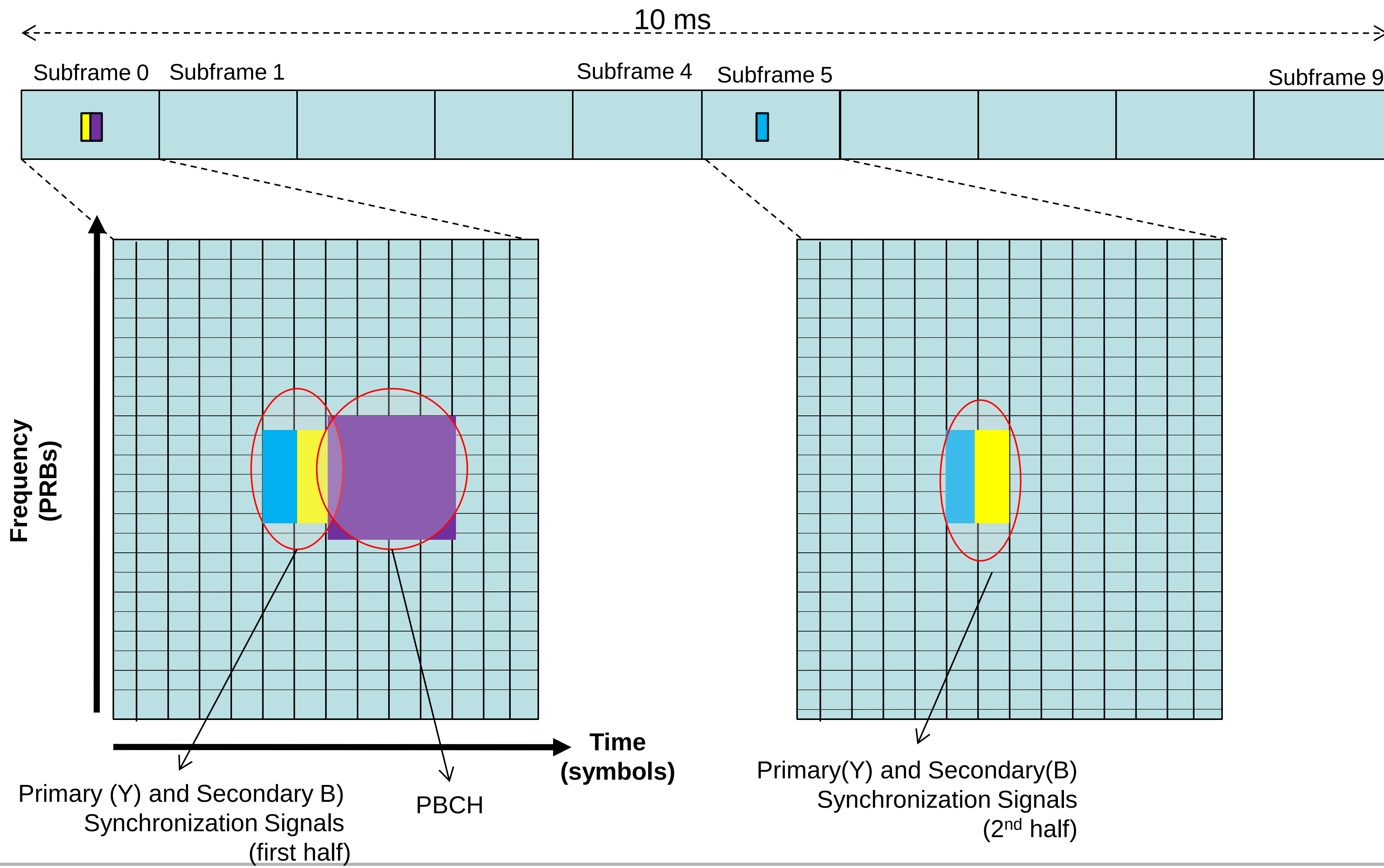


EURECOM

LTE UE Synchronization Procedures

- **Cell Search comprises**
  1. Timing and frequency synchronization with the cell using the primary synchronization reference signal. This also gives the Cell ID group $N_{ID}^{(2)}$ $(0,1,2)$
  2. Cell ID $N_{ID}^{(1)}$ (0,…,166) and Frame type (FDD/TDD, Normal/Extended Prefix) determination from secondary synchronization reference signals
  3. Demodulation of PBCH (using $N_{ID}^{Cell} = 3N_{ID}^{(1)} + N_{ID}^{(2)}$) to receive basic system information during steady-state reception
     1. N_{RB}^{DL} (cell bandwidth)
     2. PHICH-config (to allow PDCCH demodulation, for system information)
     3. Frame number (8 bits from payload, 2 bits from redundancy version)
     4. Antenna configuration (1,2,4 from CRC mask)

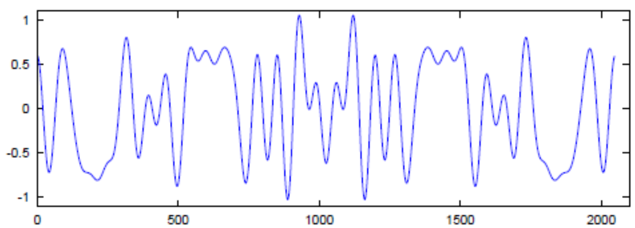
Initial Timing/Frequency Acquisition (Synchronization Signals, FDD Normal CP)



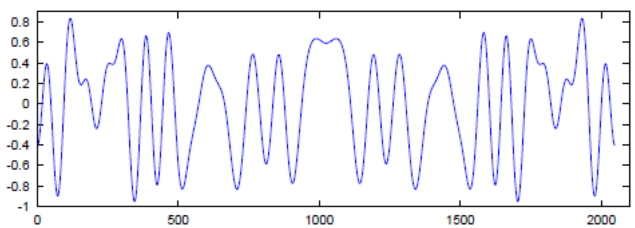
Primary Synchronization Signal (PSS)

- **Zadoff-Chu root-of-unity sequence has excellent auto-correlation properties and is very tolerant to frequency-offsets**

$$d_u(n) = \begin{cases} e^{-j\frac{\pi u n(n+1)}{63}} & n = 0,1,\ldots,30 \\ e^{-j\frac{\pi u(n+1)(n+2)}{63}} & n = 31,32,\ldots,61 \end{cases}$$

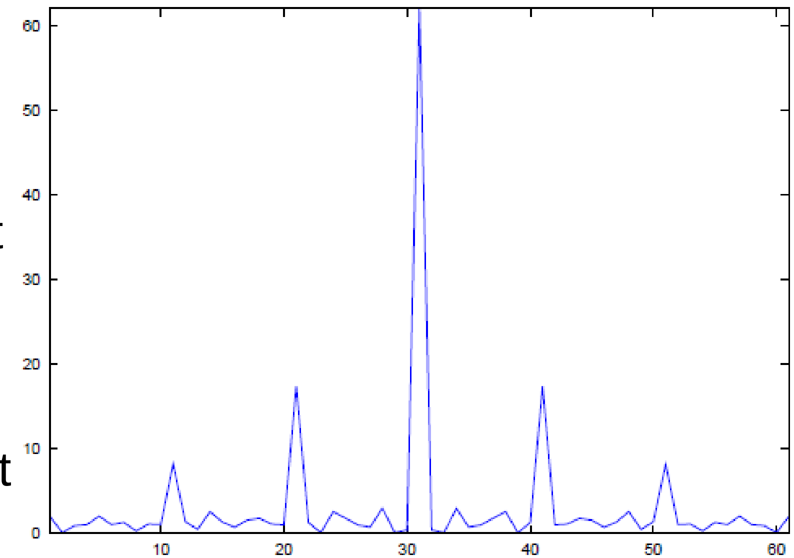


Real component (time-domain u=25)



Imag component (time-domain, u=25)

Autocorrelation sequence



EURECOM

Primary Synchronization RX

- **Correlation of 3 primary sequences ($d^*_u(-n)$) with received signal. Each eNB (or sector) has different sequence => Reuse pattern of 3 for different eNB or sectors ($N_{\mathrm{ID}}^{(2)}$)**
- **Primary Purpose: Determine start of frame**
- **Alternate purposes: Channel estimation for SSS/PBCH and frequency offset estimation**

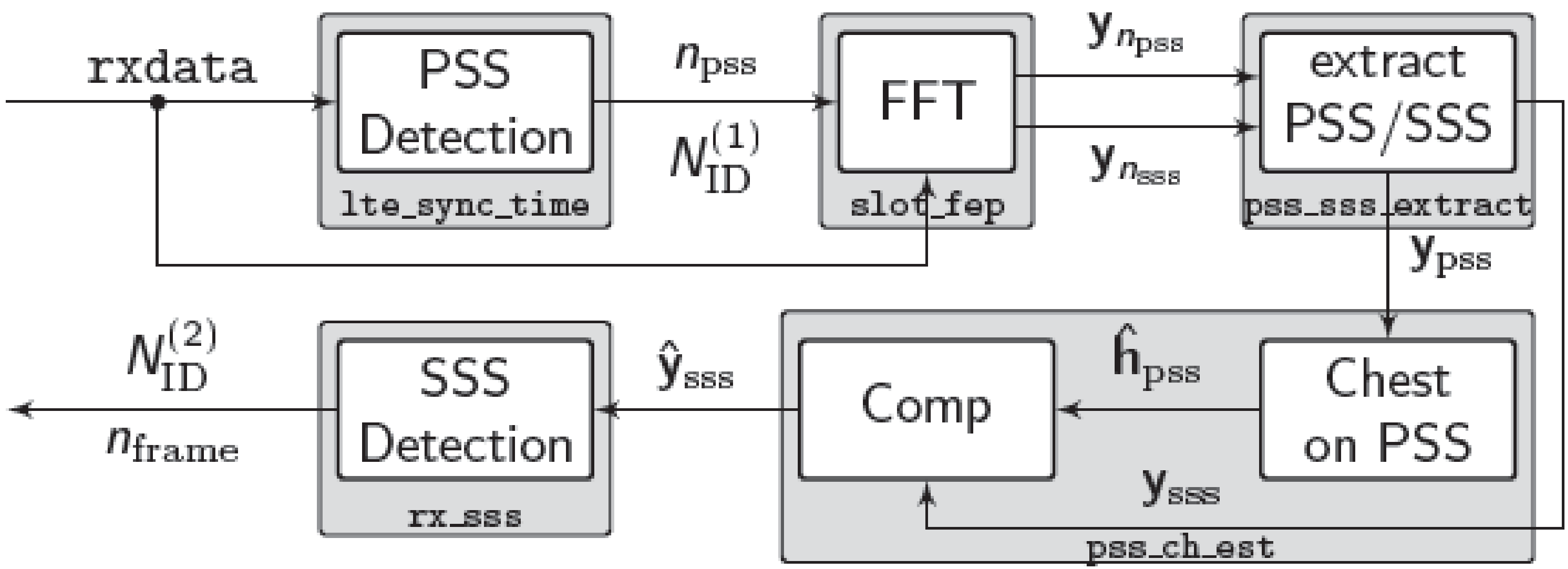
Secondary Synchronization Signal (SSS)

- <u>Purpose</u>: determine frame type and cell ID $N_{ID}^{(1)}$
- Implemented as BPSK-modulated interleaved sequence of two length-31 binary *m*-sequences (*m*=31) with cyclic shifts m_0 and m_1. and scrambled by the two different scrambling sequences
  - Results in 167 possible BPSK sequences for each of subframe 0 and 5
- The receiver must perform correlations with all 167 sequences and find the most likely transmitted sequence. It can use the output of the primary sequence correlation as a rough channel estimate to improve detection probability
- Position relative to PSS allows for frame type determination

Secondary Synchronization RX

- **Hypothesis: one of 4 frame types TDD/FDD, normal/extended prefix => gives position in samples of SSS with respect to PSS detected in primary synchronization**
- **Use channel estimate (partially coherent) from PSS and quantized uniform phase offset to compensate residual frequency offset (PSS/SSS not in same symbol) and amplitudes in SSS symbol**
- **Correlate with 167 out of 167 * 3 sequences (167 per PSS $N_{\mathrm{ID}}^{(2)}$) of length 62 in each of slots 0 and 10**
- **Choose sequence which has highest coherent correlation**
- **This has to be done with 2 different assumptions (subframe 0 or subframe 5 is first in RX buffer), or we just wait until we receive an RX frame in the correct order (i.e. when subframe 0 falls in the first 5 ms of the RX buffer)**

Summary


rxdata
PSS Detection
lte_sync_time
n_{pss}
$N_{\mathrm{ID}}^{(1)}$
FFT
slot_fep
$\mathbf{y}_{n_{\mathrm{pss}}}$
$\mathbf{y}_{n_{\mathrm{sss}}}$
extract PSS/SSS
pss_sss_extract
$\mathbf{y}_{\mathrm{pss}}$
$N_{\mathrm{ID}}^{(2)}$
n_{frame}
SSS Detection
rx_sss
$\hat{\mathbf{y}}_{\mathrm{sss}}$
Comp
$\hat{\mathbf{h}}_{\mathrm{pss}}$
Chest on PSS
$\mathbf{y}_{\mathrm{sss}}$
pss_ch_est

Building the PSS/SSS Part First

- **OCTAVE files for PSS generation can be found here**
  - `openair1/PHY/LTE_REFSIG/primary_synch.m`
    - `primary_synch0` PSS in frequency domain
    - `primary_synch0_time` PSS in time domain (1.92MHz)
- **OCTAVE files for SSS generation can be found here**
  - `openair1/PHY/LTE_TRANSPORT/sss_gen.m`
    - `d0, d5` all 168*3 SSS in frequency domain
- **Start an editor and create a file based on** `rx_spec.m`, **in the same directory so you have the OpenAir4G .**oct **files**

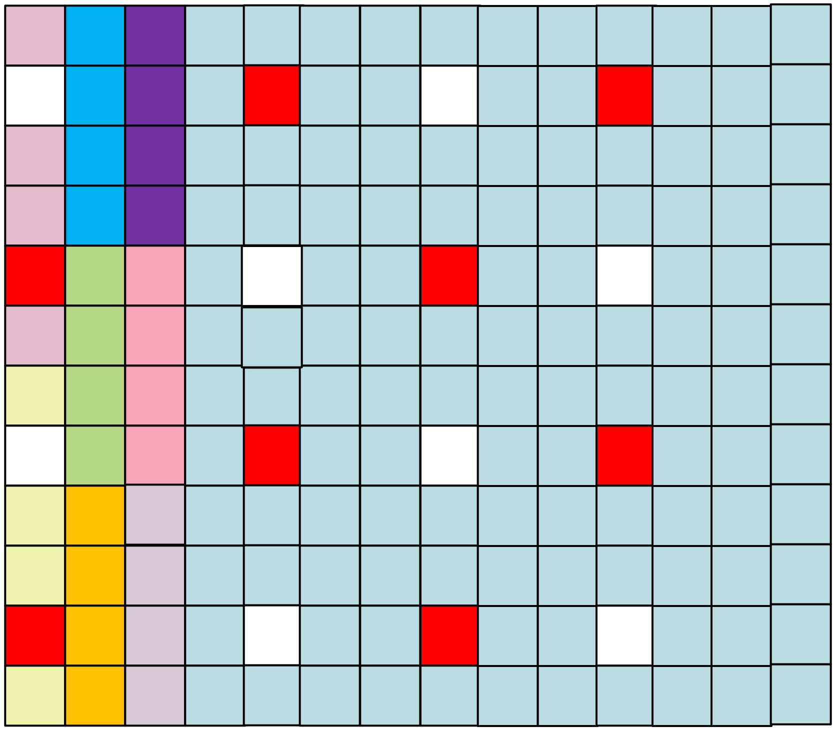
Steps

- **Upsample time-domain PSS to 7.68MHz (factor 4)**
- **Correlate received signal with this sequence and take magnitude (use** `conv` **and** `abs`**)**
- **Search for peaks (both) separated by 38400 samples (5 ms @ 7.68 Ms/s)**
- **Do above SSS procedure according to 4 potential SSS positions**
  - FDD/Normal CP : SSS is (512+36) 548 samples before PSS
  - FDD/Extended CP: SSS is (512 + 128) 640 samples before PSS
  - TDD/Normal CP: SSS is (512+36+512+40+512+40) 1652 samples before PSS
  - TDD/Extended CP: SSS is (512+128+512+128+512+128) 1920 samples before PSS

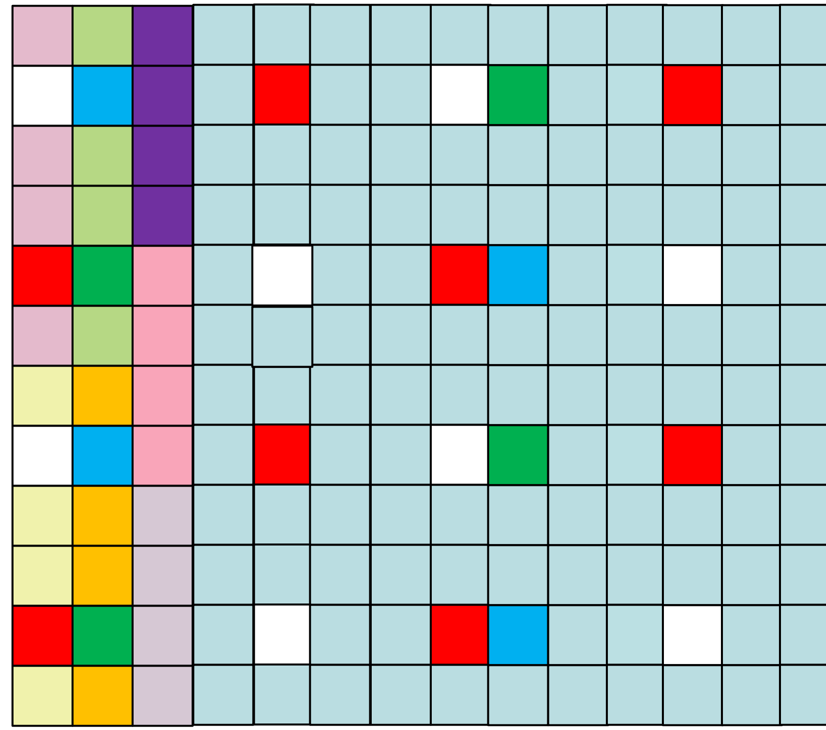
PBCH Detection

- **Detection of the PBCH requires the following steps**
  1. Generation of the cell-specific reference signals based on the cell ID derived from SSS detection
  2. Performing channel estimation for the 4 symbols of the PBCH
  3. Extracting the PBCH reference elements
  4. Applying the conjugated channel estimates to the received reference elements
  5. Channel decoding

Cell-Specific Reference Signals



$p=\{0\},p=\{0,1\}$



$p=\{0,1,2,3\}$

$p=0$

$p=1$ (if active)

$p=0$

$p=1$

$p=0$

$p=1$ (if active)

Cell-Specific Reference Signals

- **Pseudo-random QPSK OFDM symbols**
  - Based on generic LTE Gold sequence
  - Different sequence for different cell IDs
  - Different in each symbol of sub-frame
  - Different in each sub-frame, but periodic across frames (10ms)
- **Evenly spaced in subframe to allow for simple and efficient least-squares interpolation-based receivers**
  - Between REs in frequency-domain
  - Across symbols in time-domain

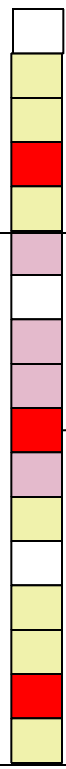
Channel Estimation in LTE (simple)

– Recall that receiver sees

$$\mathbf{R}_k^N = \mathrm{DFT}(\mathbf{r}_k^N) = \mathbf{H}^N \mathbf{S}_k^N + \mathbf{Z}_k^N$$

– Must get channel estimate for channel compensation

$$\hat{\mathbf{H}}^N = \mathbf{H}^N + \mathbf{H}_e^N$$ Estimation error



$R(16) = P(16)H_1(16) + Z(16)$ → $\hat{H}_1(16) = P^*(16)R(16)$

PRB1

$R(13) = P(13)H_0(13) + Z(13)$ → $\hat{H}_0(13) = P^*(13)R(13)$

$R(10) = P(10)H_1(10) + Z(10)$ → $\hat{H}_1(10) = P^*(10)R(10)$

$R(7) = P(7)H_0(7) + Z(7)$ → $\hat{H}_0(7) = P^*(7)R(7)$

PRB0

$\hat{H}_1(5) = (5/6)\hat{H}_1(4) + (1/6)\hat{H}_1(10)$ Interpolation

$R(4) = P(4)H_1(4) + Z(4)$ → $\hat{H}_1(4) = P^*(4)R(4)$

$\hat{H}_1(3) = (7/6)\hat{H}_1(4) - (1/6)\hat{H}_1(10)$ Extrapolation

$\hat{H}_0(2) = (5/6)\hat{H}_0(1) + (1/6)\hat{H}_0(7)$ Interpolation

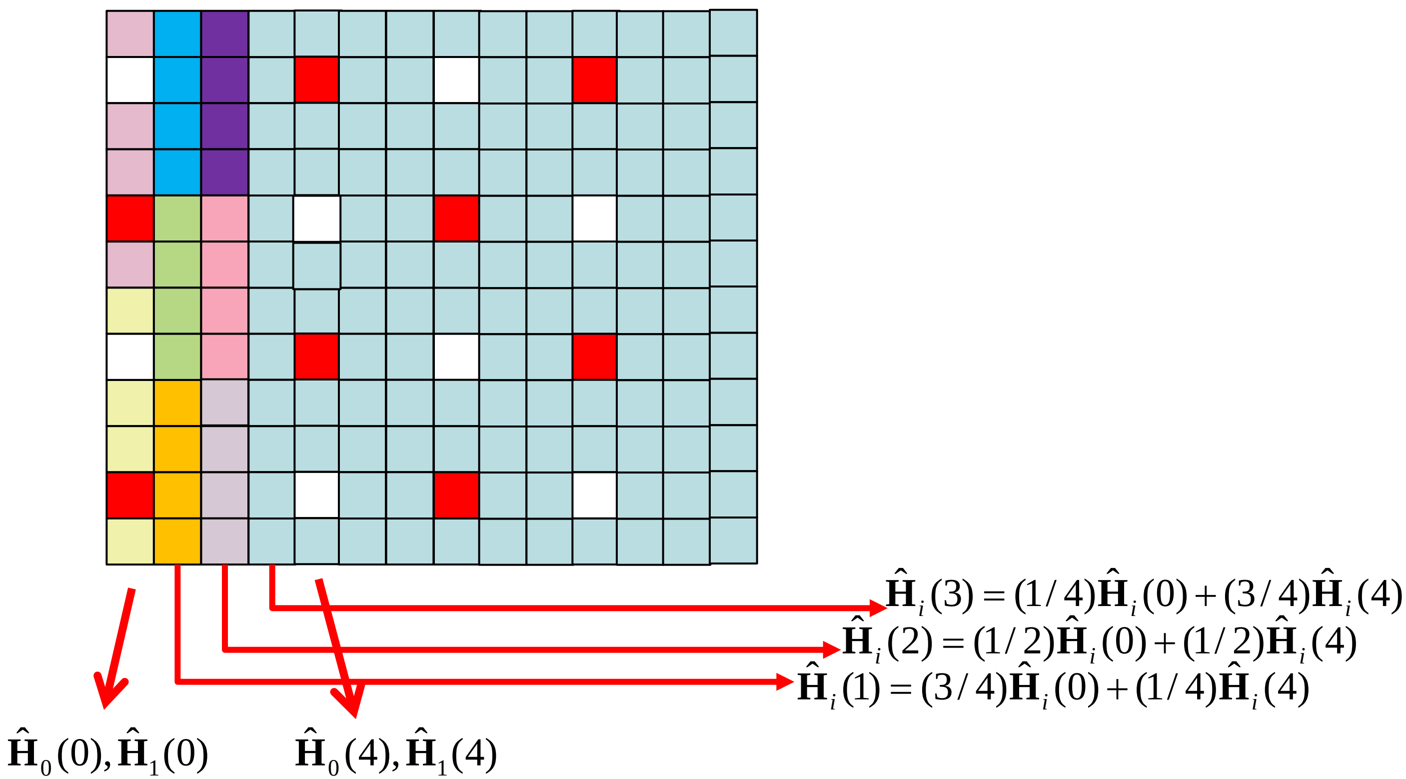
$R(1) = P(1)H_0(1) + Z(1)$ → $\hat{H}_0(1) = P^*(1)R(1)$

$\hat{H}_0(0) = (7/6)\hat{H}_0(1) - (1/6)\hat{H}_0(7)$ Extrapolation

NO pilots here

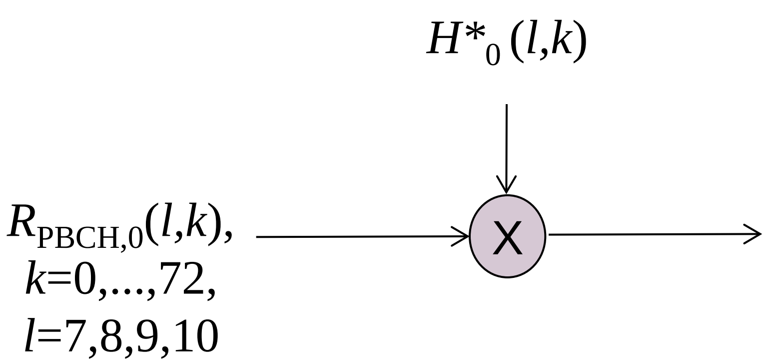
Channel Estimation in LTE (simple) – cont'd

- **The previous steps allow for determining the frequency response (MIMO) on symbols where the pilots are located**
- **For the remaining symbols, we perform time-interleaving across adjacent symbols with pilots**



Performing the Channel Estimation and Channel Compensation

- **Use the supplied `txsigF0.m` file, which contains the transmit signal for the PBCH (normal prefix)**
  - This is usually recomputed in the receiver (we will examine the C version later)
- **Extract reference symbols (symbols 7 and 11) and perform the time/frequency interpolation**
- **Apply (channel compensation) the channel estimate to the received resource elements and plot the constellation of the output**



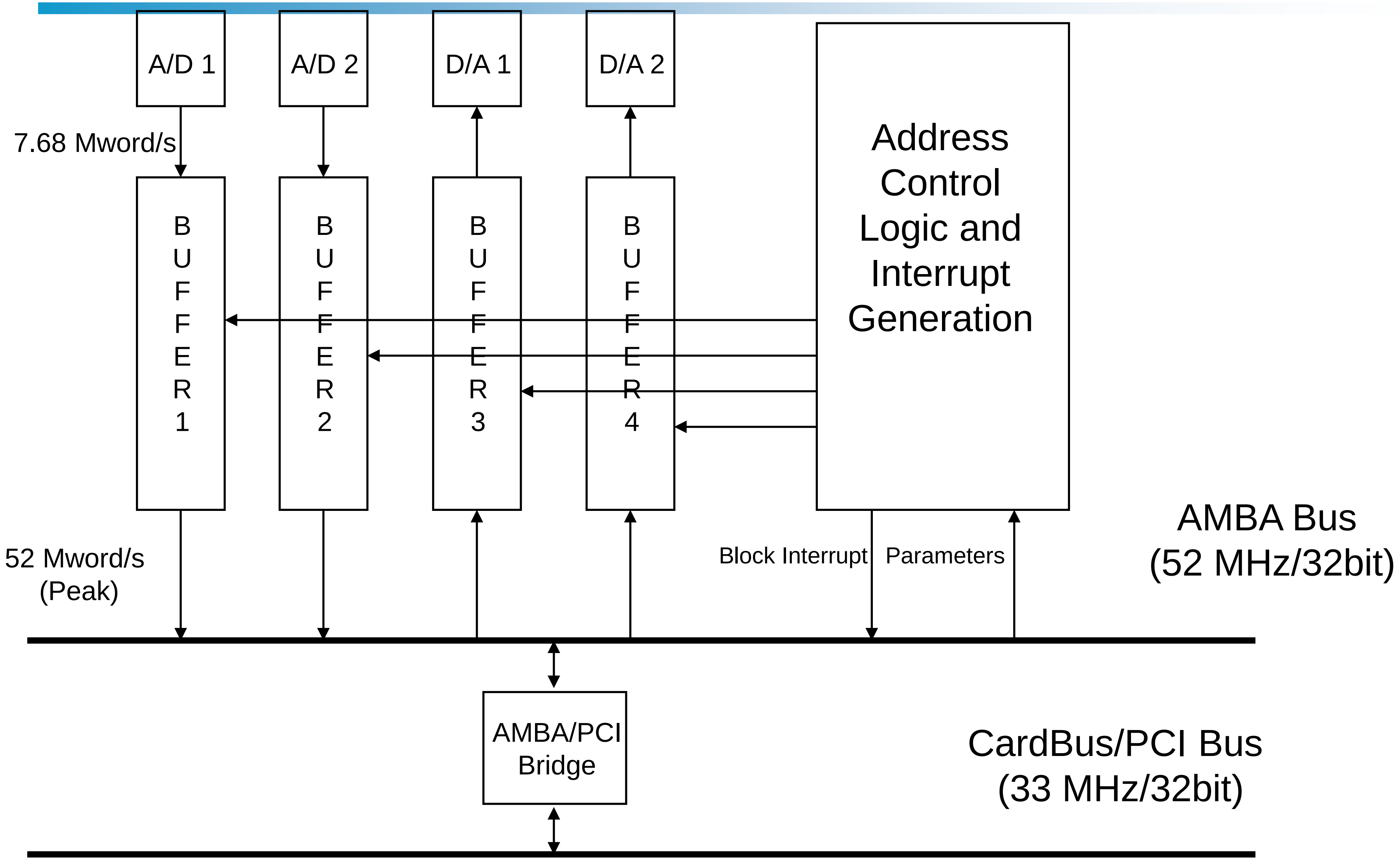
The rest …

- **The rest we cannot do in OCTAVE (unless we implement all the channel decoding functions), but need to go to the OpenAir4G C implementation for**
  - Deinterleaving
  - Channel decoding
  - Descrambling
- **To see how this is done check out the following files**
  - `openair1/PHY/LTE_TRANSPORT/initial_sync.c`
  - `openair1/PHY/LTE_TRANSPORT/sss.c`
  - `openair1/PHY/LTE_TRANSPORT/pbch.c`

Towards real-time operation

- **Real-time operation requires real-time Linux extension**
  - Assures that threads are served with minimum latency and highest priority
- **Examples**
  - RTAI (used currently)
  - Xenomai
  - RT-preement

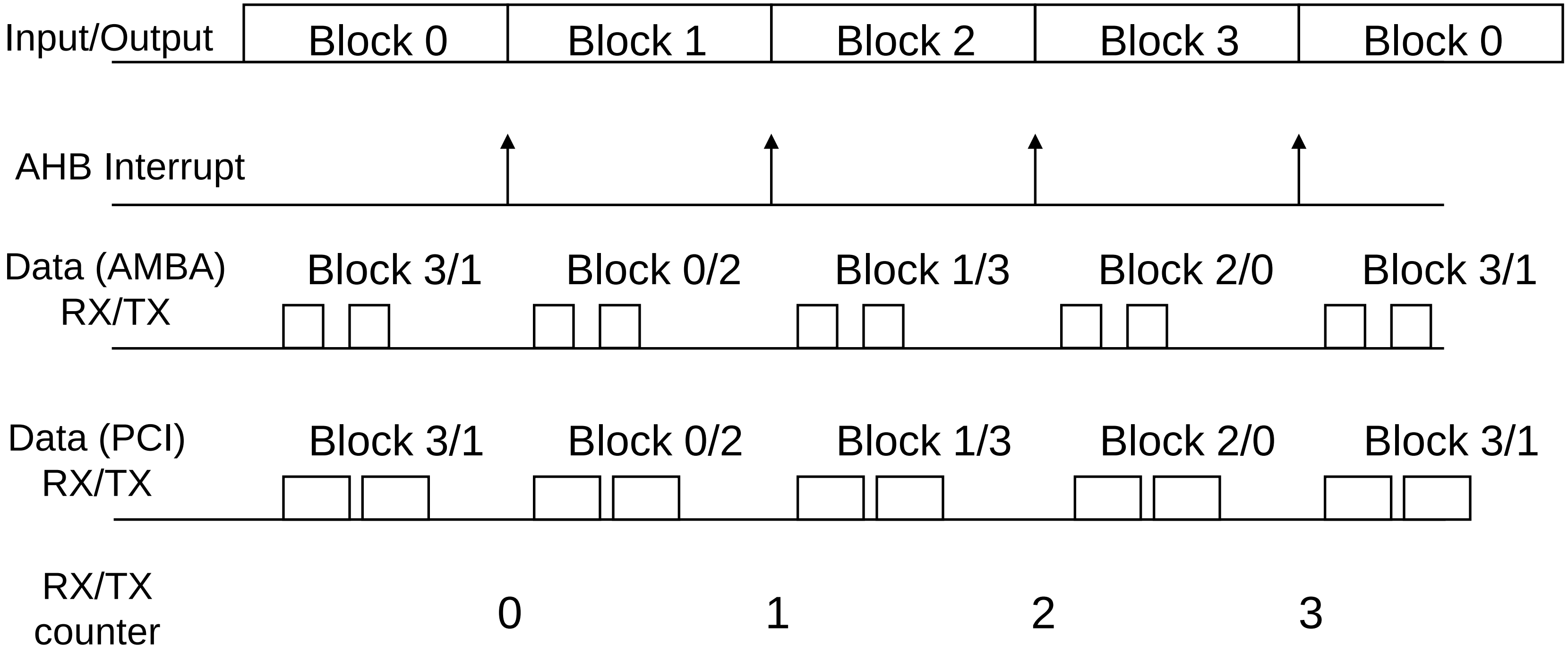
Acquisition (Card side)



Acquisition (AMBA)

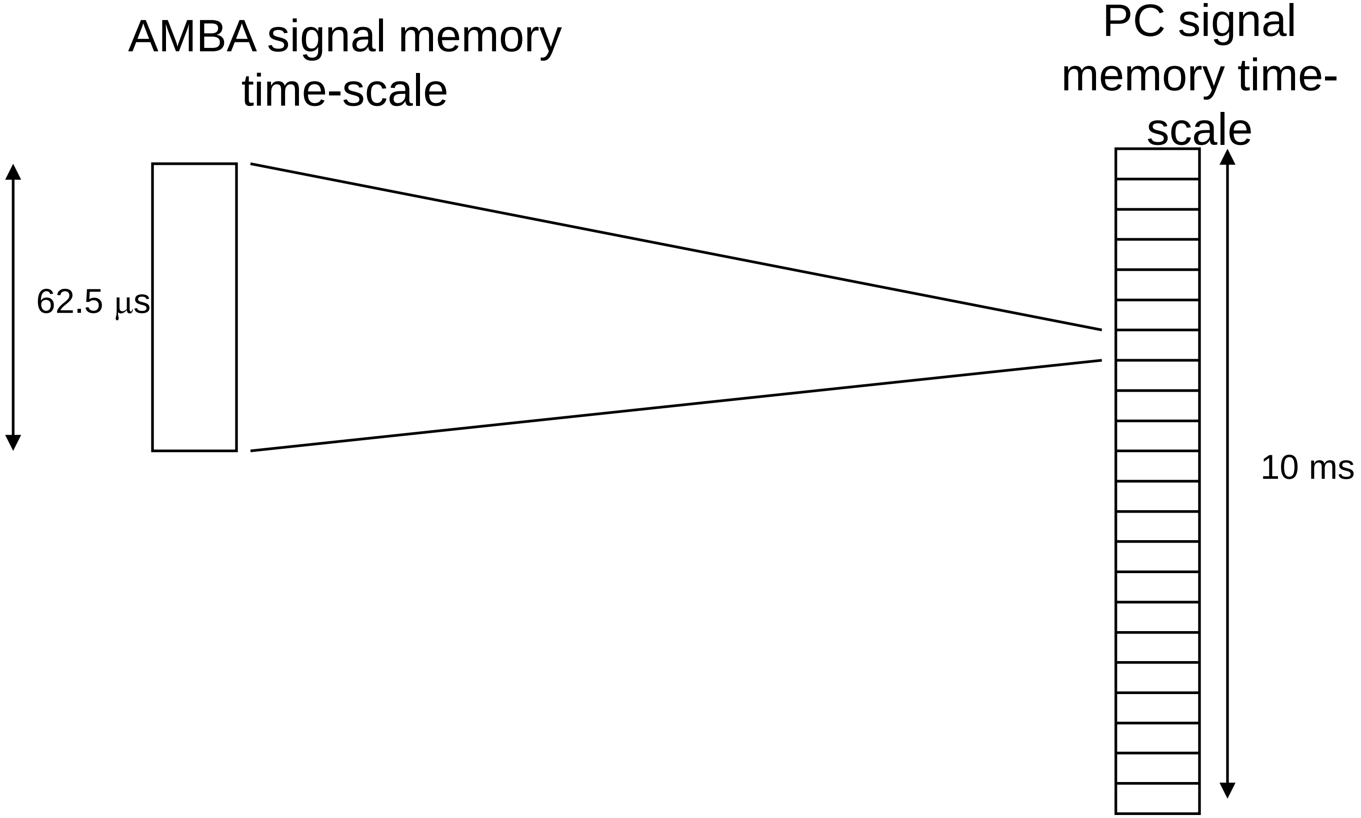
- **AMBA can burst at a peak rate of 52 MHz, very comfortable.**
- **PCI DMA controller (GRPCI) on AMBA can't do quite this but it's close enough**
- **Acquisition unit stores blocks (minimum 2) of a programmable size (<= 1Kbyte) and generates an interrupt to CPU at the end of each block. The CPU programs a 2 DMAs (one for each chain) AMBA->PCI (RX) or PCI->AMBA (TX)**
- **TX and RX cannot occur at the same time (time-division duplex)**

Acquisition (AMBA)



- Blocks are 480 samples of signal (62.5 μs)

PC Memory View

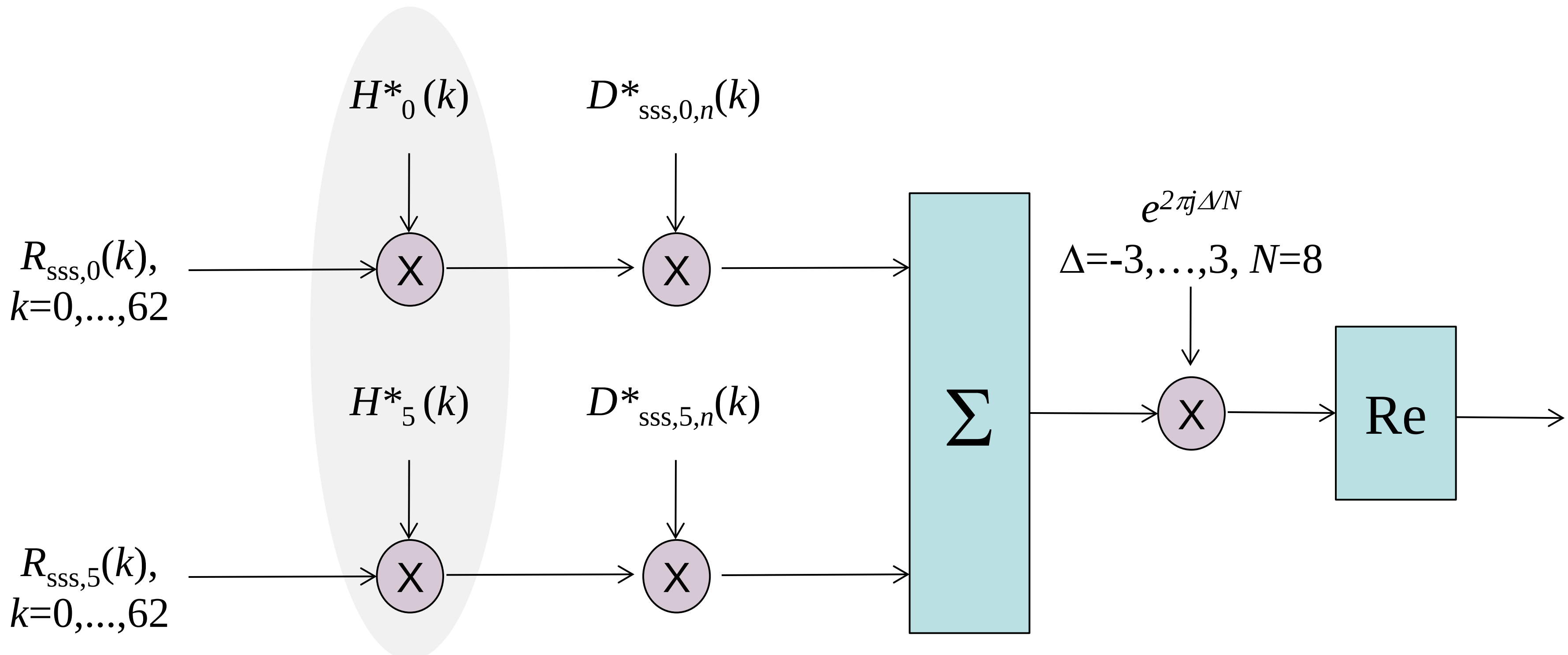


LTE softmodem application

- `targets/RTAI/USER/lte-softmodem.c`
- **Top-level UE/eNB real-time thread**
  - Synchronization to TX/RX counter using polling and sleep
  - Invokes inner-modem DSP processing (`phy_procedures_lte_ue` or `phy_procedures_lte_eNB`)
- **Other real-time threads for ULSCH/DLSCH decoding**
  - Woken up on reception of a ULSCH/DLSCH
  - Make use of multi-core CPUs to parallelize inner-MODEM and channel decoding (turbo-decoder)
  - Multiple decoding threads for higher throughput

BACKUP SLIDES

Secondary Synchronization RX



$H^*_0(k)=D_{pss,0,u}(k)\, R_{sss,0}^*(k)$, $k=0,\ldots,62$
$H^*_5(k)=D_{pss,5,n}(k)\, R_{sss,5}^*(k)$, $k=0,\ldots,62$
PSS-based channel estimates