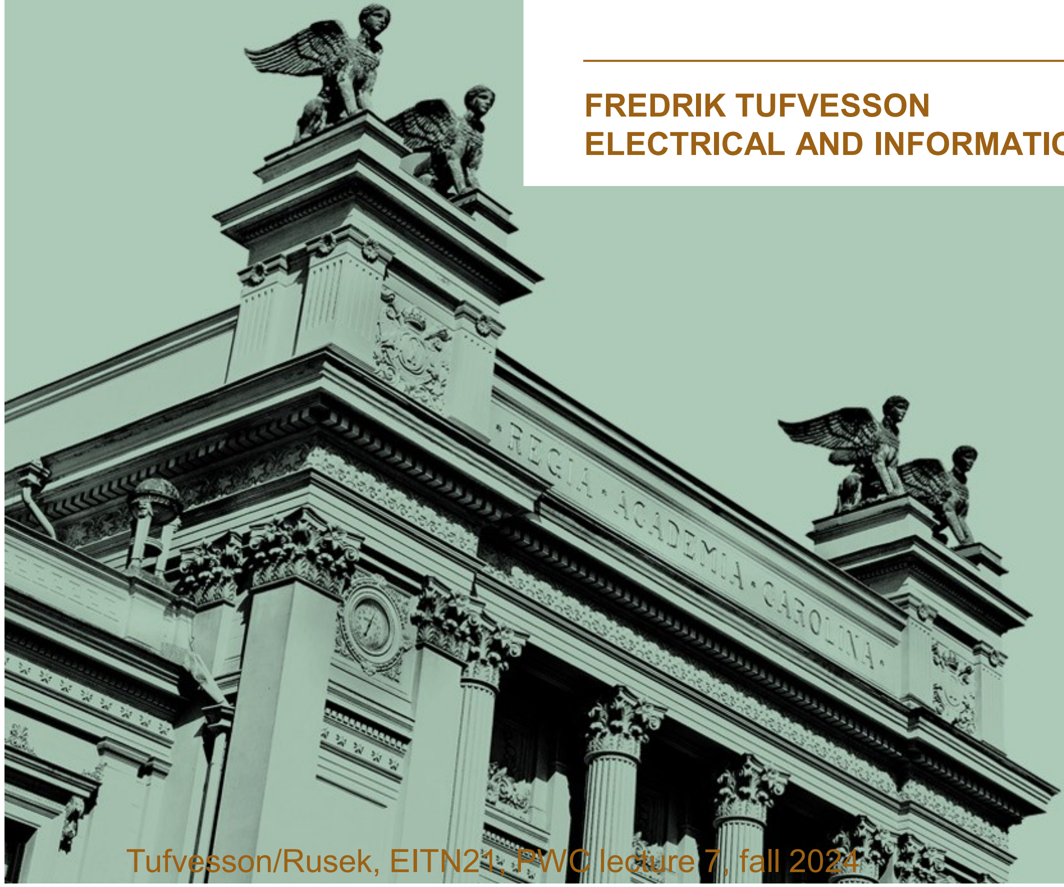
Lund University

Project in Wireless Communication

# Lecture 7: Software Defined Radio

FREDRIK TUFVESSON
ELECTRICAL AND INFORMATION TECHNOLOGY



## Project overview, part two: the audio channel

**Part two is divided into two tasks**

Task 1 – the basic link: Implement an OFDM system and send a data sequence from one computer to the other via the audio channel and decode.

Task 2 – the advanced link: Implement the packet based full duplex system on the audio channel with ARQ.

- Deadline is Sunday Dec 1, 2024 for part two

# Project overview, part three: the radio channel

Implement a basic OFDM based file transfer system over the radio channel using the ADALM Pluto SDR

**Part three is also divided into two tasks**

Task 1 – the basic link: Implement an OFDM transceiver and send a file between the Tx and Rx part of the same Pluto SDR.

Task 2 – the advanced link: Transfer the file from one Pluto to another Pluto over the radio channel.

- Deadline is Friday Jan 5, 2025 for part three, oral presentation of reports Jan 7-10

The following slides are to a large extent based on the book

*Software-Defined Radio for Engineers,*

by Travis F. Collins, Robin Getz, Di Pu, and Alexander M. Wyglinski,

2018, ISBN-13: 978-1-63081-457-1

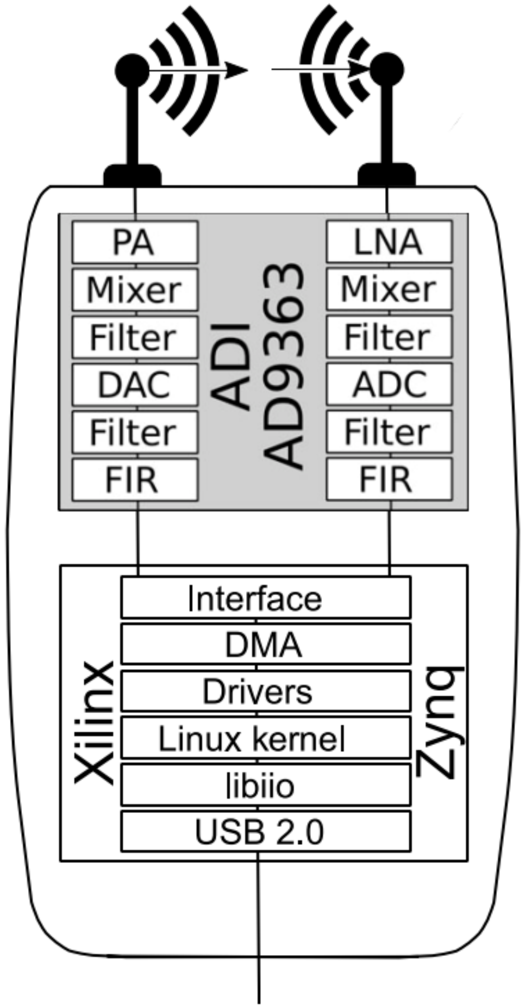
Don't miss to read it,

especially chapter 5

## The PLUTO Software Defined Radio

- The Pluto SDR includes:
  - An analog RF section (antenna, RF filters, input mux, LNA, gain, attenuation, mixer)
  - Analog baseband part (analog filters, ADC or DAC) is implemented in the AD9363, Integrated RF Agile Transceiver
  - Digital signal processing unit for dedicated RF processing
  - Xilinx Zynq' FPGA for further signal processing.
  - ARM cortex 9 processor
- Antenna and RF filters are expected to be done outside the Pluto SDR and are the responsibility of the end user

## What is inside the box?

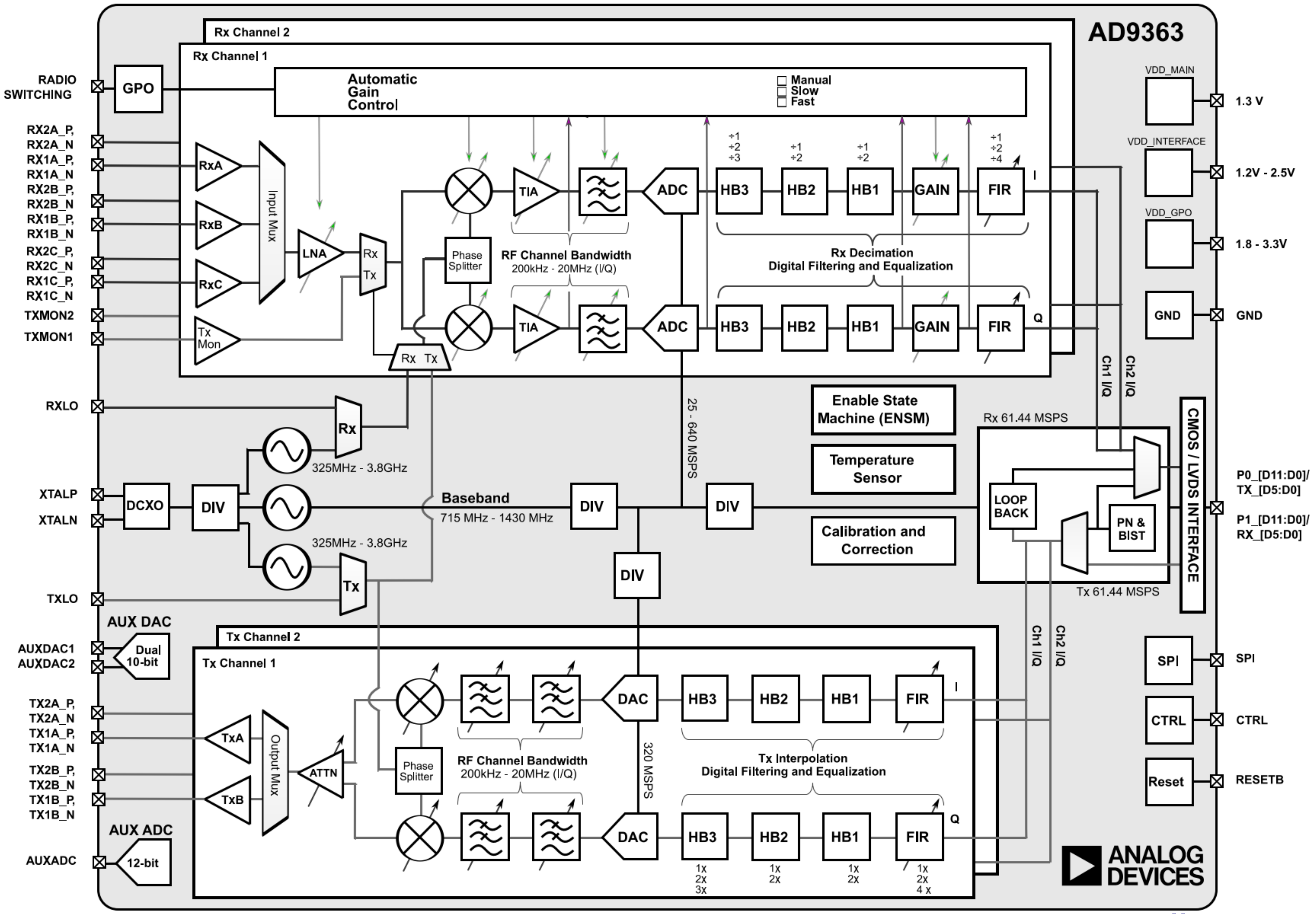


## Transmit specifications

- Center frequency: 300 MHz – 3.8 GHz, 70 MHz – 6 GHz via software modification and reduced specifications.
- Channel bandwidth 200 kHz – 20 MHz,
- Sample rate: 65.1 kSPS–61 MSPS
- 2.4 Hz LO step size, 5 Hz sample rate step size
- Modulation accuracy (EVM): 40 dB (typical, not measured on every unit)
- 12-bit DACs

## Receive specifications

- Center frequency: 300 MHz – 3.8 GHz, 70 MHz – 6 GHz via software modification and reduced specifications.
- Channel bandwidth 200 kHz–20 MHz.
- Sample rate 65.1 kSPS–61 MSPS
- 2.4 Hz LO step size, 5 Hz sample rate step size
- Modulation accuracy (EVM): 40 dB (typical, not measured on every unit)
- 12-bit ADCs


AD9363
Rx Channel 2
Rx Channel 1
Automatic Gain Control
Manual
Slow
Fast
RADIO SWITCHING
GPO
RX2A_P, RX2A_N
RX1A_P, RX1A_N
RX2B_P, RX2B_N
RX1B_P, RX1B_N
RX2C_P, RX2C_N
RX1C_P, RX1C_N
TXMON2
TXMON1
RxA
RxB
RxC
Tx Mon
Input Mux
LNA
Rx Tx
Phase Splitter
TIA
ADC
HB3
HB2
HB1
GAIN
FIR
I
Q
÷1 ÷2 ÷3
÷1 ÷2
÷1 ÷2
÷1 ÷2 ÷4
RF Channel Bandwidth
200kHz - 20MHz (I/Q)
Rx Decimation
Digital Filtering and Equalization
25 - 640 MSPS
Ch1 I/Q
Ch2 I/Q
VDD_MAIN
1.3 V
VDD_INTERFACE
1.2V - 2.5V
VDD_GPO
1.8 - 3.3V
GND
RXLO
Rx
325MHz - 3.8GHz
XTALP
XTALN
DCXO
DIV
Baseband
715 MHz - 1430 MHz
Enable State Machine (ENSM)
Temperature Sensor
Calibration and Correction
Rx 61.44 MSPS
LOOP BACK
PN & BIST
Tx 61.44 MSPS
CMOS / LVDS INTERFACE
P0_[D11:D0]/ TX_[D5:D0]
P1_[D11:D0]/ RX_[D5:D0]
TXLO
Tx
AUX DAC
AUXDAC1
AUXDAC2
Dual 10-bit
Tx Channel 2
Tx Channel 1
TX2A_P, TX2A_N
TX1A_P, TX1A_N
TX2B_P, TX2B_N
TX1B_P, TX1B_N
TxA
TxB
Output Mux
ATTN
DAC
320 MSPS
Tx Interpolation
Digital Filtering and Equalization
1x 2x 3x
1x 2x
1x 2x
1x 2x 4 x
AUX ADC
AUXADC
12-bit
SPI
CTRL
Reset
RESETB
ANALOG DEVICES

## Input/output

- USB 2 OTG (480 Mbits/seconds), *device mode*
  - libiio USB class, for transfering IQ data from/to the RF device to the host
  - Network device, provides access to the Linux on the Pluto device
  - USB serial device, provides access to the Linux console on the Pluto device
  - Mass storage device

## Input/output

- USB 2 OTG (480 Mbits/seconds), *host mode*
  - Mass storage device, plug in a thumb drive, and capture or playback waveforms
  - Wifi dongle, access the Pluto SDR via WiFi
  - Wired LAN, access the Pluto SDR via wired LAN
  - External power, for when using the Pluto SDR in host mode.

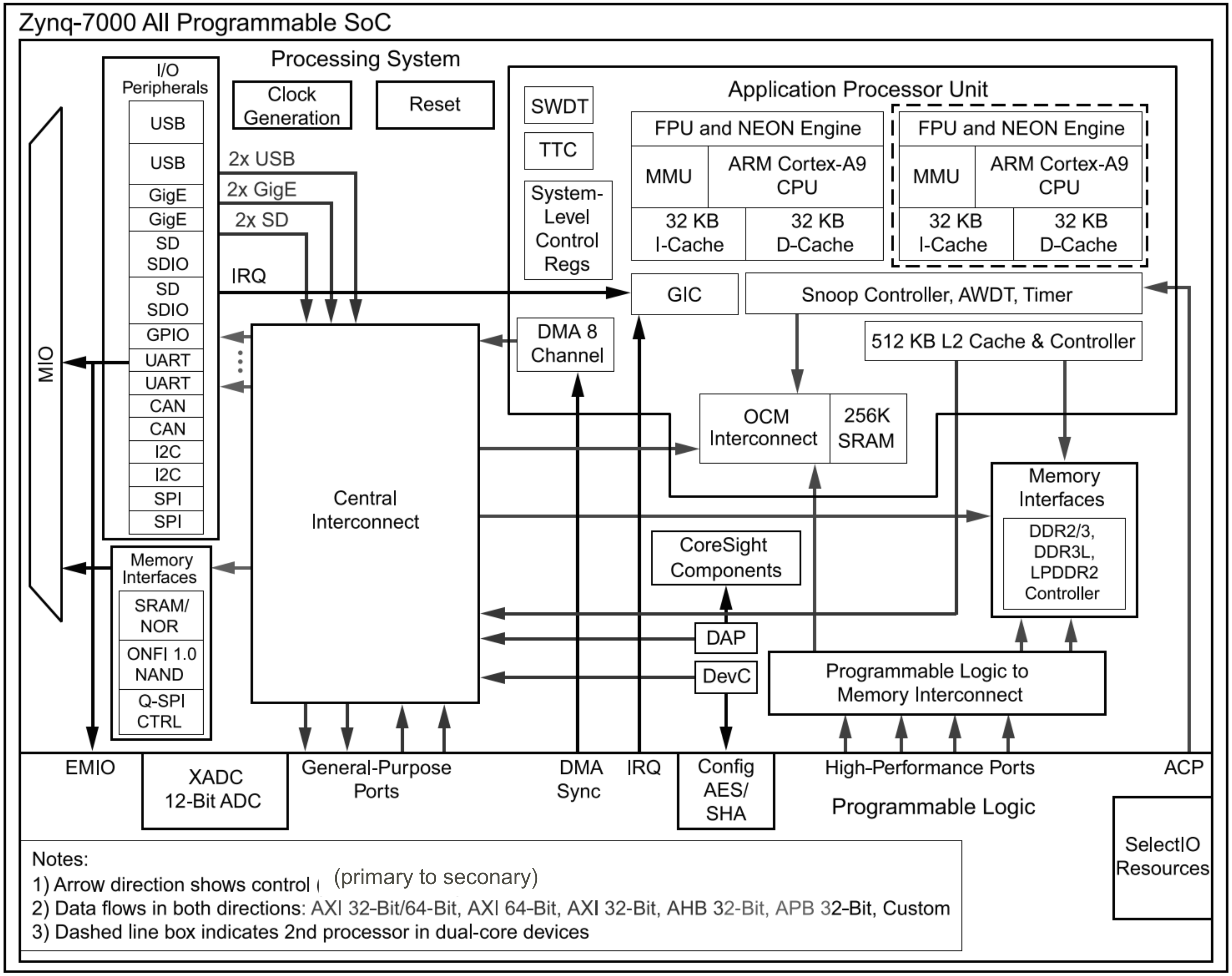
## Input/output

Libiio is the userspace library for accessing local and remote IIO devices, in our case both in the ARM, and on the host

- libiio is used to interface to the Linux industrial input/output (IIO) subsystem.
- libiio can be natively used on an embedded Linux target (local mode) or to
- communicate remotely to that same target from a host Linux, Windows, or MAC over USB, Ethernet, or Serial.

## Xilinx Zynq System on Chip

- Once the data is digitized it is passed to the Xilinx Zynq System on Chip
- The Zynq-7000 family includes and FPGA for flexibility and scalability,
  - Integrated ARM Cortex-A9 based processing system (PS) and programmable logic (PL).
- The Zynq is the used in the Pluto SDR as the main controller



Notes:
1) Arrow direction shows control (primary to seconary)
2) Data flows in both directions: AXI 32-Bit/64-Bit, AXI 64-Bit, AXI 32-Bit, AHB 32-Bit, APB 32-Bit, Custom
3) Dashed line box indicates 2nd processor in dual-core devices

DS190_01_072916

UNIVERSITY

## Matlab requirements

- MATLAB can be used as a cross-platform IIO client to interface with the Pluto.
- SDRToolboxes that are required to use MATLAB with the PlutoSDR:
  - DSP System Toolbox
  - Signal Processing Toolbox
  - Communications System Toolbox

# Using PLUTO with MATLAB

ADALM-PLUTO Radio Support from Communications Toolbox

- Install the support package if using your own laptop

Matlab 2017b or later is highly recommended.

# Using PLUTO with MATLAB

The two system objects provided in the hardware support package (HSP) for Pluto SDR are:

- comm.SDRRxPluto: Pluto SDR Receiver System object
- comm.SDRTxPluto: Pluto SDR Transmitter System object

```
rx = sdrrx('Pluto')
rx =
  comm.SDRRxPluto with properties:
  DeviceName: 'Pluto'
  RadioID: 'usb:0'
  CenterFrequency: 2.4000e+09
  GainSource: 'AGC Slow Attack'
  ChannelMapping: 1
  BasebandSampleRate: 1000000
  OutputDataType: 'int16'
  SamplesPerFrame: 3660
  ShowAdvancedProperties: false
```

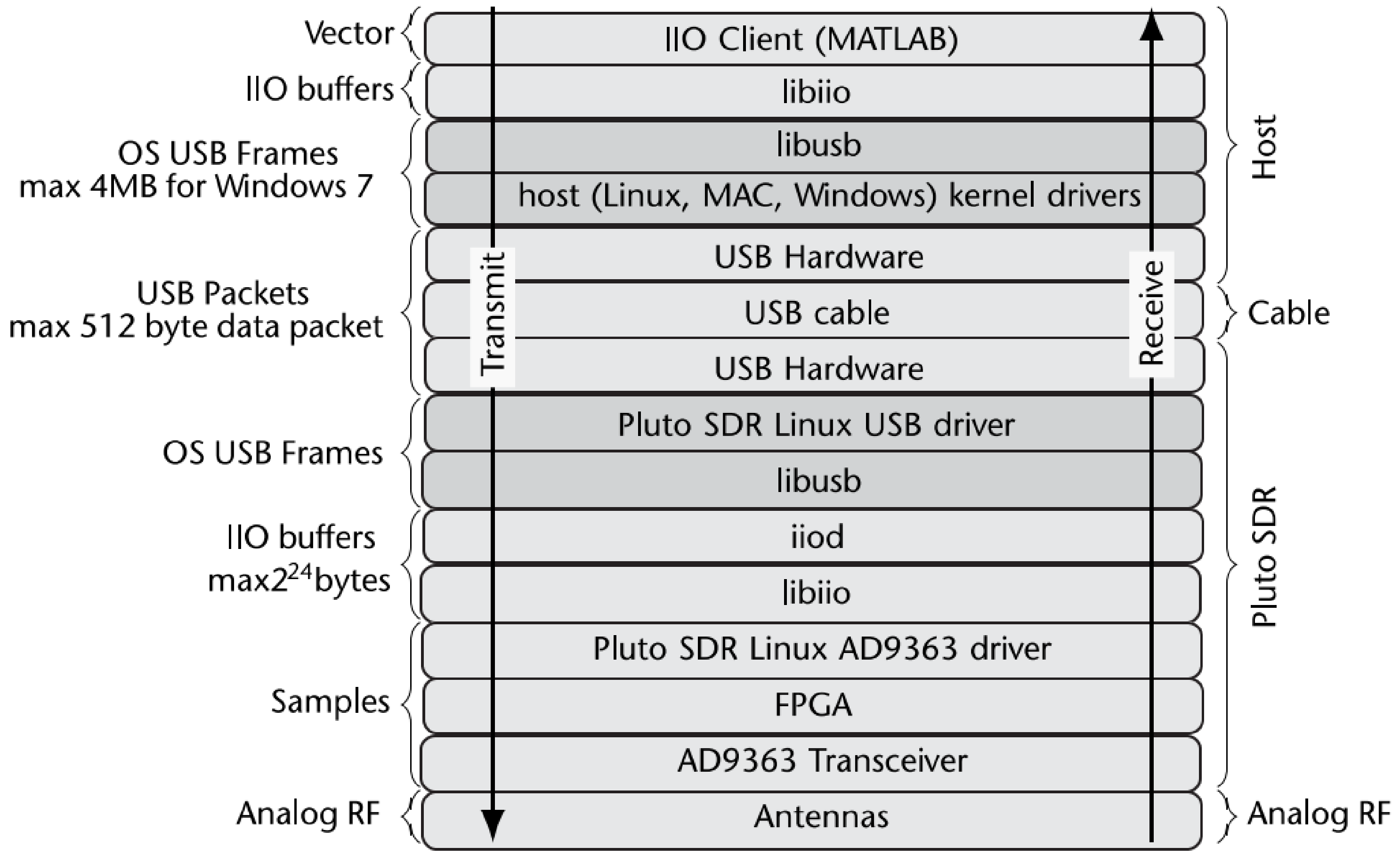
## Basic object parameters

- **CenterFrequency** defines the RF center frequency in Hertz. Note there are separate Rx and Tx LO, on the Pluto SDR, and these are managed separately
- **BasebandSampleRate** defines the sample rate of the in-phase/quadrature receive chains, respectively. Note, there is only one clock generator for both the ADC and DAC
- **GainSource** has three possible options:
  - Manual, AGC Slow Attack, and AGC Fast Attack.

## Basic object parameters

- The **ChannelMapping** attribute for the Pluto SDR can only be set to 1.
- **OutputDataType** determines the format data is provided out of the object.
  - Technically, from the AD9363 and libiio, MATLAB can only receive 16-bit complex integers, but we can tell MATLAB to cast them to other data types. Typically, we will cast them to doubles
- **SamplesPerFrame** determines the number of samples in the buffer or frame that is passed to MATLAB from iiod.

# Transmitting and receiving data



## Capturing data with PLUTO

There are three basic methods to capture and process data:

1. Read and process
2. Save, load and process
3. Stream processing

# Read and process

```
%% Template 1
% Perform data collection then offline processing
data = zeros(frameSize, framesToCollect);
% Collect all frames in continuity
for frame = 1:framesToCollect
    [d,valid,of] = rx();
    % Collect data without overflow and is valid
    if ~valid
        warning('Data invalid')
    elseif of
        warning('Overflow occurred')
    else
        data(:,frame) = d;
    end
end

% Process new live data
sa1 = dsp.SpectrumAnalyzer;
for frame = 1:framesToCollect
    sa1(data(:,frame)); % Algorithm processing
end
```

# Save, load and process

```
% Save data for processing
bfw = comm.BasebandFileWriter(’PlutoData.bb’,...
    rx.BasebandSampleRate,rx.CenterFrequency);
% Save data as a column
bfw(data(:));
bfw.release();
```

```
%% Template 2
% Load data and perform processing
bfr = comm.BasebandFileReader(bfw.Filename, ’SamplesPerFrame’,frameSize);
sa2 = dsp.SpectrumAnalyzer;
% Process each frame from the saved file
for frame = 1:framesToCollect
    sa2(bfr()); % Algorithm processing
end
```

## Stream processing

```
%% Template 3
% Perform stream processing
sa3 = dsp.SpectrumAnalyzer;
% Process each frame immediately
for frame = 1:framesToCollect
    [d,valid,of] = rx();
    % Process data without overflow and is valid
    if ~valid
        warning('Data invalid')
     else
         if of
             warning('Overflow occurred')
         end
         sa3(d); % Algorithm processing
     end
 end
```

LUND UNIVERSITY

## Transmitting data

As the spectrum use is regulated we are only allowed to transmit at certain frequencies, given that we fullfill the rules at that particular band.

**Use the ISM bands!**
**Nothing but those.**

Anytime the Pluto SDR is powered on, the transceiver is activated and begins to operate even if the user did not intend to. When powered on Pluto SDR will transmit!

## The ISM bands

The industrial, scientific and medical (ISM) radio bands are radio bands (portions of the radio spectrum) reserved internationally for the use of radio frequency (RF) energy for industrial, scientific and medical purposes other than telecommunications.

Radio communication services operating within these bands must accept harmful interference which may be caused by these applications, and they are subject to specific rules

| | |
|---:|---:|
| 863 - 870 MHz | Short range devices |
| 2.4 - 2.5 GHz | ISM |
| 5.725 - 5.875 GHz | ISM |

## Self interference

There are two ways to reduce the self interference when receiving:

1. instantiate a transmitter System object (comm.SDRTxPluto) and write a vector of zeros to the object
2. shift the LO of the transmitter to a frequency beyond the receive bandwith.

## Frequency offsets

When transmitting between two separate devices there might be a frequency offset between the local oscillators

- For PLUTO the internal LO is rated at 25 PPM

*What does this mean in possible offset?*

We can model the received signal without noise as

$$r(t) = s(t)e^{j2\pi \Delta_f t/f_s}$$

## Carrier frequency offset estimation

Use the repeated pilot also for carrier frequency offset estimation

$$p(k) = \sum_{m=0}^{L-1} r^*(k+m)r(k+m+L)$$

$$p(k) = \sum_{m=0}^{L-1} s(k+m)^* e^{-j2\pi \Delta_f (k+m)/f_s} s(k+m+L) e^{j2\pi \Delta_f (k+m+L)/f_s}$$

$$p(k) = e^{j2\pi \Delta_f (L)/f_s} \sum_{m=0}^{L-1} s(k+m)^* s(k+m+L)$$

$$\boxed{p(k) = e^{j2\pi \Delta_f (L)/f_s} \sum_{m=0}^{L-1} |s(k+m)|^2}$$

# Estimating the offset

The phase of the synch signal can be written as

$$\phi = \frac{2\pi L \Delta_f}{f_s}$$

The maximum detectable frequency offset with this method is

$$\Delta_{f,max} = \frac{f_s}{2LN}$$

A large symbol length (number of sub-carriers) reduces the estimation range!

## Some advice

- Start simple
- Verify that the SDR is working with known functions first
- Verify your code stepwise
- Look at your signals by plotting them
- Look at what happens with the continous pilot symbols
- The antennas are not optimized for 2.4 GHz operation, but they work there.
- Verify your CFO compensation with the loopback cable and intentional frequency offset
- Try some of the more advanced stuff found on the web.

Should we have a demo day in January?

## Resources

- Recommended reading, chapter 5:
  - Software-Defined Radio for Engineers, by Travis F. Collins, Robin Getz, Di Pu, and Alexander M. Wyglinski, 2018, ISBN-13: 978-1-63081-457-1.
- There is a free pdf of the book available, see http://www.analog.com/en/education/education-library/software-defined-radio-for-engineers.html.
- Analog devices, Pluto webpage https://www.analog.com/en/design-center/evaluation-hardware-and-software/evaluation-boards-kits/adalm-pluto.html#eb-overview

# PLUTO and MATLAB

You have to install the support package for PLUTO to MATLAB

- Pluto and Matlab:

https://se.mathworks.com/hardware-support/adalm-pluto-radio.html

- Tutorials and examples

https://se.mathworks.com/help/supportpkg/plutoradio/getting-started-with-communications-system-toolbox-support-package-for-pluto-radio.html

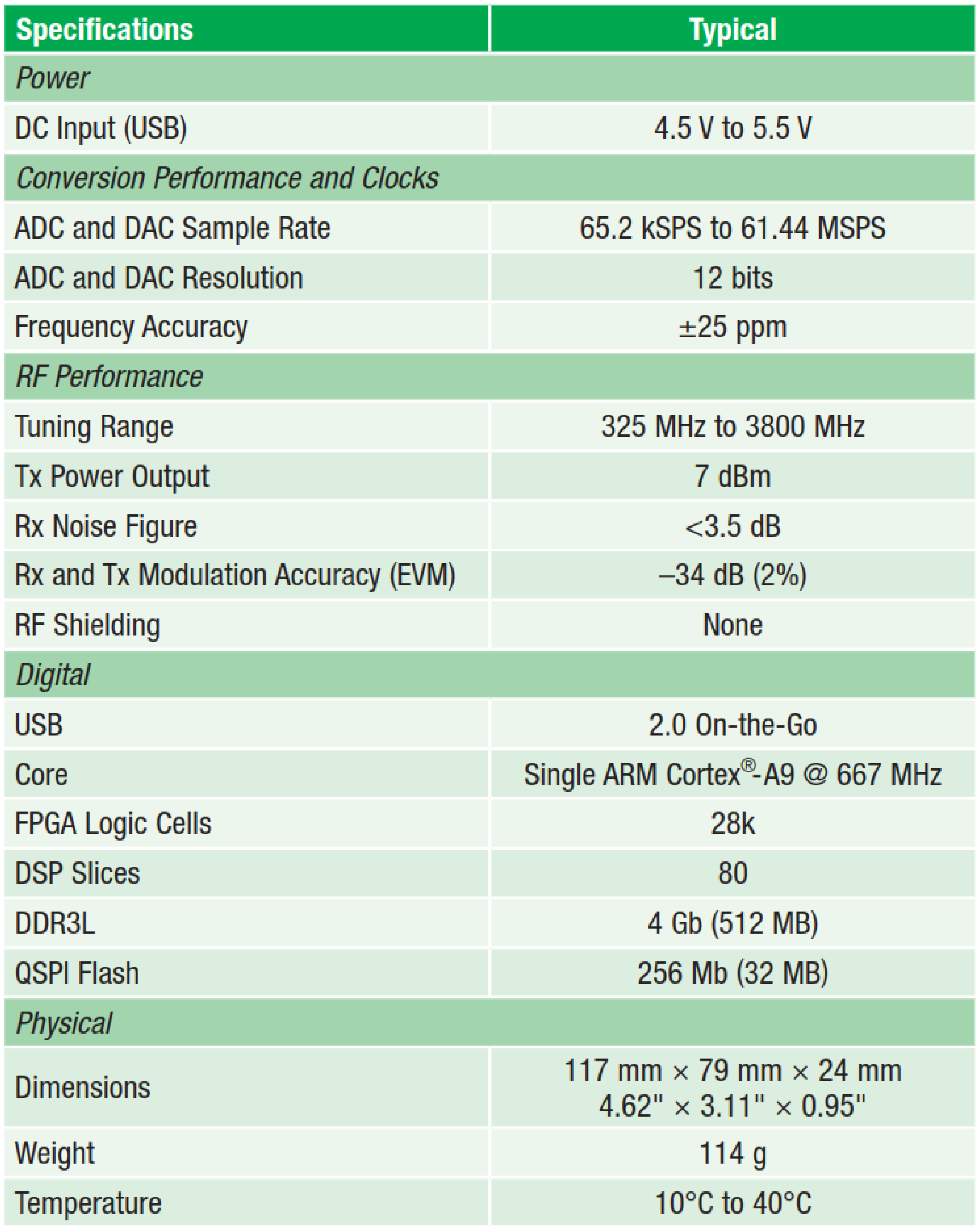
https://se.mathworks.com/matlabcentral/fileexchange/69417-introductory-communication-systems-course-using-sdr?s_tid=FX_rc1_behav&s_tid=mwa_osa_a

## PLUTO and Python

- It is recommend installing an Ubuntu 22 VM. Alternatively, if you’re on Windows 11, Windows Subsystem for Linux (WSL) using Ubuntu 22 tends to run fairly well and supports graphics out-of-the-box,
- See https://pysdr.org/content/pluto.html for further details

## Resources, GNU radio

- GNU Radio is a free & open-source software development toolkit that provides signal processing blocks to implement software radios
- GNU Radio is compatible with PLUTO and performs the radio related signal processing, written in either C++ or Python
- https://www.gnuradio.org/
- https://wiki.analog.com/resources/tools-software/linux-software/gnuradio



| Specifications | Typical |
|---|---|
| *Power* | |
| DC Input (USB) | 4.5 V to 5.5 V |
| *Conversion Performance and Clocks* | |
| ADC and DAC Sample Rate | 65.2 kSPS to 61.44 MSPS |
| ADC and DAC Resolution | 12 bits |
| Frequency Accuracy | ±25 ppm |
| *RF Performance* | |
| Tuning Range | 325 MHz to 3800 MHz |
| Tx Power Output | 7 dBm |
| Rx Noise Figure | <3.5 dB |
| Rx and Tx Modulation Accuracy (EVM) | –34 dB (2%) |
| RF Shielding | None |
| *Digital* | |
| USB | 2.0 On-the-Go |
| Core | Single ARM Cortex®-A9 @ 667 MHz |
| FPGA Logic Cells | 28k |
| DSP Slices | 80 |
| DDR3L | 4 Gb (512 MB) |
| QSPI Flash | 256 Mb (32 MB) |
| *Physical* | |
| Dimensions | 117 mm × 79 mm × 24 mm<br>4.62" × 3.11" × 0.95" |
| Weight | 114 g |
| Temperature | 10°C to 40°C |

LUND
UNIVERSITY