

# **2G hopping channels sniffing**

Cyberspectrum 2017,  
Vegas

Pedro Cabrera (@PCabreraCamara)

# Objectives


---

- Channel hopping sniffing using SDRs HackRF and BladeRF
- Review the Bogdan version of airprobe, able to capture multiple channels using USRP N210.
- Review of the most basic terms of SDR that allows to understand the digital processing tools of applied signals that allow us to capture the frequency hops.
- Demonstrate the ability to intercept from different devices, like ARM smartphones/tablets.



# Context


- Airprobe: Holland, 2009 – **Hacking at Random**, Harald Welte

 **Hacking at Random**  
*Hacking happens*


HAR2009 - PREVIEW

---

**Airprobe**  
*Monitoring GSM traffic with USRP*



The GNU Radio project, and the associated Universal Software Radio Peripheral hardware, has for the first time put access to wideband radio reception and digital signal processing into the hands of the average hacker. The airprobe project focuses on a software stack that provides GSM mobile phone reception and decoding. This talk gives a gentle introduction into GSM, the airprobe software and its practical usage.

Speakers	
	<a href="#">Harald Welte</a>

Schedule	
Day	Saturday - 2009-08-15
Room	Monty Hall
Start time	16:00
Duration	01:00

- BlackHat USA, 2010 – Harald Welte & Karsten Nohl: “A real-world practical A5/1 attack using airprobe and Kraken”



# The Airprobe family

---

- (2011) chemeris/airprobe-gprs: *airprobe with a simple GPRS support. Single-ARFCN downlink and uplink supported. No frequency-hopping supported. No Timing Advance for uplink is supported.*
- (2014) scateu/airprobe-3.7-hackrf-patch: GNU Radio 3.7 adaptation.
- (2014) BogdanDIA/airprobe-hopping: *airprobe for frequency hopping GSM.*
- (8/2014) NSA playset, Twilight Vegetable (bootable image)
- (2015) ptrkrysik/gr-gsm: *the gr-gsm project is based on the gsm-receiver written by Piotr Krysik for the Airprobe project.*



# Airprobe limitations

- *Airprobe has been extended to fully support decoding of TCH/F (FACCH, SACCH and traffic), as well as SDCCH/SACCH control channels, and to specify the timeslot and physical channel configuration from the command line. Using this, you can:*
  - *decode the AGCH, wait for an IMMEDIATE ASSIGNMENT of a SDCCH*
  - *decode that very SDCCH and wait until encryption is turned on*
  - *dump an encrypted burst where you have sufficient known plaintext*
  - *use a different program to actually recover the A5/1 ciphering key*
  - *feed that key into airprobe and decrypt+decode the ASSIGNMENT COMMAND of the TCH*
  - *use airprobe to decrypt+decode that assigned TCH/F*
- *So what are the limitations? Well, so far this only works on non-hopping cells with a single ARFCN. The limitations are those of the receiver hardware (and SDR software), and not really limitations of the airprobe GSM decoder or the actual software tools.*

Source: [http://laforge.gnumonks.org/blog/20100730-practical\\_gsm\\_a51\\_attack/](http://laforge.gnumonks.org/blog/20100730-practical_gsm_a51_attack/)



# Channel Hopping alternatives



- Alternative #1: **OsmocomBB**. *Harald Welte, Dieter Spaar, Andreas Eversberg, Sylvain Munaut. Berlin 2010*
- Demonstration of adaptation of the original project to capture and decipher TCH channels with frequency hopping: : 27C3 “GSM Sniffing”, *Karsten Nohl y Sylvain Munaut, Berlin 2010.*

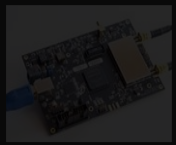
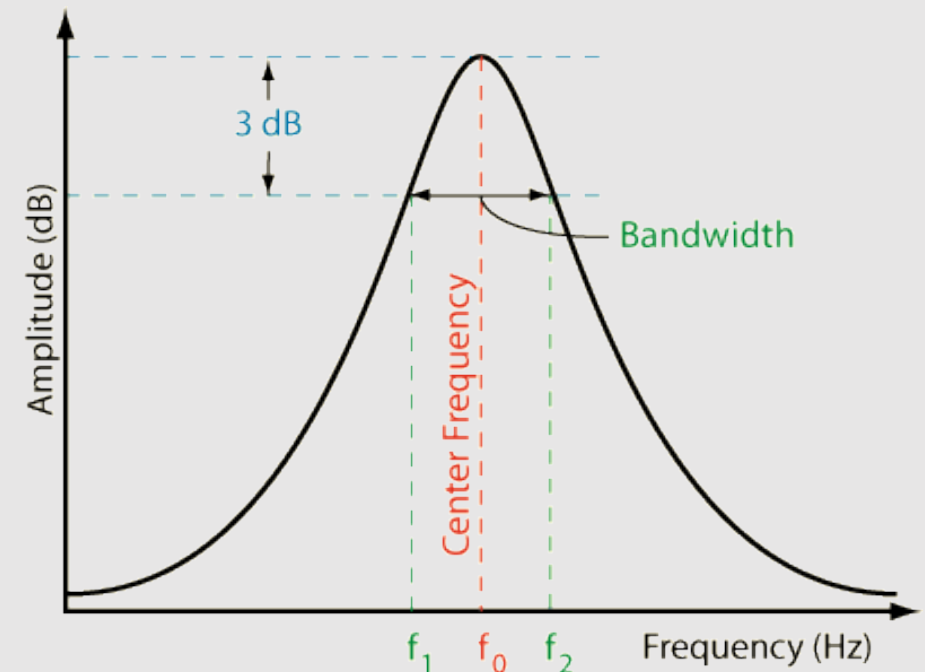
· After the capacity demonstration, the code used in the event has not been published, nor the secondary tools necessary to perform the steps.

· Alternative #2: **GR-GSM**. “Decoding hopping channels can be achieved by feeding one input stream per hopping channel into the GSM Receiver block and connecting the CX port to a CX Channel Hopper block. At the current stage of development however, it is computationally expensive to split a wideband capture into multiple streams in real time. Therefore, the `grgsm_channelize` app should be used to perform this task as a preprocessing step.”



# DSP concepts

- **Bandwidth:** the bandwidth describes the range of frequencies an oscilloscope can accurately measure. Is measured between the lower and upper frequency points where the signal amplitude falls to -3 dB below the passband frequency (is the difference between the corner frequencies).
- **Sample Rate:** is the frequency at which the ADC converts the analog input waveform to digital data.
- **Decimation:** is the process of reducing the sampling rate of a signal. Is a two-step process of lowpass filtering followed by an operation known as downsampling. Complementary to interpolation, which increases sampling rate, it is a specific case of sample rate conversion.
- **Clock Rate:** we refer to the clock speed of the ADC, which will be adjusted using a decimation to reduce sampling to satisfy the sample rate indicated by the user.





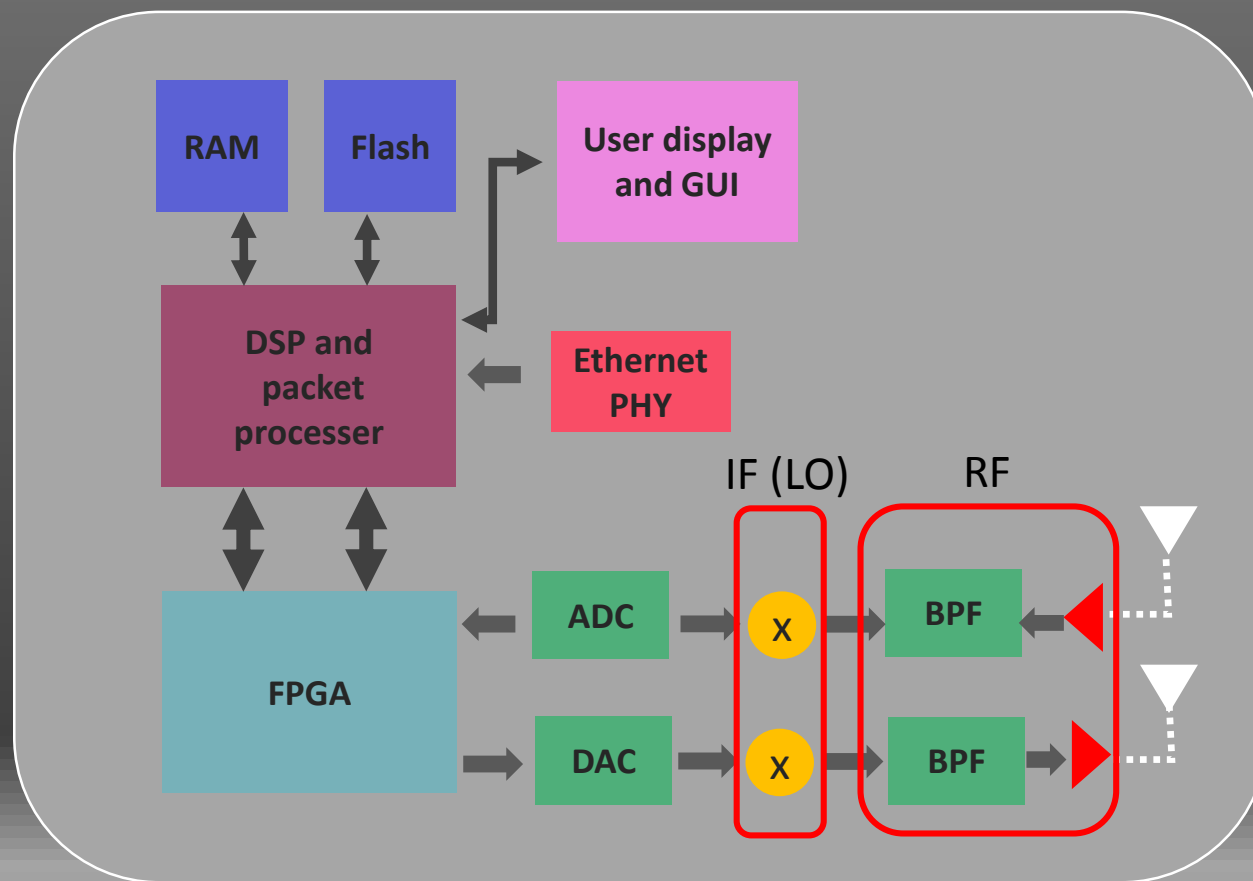
# DSP concepts II

*“Three general types of bandwidth:*

- analog bandwidth*
- the FPGA processing bandwidth*
- host bandwidth”*

*From ETTUS*

The system bandwidth is generally the minimum of this three bandwidths.



The mixer translates the RF signal to IF frequencies. The RF and LO signals mix to produce a difference frequency known as IF frequency ( $f_{IF} = |f_{RF} - f_{LO}|$ ).





# DSP concepts III

---

The **Nyquist Sampling Theorem** explains the relationship between the sample rate and the frequency of the measured signal. It states that the sample rate  $f_s$  must be greater than twice the highest frequency component of interest in the measured signal. This frequency is often referred to as the Nyquist frequency,  $f_N$ .

Synthesis filter banks: A basic operation in multirate signal processing is to decompose a signal into a number of sub band components, which can be processed at a lower rate corresponding to the bandwidth of the frequency bands. **Polyphase Channelizer**

Recommended reading, the paper presented at the Karlsruhe Institute of Technology's Workshop on Software Radio (WSR): <http://www.trondeau.com/examples/2014/1/23/pfb-channelizers-and-synthesizers.html>

## Designing Analysis and Synthesis Filterbanks in GNU Radio

Thomas W. Rondeau  
University of Pennsylvania & Rondeau Research  
Shelburne, VT  
tom@trondeau.com

Timothy J. O'Shea  
The Hume Center  
Virginia Tech  
Arlington, VA  
oshea@vt.edu

Cyberspectrum 2017, Ve

*hopping channels sniffing*



# GSM Frequencies

System	Band	Frequency		Channel Assignment
		Uplink	Downlink	
GSM - 850	850	824.0-849.0	869.0-894.0	128-251
E-GSM - 900	900	880.0-914.8	925.0-959.8	975-1023, 0-124
DCS - 1800	1800	1710.2-1784.8	1805.2-1879.8	512-885
PCS - 1900	1900	1850.0-1910.0	1930.0-1990.0	512-810

Carrier	2G Frequency in MHz Band name			3G Frequency in MHz Band name		
	800	850	1900	850	1700 2100	1900
	SMR	CLR	PCS	CLR	AWS	PCS
AT&T	No	No	No	UMTS	No	UMTS
T-Mobile	No	No	GSM	No	UMTS	UMTS
Sprint	CdmaOne <sup>[4]</sup>	No	CdmaOne	No	No	CDMA2000
Verizon	No	CdmaOne	CdmaOne	CDMA2000	No	CDMA2000
U.S. Cellular	No	CdmaOne	CdmaOne	CDMA2000	No	CDMA2000

Source: [https://en.wikipedia.org/wiki/Cellular\\_frequencies\\_in\\_the\\_US](https://en.wikipedia.org/wiki/Cellular_frequencies_in_the_US)



# GSM & Channel Hopping

```
▷ GSM TAP Header, ARFCN: 0 (Downlink), TS: 0, Channel: BCCH (0)
▲ GSM CCCH - System Information Type 1
▷ L2 Pseudo Length
▷ ... 0110 = Protocol discriminator: Radio Resources Management messages (0x06)
Message Type: System Information Type 1
▲ Cell Channel Description
10...111 = Format Identifier: variable bit map (0x47)
List of ARFCNs = 727 784
▷ RACH Control Parameters
▷ SI 1 Rest Octets
```

```
▷ GSM TAP Header, ARFCN: 0 (Downlink), TS: 0, Channel: PCH (0)
▲ GSM CCCH - Immediate Assignment
▷ L2 Pseudo Length
▷ ... 0110 = Protocol discriminator: Radio Resources Management messages (0x06)
Message Type: Immediate Assignment
▷ Page Mode
▷ Dedicated mode or TBF
▲ Channel Description
0100 0... = SDCCH/8 + SACCH/C8 or CBCH (SDCCH/8): 8
Subchannel: 0
... 010 = Timeslot: 2
110. .... = Training Sequence: 6
...1 .... = Hopping Channel: Yes
Hopping channel MAIO: 0
HSN: 13
▷ Request Reference
```

## Theory:

- GSM devised a combination of TDMA/FDMA as the method to divide the bandwidth among the users.
- Each BaseStation is assigned with one or **multiple frequencies**, and each of this frequency is divided into eight timeslots using a TDMA scheme.

**HSN (hopping sequence number)**: Is used to define the skip sequence of the frequency list.

**MAIO (mobile allocation index offset)**: Is used to set the initial frequency of this list.



# 2010 Airprobe workflow

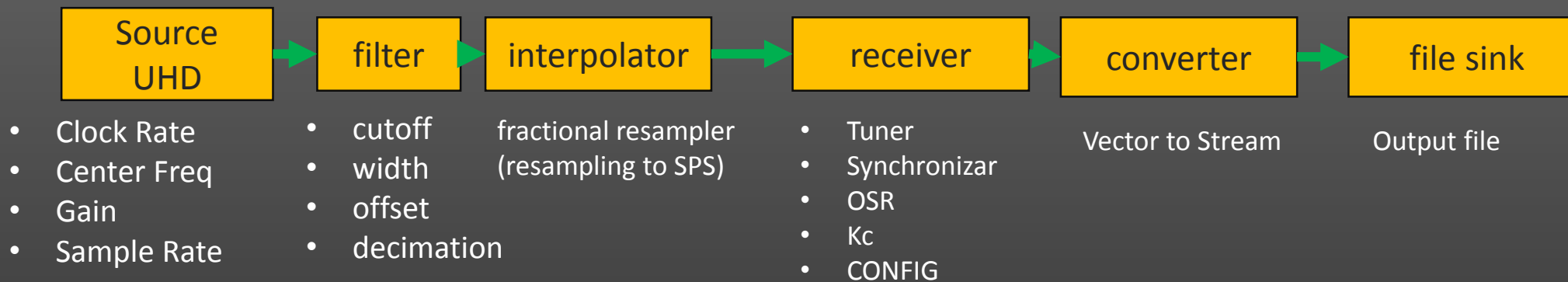
gsm\_receive\_usrp.py:  
GSM\_RECEIVER

synchronizer

Timing offset

tuner

Freq offset



- Input Rate: USRP Sample Rate
- GSM Symb Rate:  $1625000/6$
- SPS:  $\text{InputRate}/\text{GSM Symb Rate}/\text{OSR}$



# Bodgan Airprobe

## Prerequisites:

Use USRP N210 or USRP2 from Ettus. For this work USRP N210 has been used.



Phase	Description
0	Frequency calculation phase and SDR parameters settings (sample rate and center frequency)
1	Capture to a single file with the previous configuration.
2	Polyphase filter applied to split all captured channels into separate files.
3	Decoding the channels under study.

Evaluating GSM A5/1 security on hopping channels

© April 7, 2012    Uncategorized

Paper: Evaluating\_GSM\_hopping\_V1.2.pdf

Fuente: <http://yo3iiu.ro>



# Bodgan Airprobe

0	Frequency calculation phase and SDR parameters settings (sample rate and center frequency)
1	Capture to a single file with the previous configuration.
2	Polyphase filter applied to split all captured channels into separate files.
3	Decoding the channels under study.

```
minARFCN=`echo $CA | awk '{print $1}`
maxARFCN=`echo $CA | awk '{print $NF}`
```

```
ARFCN_fc=$(( ($maxARFCN+$minARFCN)/2))
```

```
if [ $ARFCN_fc -gt 125 ]
then
```

```
    FC=$((1805200000 + 200000*$(( $ARFCN_fc-512))))
```

```
else
```

```
    FC=$((935000000 + 200000*$ARFCN_fc))
```

```
Fi
```

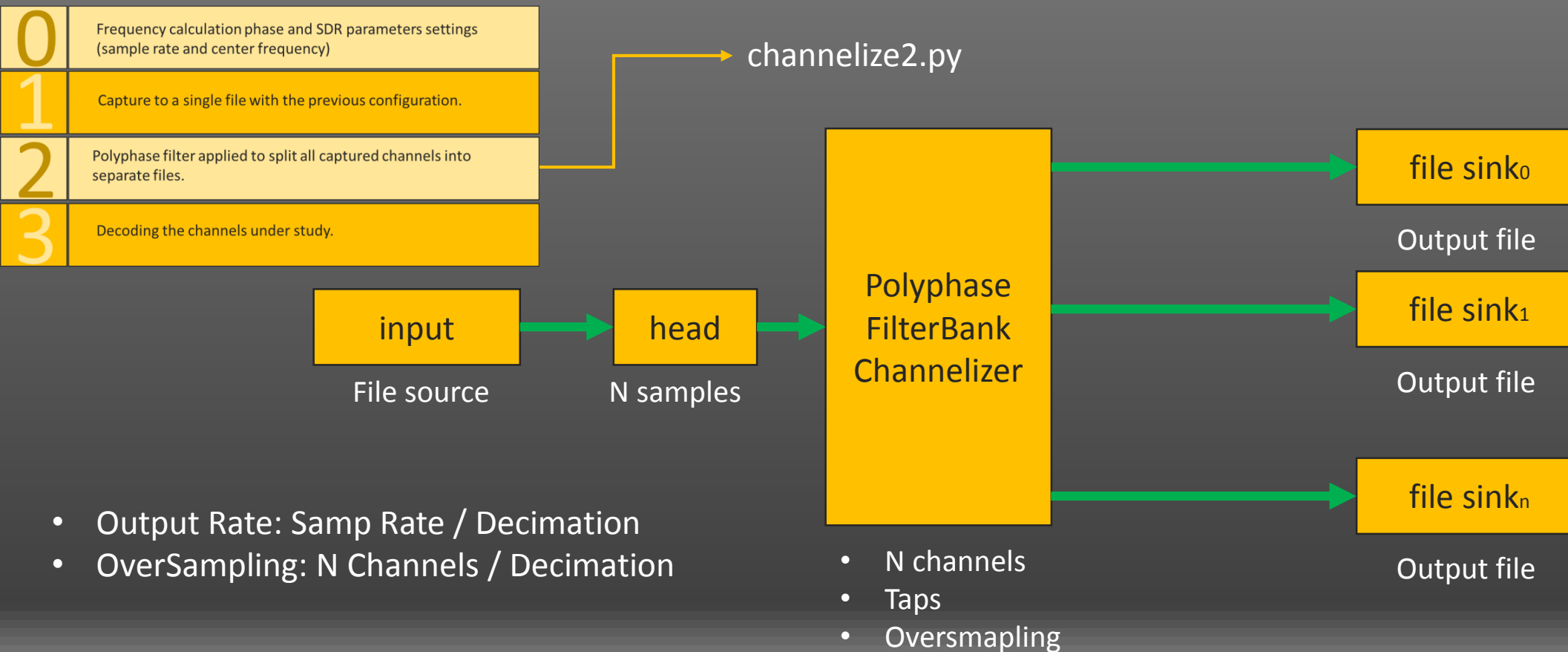
```
BW=$(( ($maxARFCN-$minARFCN+1)*200))
```

```
uhd_rx_cfile -g 76 -f "$FC" --samp-rate="$SR" out/out.cf -N "$NSAMPLES"
```

BW > 10M	<b>SR = 25 Msps</b>
	<i>N canales = 125</i>
	<i>pfbDECIM = 46</i>
	<i>totDECIM = 184</i>
10M => BW > 200k	<b>SR = 10 MSps</b>
	<i>N canales = 50</i>
	<i>pfbDECIM = 17</i>
	<i>totDECIM = 170</i>
BW <= 200k	<b>SR = 574712</b>
	<i>N canales = 1</i>
	<i>pfbDECIM = 1</i>
	<i>totDECIM = 174</i>

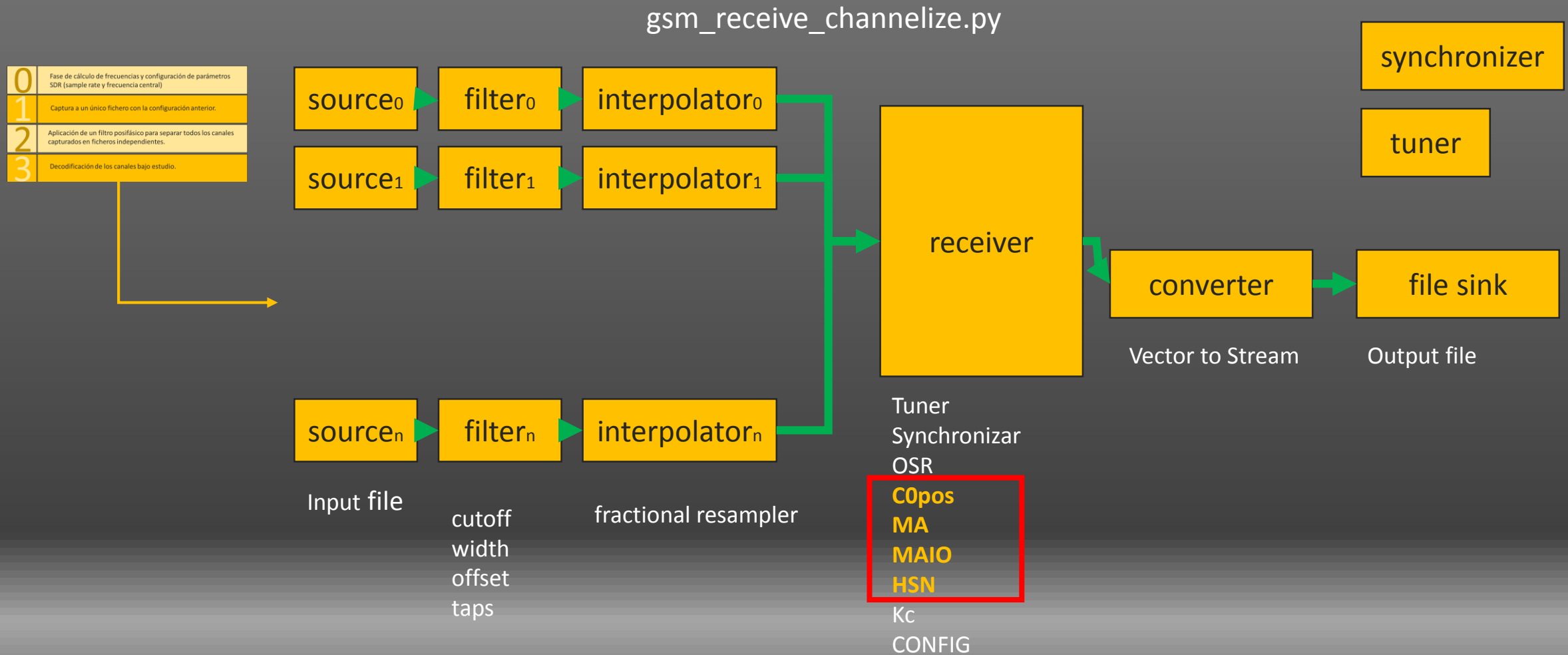


# Bodgan Airprobe





# Bodgan Airprobe



# HackRF/BladeRF

---



## HackRF (HalfDuplex)

- Clockrate: 40Mhz
- 8 bit quadrature samples
- Si5351 (RF IC)
- LPC43XX (Micro)
- No FPGA



## BladeRF (FullDuplex)

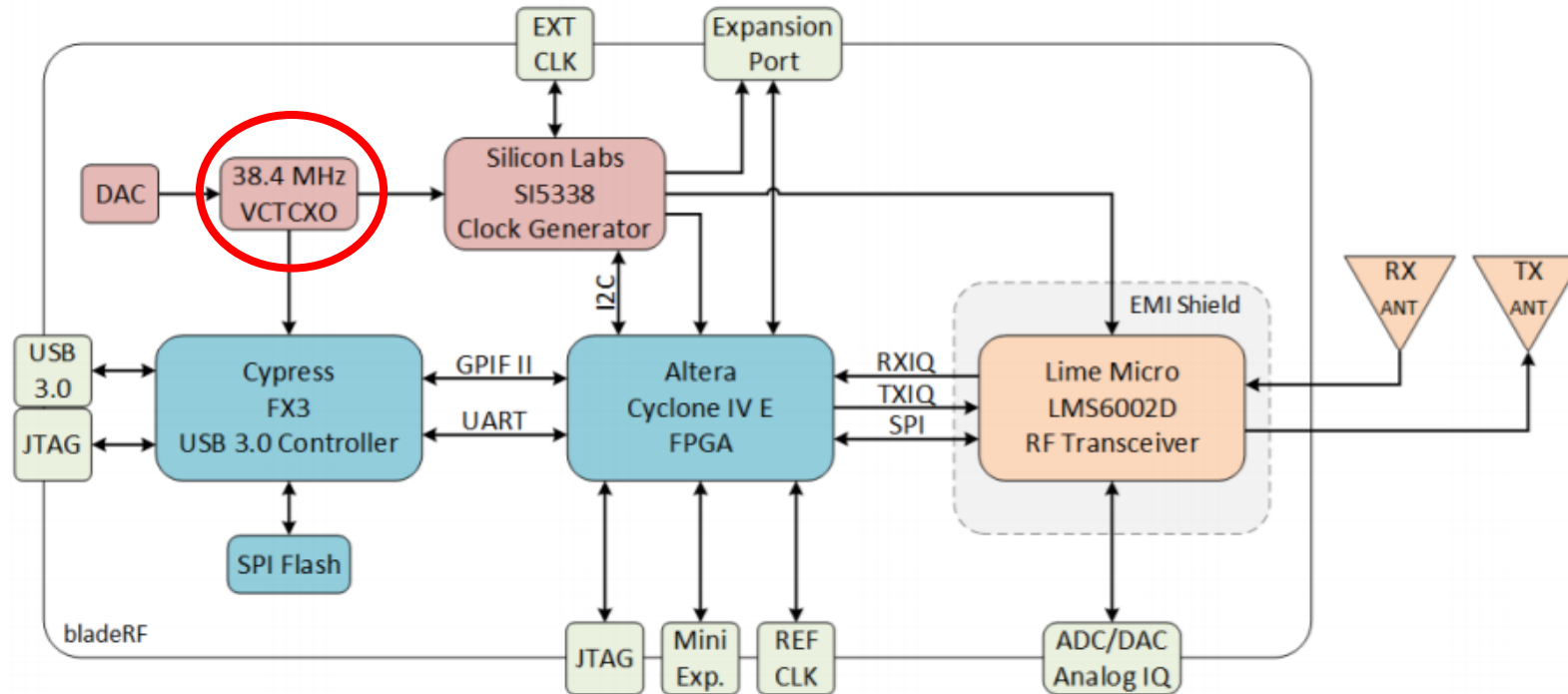
- Clockrate: 38,4Mhz
- 12-bit quadrature samples
- LMS6002D (RF IC)
- Cypress FX3
- Cyclone 4 (FPGA)



# HackRF/BladeRF



*“Flexible  
clocking  
architecture for  
arbitrary  
sample rates”*



# HackRF/BladeRF



## Sampling Rate and Baseband Filters

Using a sampling rate of less than 8MHz is not recommended. Partly, this is because the MAX5864 (ADC/DAC chip) isn't specified to operate at less than 8MHz, and therefore, no promises are made by Maxim about how it performs. But more importantly, the baseband filter in the MAX2837 has a minimum bandwidth of 1.75MHz. It can't provide enough filtering at 2MHz sampling rate to remove substantial signal energy in adjacent spectrum (more than +/-1MHz from the tuned frequency). The MAX2837 datasheet suggests that at +/-1MHz, the filter provides only 4dB attenuation, and at +/-2MHz (where a signal would alias right into the center of your 2MHz spectrum), it attenuates about 33dB. That's significant. Here's a picture:



Fuente: <https://github.com/mossmann/hackrf/wiki/Tips-and-Tricks>

## Clocking

mossmann edited this page on 19 Sep 2012 · 10 revisions

HackRF clock signals are generated by the Si5351. The plan so far:

- crystal frequency: 25 MHz (supports 25 or 27 MHz)
- optional clock input frequency: 10 MHz recommended (supports 10 to 40 MHz, or higher with division)
- VCO frequency: 800 MHz (supports 600 to 900 MHz)
- MAX2837 clock: 40 MHz
- preferred MAX5864 clocks: 8, 10, 12.5, 16, 20 MHz
- A clock at double the MAX5864 rate will be delivered to the CPLD and SGPIO.

Fuente:

<https://github.com/mossmann/hackrf/wiki/Clocking>



# Bodgan Airprobe ▪ HackRF/BladeRF

## USRP N210

- Clockrate: 100Mhz
- 16 bit samples
- SampleRate<sub>1</sub>: 25Mhz
- Decimacion<sub>1</sub> = 4
- SampleRate<sub>2</sub>: 10Mhz
- Decimacion<sub>2</sub> = 10
- SampleRate<sub>3</sub>: 574712
- Decimacion<sub>3</sub> = 174

## BladeRF

- Clockrate: 38,4Mhz
- 12 bit samples
- SampleRate<sub>1</sub>: 19,2Mhz
- Decimacion<sub>1</sub> = 2
- SampleRate<sub>2</sub>: 9,6Mhz
- Decimacion<sub>2</sub> = 4
- SampleRate<sub>3</sub>: 1,2Mhz
- Decimacion<sub>3</sub> = 32

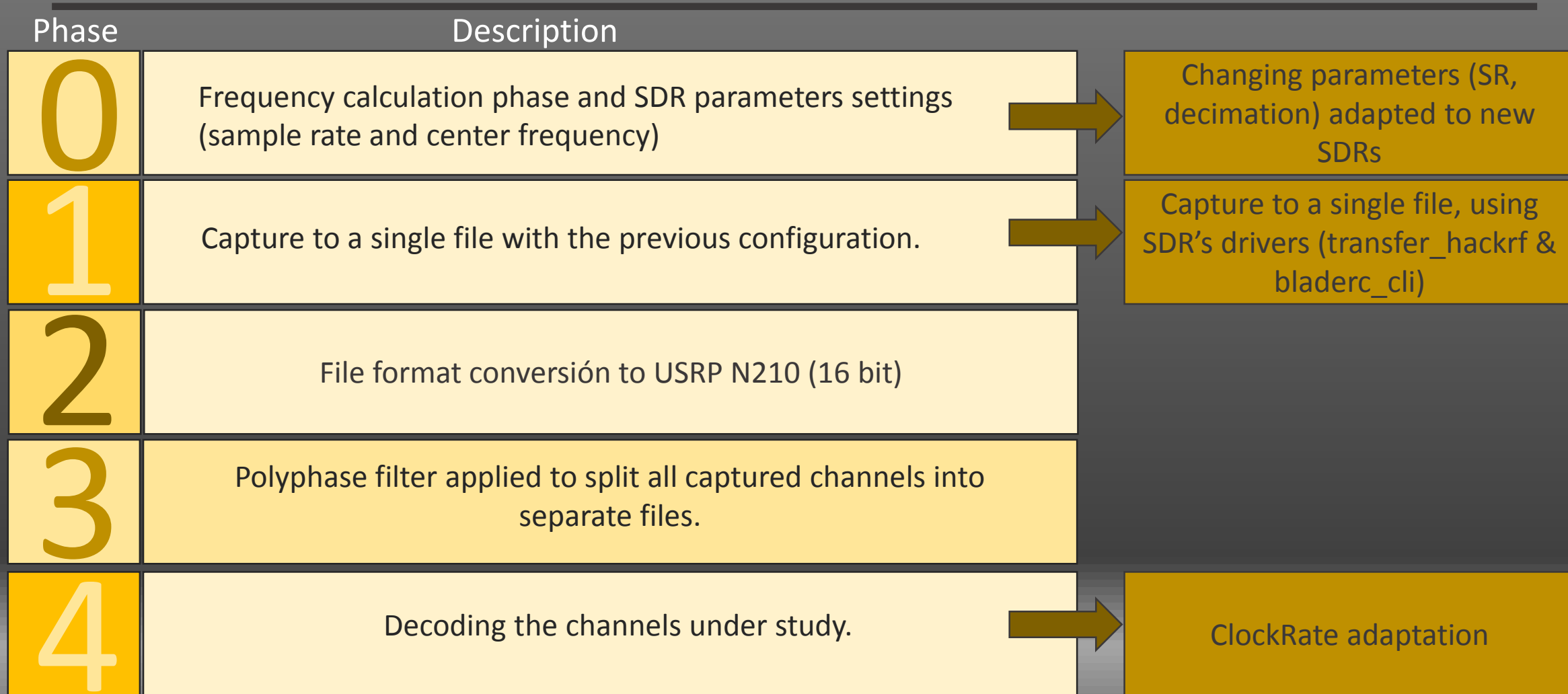
## HackRF

- Clockrate: 40Mhz
- 8 bit samples
- SampleRate<sub>1</sub>: 20Mhz
- Decimacion<sub>1</sub> = 2
- SampleRate<sub>2</sub>: 10Mhz
- Decimacion<sub>2</sub> = 4
- SampleRate<sub>3</sub>: 2,5 Mhz
- Decimacion<sub>3</sub> = 16

**GSM Symb Rate: 1625000/6**



# Bodgan Airprobe ▪ HackRF/BladeRF



# Bodgan Airprobe ▪ HackRF/BladeRF

Band Width	USRP N210	BladeRF	HackRF
BW > 10 M	<i>SR = 25 M sps</i>	<i>SR = 19,2 M sps</i>	<i>SR = 20 M sps</i>
	<i>N channels = 125</i>	<i>N channels = 96</i>	<i>N channels = 100</i>
	<i>pfbDECIM = 46</i>	<i>pfbDECIM = 16</i>	<i>pfbDECIM = 16</i>
	<i>totDECIM = 184</i>	<i>totDECIM = 32</i>	<i>totDECIM = 32</i>
10 M => BW > 200 k	<i>SR = 10 M Sps</i>	<i>SR = 9,6 M Sps</i>	<i>SR = 10 M Sps</i>
	<i>N channels = 50</i>	<i>N channels = 48</i>	<i>N channels = 50</i>
	<i>pfbDECIM = 17</i>	<i>pfbDECIM = 16</i>	<i>pfbDECIM = 16</i>
	<i>totDECIM = 170</i>	<i>totDECIM = 64</i>	<i>totDECIM = 64</i>
BW <= 200 k	<i>SR = 574712</i>	<i>SR = 1,2 M sps</i>	<i>SR = 2,5 M sps</i>
	<i>N channels = 1</i>	<i>N channels = 1</i>	<i>N channels = 1</i>
	<i>pfbDECIM = 1</i>	<i>pfbDECIM = 1</i>	<i>pfbDECIM = 1</i>
	<i>totDECIM = 174</i>	<i>totDECIM = 32</i>	<i>totDECIM = 16</i>

Changing parameters adapted to new SDRs





# Bodgan Airprobe ■ HackRF/BladeRF

0	Fase de cálculo de frecuencias y configuración de parámetros SDR (sample rate y frecuencia central)
1	Captura a un único fichero con la configuración anterior.
2	Conversión de ficheros a formato de USRP N210 (16 bit)
3	Aplicación de un filtro posifásico para separar todos los canales capturados en ficheros independientes.
4	Decodificación de los canales bajo estudio.

```
minARFCN=`echo $CA | awk '{print $1}'`  
maxARFCN=`echo $CA | awk '{print $NF}'`
```

```
ARFCN_fc=$((($maxARFCN+$minARFCN)/2))
```

```
if [ $ARFCN_fc -gt 125 ] && [ $ARFCN_fc -lt 975 ]  
then  
    FC=$((1805200000 + 200000*($ARFCN_fc-512)))  
elif [ $ARFCN_fc -lt 125 ]  
then  
    FC=$((935000000 + 200000*$ARFCN_fc))  
else  
    FC=$((935000000 + 200000*($ARFCN_fc-1024)))  
Fi
```

+ E-GSM

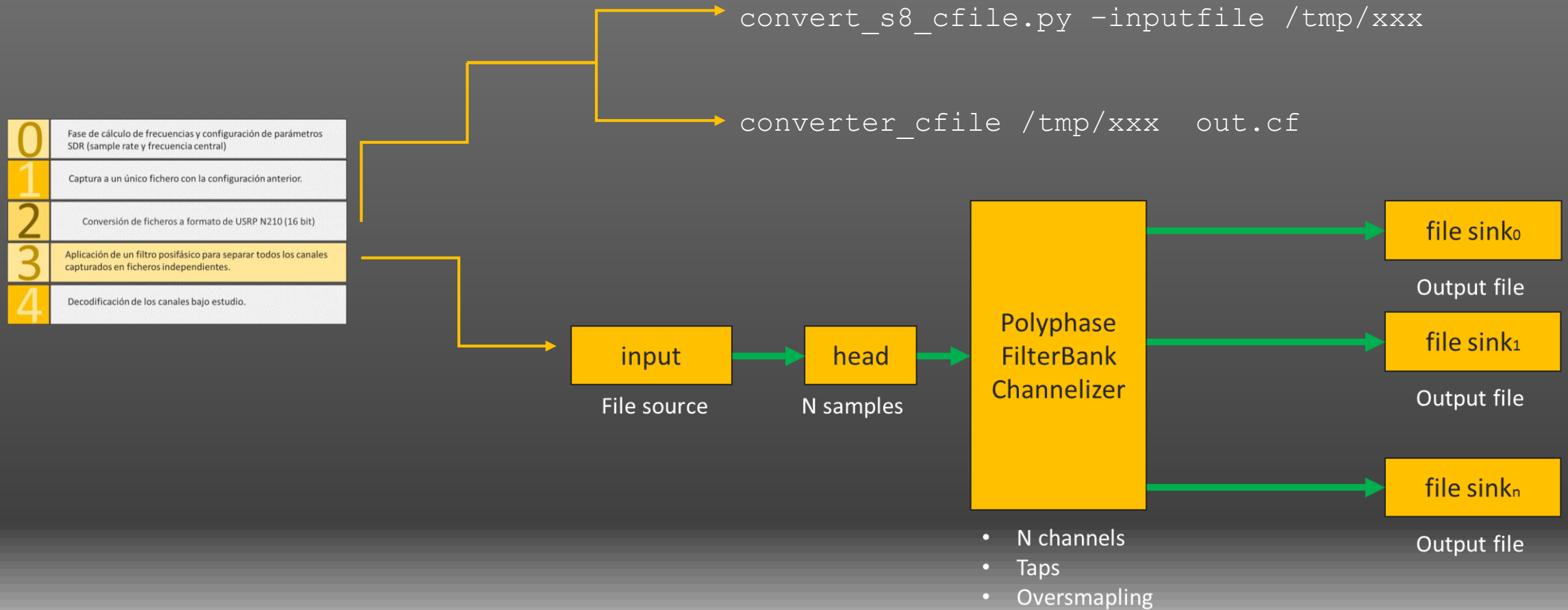
```
BW=$((($maxARFCN-$minARFCN+1)*200))
```

```
hackrf_transfer -f $FC -s $SR -n $NSAMPLES -l 16 -g 14 -x 16 -r /tmp/xxx
```

```
bladeRF-cli -e "set frequency rx $FC" -e "set samplerate rx $SR" -e 'set bandwidth  
rx 10M' -e 'set lnagain 6' -e 'set rxvga1 15' -e 'set rxvga2 3' -e "rx config  
file=/tmp/xxx"
```



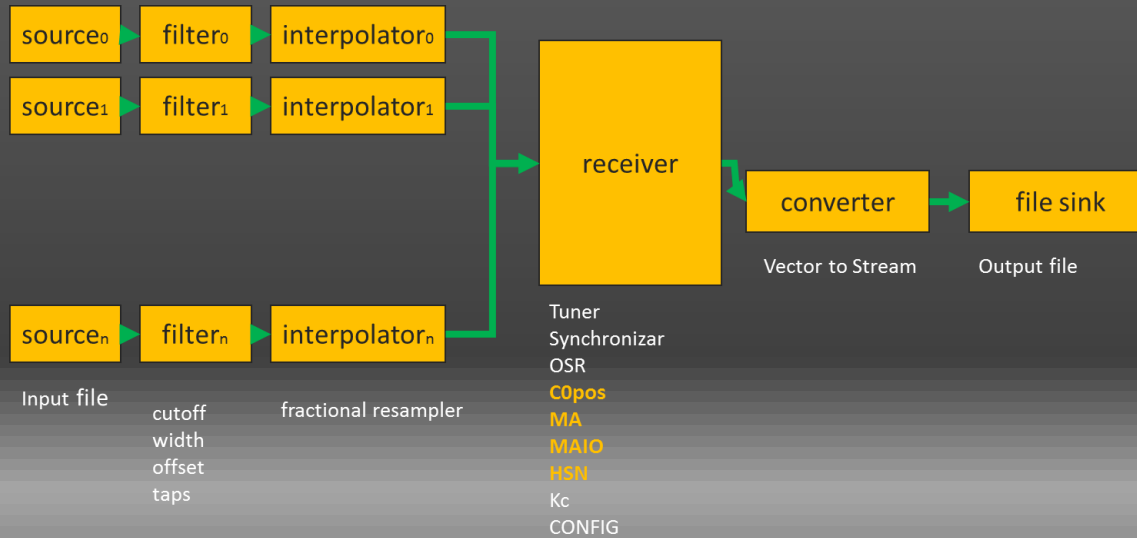
# Bodgan Airprobe ▪ HackRF/BladeRF



# Bodgan Airprobe ■ HackRF/BladeRF

0	Fase de cálculo de frecuencias y configuración de parámetros SDR (sample rate y frecuencia central)
1	Captura a un único fichero con la configuración anterior.
2	Conversión de ficheros a formato de USRP N210 (16 bit)
3	Aplicación de un filtro posifásico para separar todos los canales capturados en ficheros independientes.
4	Decodificación de los canales bajo estudio.

gsm\_receive\_channelize2.py



gsm\_receiveHackRF40\_channelize2.py

```
95 def _set_rates(self):  
96     options = self.options  
97     clock_rate = 40e6  
98     self.clock_rate = clock_rate
```

gsm\_receiveBladeRF38.4\_channelize2.py

```
95 def _set_rates(self):  
96     options = self.options  
97     clock_rate = 38.4e6  
98     self.clock_rate = clock_rate
```



# Bodgan Airprobe ■ HackRF/BladeRF

1	Captura a un único fichero con la configuración anterior.
2	Conversión de ficheros a formato de USRP N210 (16 bit)
3	Aplicación de un filtro posfásico para separar todos los canales capturados en ficheros independientes.
4	Decodificación de los canales bajo estudio.

## Conversion formats to cfile

Cfile format its a binary file containing 64-bit complex numbers. Each 64 bit complex number is two 32-bit floating point numbers corresponding to the real and imaginary parts of a complex number: *“A floating point data stream is saved as 32 bits in the file, one after the other. A complex signal has 32 bits for the real part and 32 bits for the imaginary part. Reading back a complex number means reading in 32 bits, saving that to the real part of a complex data structure, and then reading in the next 32 bits as the imaginary part of the data structure.”*

Adaptation to HackRF captured file:

Convert the file from unsigned 8-bit integers to 32-bit floats.

Adaptation to BladeRF captured file:

Convert from little-endian 16-bit to 32 bit floats.



# Bodgan Airprobe ■ HackRF/BladeRF



Class	Minimum Speed
2	2MB/s
4	4MB/s
6	6MB/s
8	8MB/s
10	10MB/s

UHS Class	Minimum Speed
1	10MB/s
3	30MB/s

	SD-class10	SD-UHS1	eMMC 5.0
Write speed (MB/s)	8.5	10.8	39.3
Read speed (MB/s)	18.9	35.9	140

HackRF		BladeRF		
8	Bit	16	Bit (SC16Q11)	<i>File Resolution</i>
256.000.000	Samples	256.000.000	Samples	<i>Sample size</i>
10.000.000	Samples/seg	9.600.000	Samples/seg	<i>SDR Sample Rate</i>
20.000.000	Bytes/seg	38.400.000	Bytes/seg	<i>Transfer to Host (PC)</i>
20	MB/seg	38,4	MB/seg	
25,60	Seconds	26,67	Seconds	<i>Capture duration</i>
512	MB	1024	MB	<i>Capture size</i>

HackRF		BladeRF		
8	Bit	16	Bit (SC16Q11)	<i>File Resolution</i>
32.000.000	Samples	32.000.000	Samples	<i>Sample size</i>
10.000.000	Samples/seg	9.600.000	Samples/seg	<i>SDR Sample Rate</i>
20.000.000	Bytes/seg	38.400.000	Bytes/seg	<i>Transfer to Host (PC)</i>
20	MB/seg	38,4	MB/seg	
3,20	Seconds	3,33	Seconds	<i>Capture duration</i>
64	MB	128	MB	<i>Capture size</i>



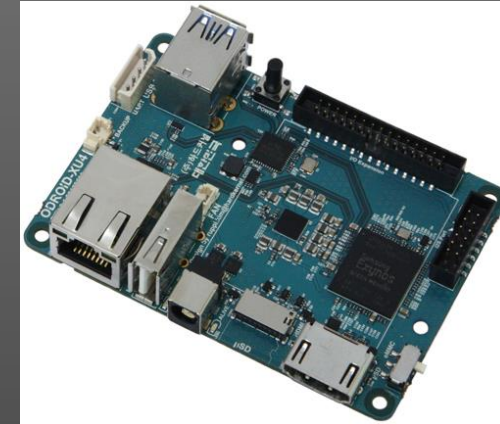
# HackRF @ Android & ARM

Dennis Mantz ported libhackrf to Android ...

Device	Does it work?	Comments	Tester
Oneplus One	yes		KR0SIV
Nexus 7 2012	yes	~ 2 Msps, Filewriter is too slow.	demantz
Nexus 7 2013	yes	15 Msps	@kx3companion
Nexus 4	needs ROM...	...and Y cable. ( <a href="http://goo.gl/rRyQVM">http://goo.gl/rRyQVM</a> )	-
Nexus 5	yes	15 Msps	demantz
Moto G	yes	~ 2 Msps	@kx3companion
Acer A500	yes	~ 5 Msps	@digital
Samsung S3 LTE	yes	10 Msps, running CM 10.1.3	dc1rdb
Samsung S4	yes		@digital
Samsung S4 LTE	yes	10 Msps	Jonyweb



ODROID-XU4



- \* Samsung Exynos5422 Cortex™-A15 2Ghz and Cortex™-A7 Octa core CPUs
- \* 2Gbyte LPDDR3 RAM PoP stacked
- \* eMMC5.0 HS400 Flash Storage
- \* 2 x USB 3.0 Host, 1 x USB 2.0 Host



- **Software Defined Radio support.** Use **Kali Nethunter** with your HackRF to explore the wireless radio space.



# The END

---

Source code:

<https://github.com/pcabreracamara/bodgan-airprobe-hackrf-bladerf>

