Decoding GPS signals using ADALM-PLUTO within the OscimpDigital Ecosystem

G. Goavec-Merou, J.-M Friedt
{gwenhael.goavec,jmfriedt}@femto-st.fr
June 16, 2019

This tutorial is intended to demonstrate the use of the OSCIMP framework to generate a bitstream in the ADALM-PLUTO programmable logic part of the Zynq CPU to decode GPS [1] signals. It focuses on

- Introducing basics of GPS and CDMA communication
- Introducing basics of Pseudo Random Number (PRN) generator and demonstrating some CDMA basics
- Introducing means of correcting carrier frequency differences between emitter and receiver and how to compensate for such differences in a context of binary phase shift keying (phase modulation)
- Demonstrating GPS decoding by running all identifiers over all possible Doppler frequency shifts

The new processing blocks introduced in this tutorial are the 7-bit PRN generator, GPS Gold Code PRN generator, complex mixer and correlator.

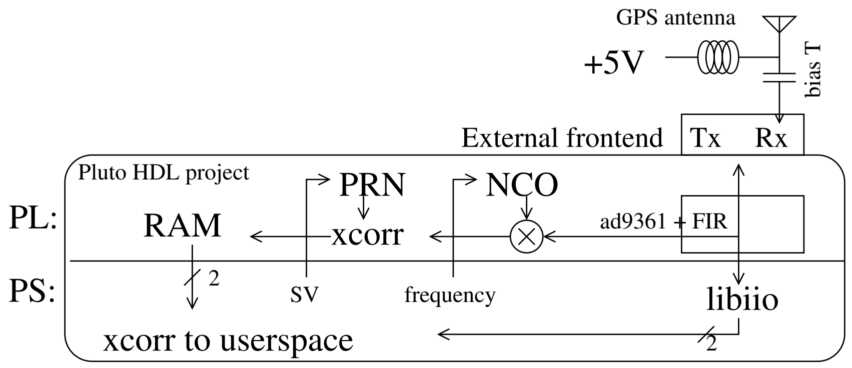


Figure 1: Objectives of this tutorial and raw schematics of the processing chains. Decoding GPS requires being able to configure from the PS the Space Vehicle (SV) number, defining which PRN Gold Code is generated, as well as the frequency offset to compensate for Doppler shift induced by satellite motion along its orbit.

The satellites of the GPS constellation all emit on the same carrier frequency of 1575.42 MHz. The way the signal coming from each satellite is separated is by cross-correlating with each unique satellite identifier, as classically done in Code Division Multiple Access. Correlating is a computationally intensive task either efficiently implemented through a Fast Fourier Transform (converting a N^2 complexity algorithm to a $N\ln(N)$ complexity task) or parallelized in an FPGA.

Indeed, remembering that the correlation $xcorr()$ between x and y is expressed as

$$xcorr(x,y)(\tau) = \int x(t)y(t+\tau)dt \underbrace{\longrightarrow}_{discrete} xcorr(x,y)_n = \sum_k x_n y_{n+k}$$

and based on the convolution theorem, then

$$FFT(xcorr(x,y)) = FFT(x)\cdot FFT^*(y) \Leftrightarrow xcorr(x,y) = iFFT\left(FFT(x)\cdot FFT^*(y)\right)$$

where * is the complex conjugate.

In this tutorial, we wish to demonstrate how the PlutoSDR firmware can be updated to include the Pseudo Random Number (PRN) generator identifying each satellite, how Doppler shift can be compensated for by programming a numerically controlled oscillator without changing the radiofrequency receiver local oscillator on-board the AD9363, and how cross-correlation is computed on the programmable logic to only provide the processed data to the processing system for display.

Rather than addressing immediately the complete GPS signal processing chain, we consider each task sequentially by first demonstrating cross-correlation with a more simple 7-bit (127-long sequence) PRN, show that the correlator is working properly on this simple signal when the emitter and receiver carrier are at the same frequency (demonstration of code decoding and code orthogonality). Since GPS signals are emitted by satellites moving around the Earth at an altitude of 20000 km inducing a Doppler shift of ±5 kHz at 1575.42 MHz in addition to the local oscillator offset, the emitter and receiver carrier frequency offset compensation is demonstrated by adding in the programmable logic of the Zynq a numerically controlled oscillator and a mixer. Having demonstrated these two basic steps, the 7-bit PRN generator is replaced with the GPS 10-bit Gold Code generator and the cross-correlator block extended to process 1023-bit long sequences. The final result is a set of PRN-Doppler maps in which strong cross-correlation demonstrate GPS signal decoding.

1 7-bit PRN on the same receive and transmit carrier frequencies

The first example aims at demonstrating the generation of an 7-bit long PRN sequence on the Zynq Programmable Logic (PL) and correlating a transmitted signal modulated with such a sequence. The PlutoSDR is fed by a signal generated from GNU Radio phase-modulating the carrier by a GLFSR block configured to generate a known sequence. First of all, we become convinced of the accuracy of computing the cross-correlation as the inverse Fourier transform of the product of two Fourier transforms. Notice (Fig. 2, left) that in order to have 0-delay in the middle of the time-domain chart and the negative delay to the left, we first compute two *inverse* Fourier transforms and the take the *direct* Fourier transform of their product. Fig. 2 (right) indeed exhibits correlation peaks separated by the PRN repetition length – 127 for an 7-bit PRN – with a weighing dependent on the windowing filter.

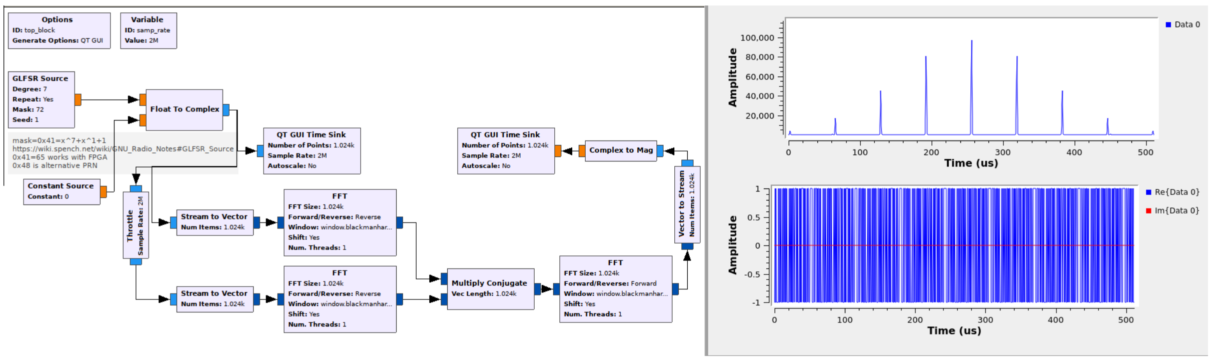
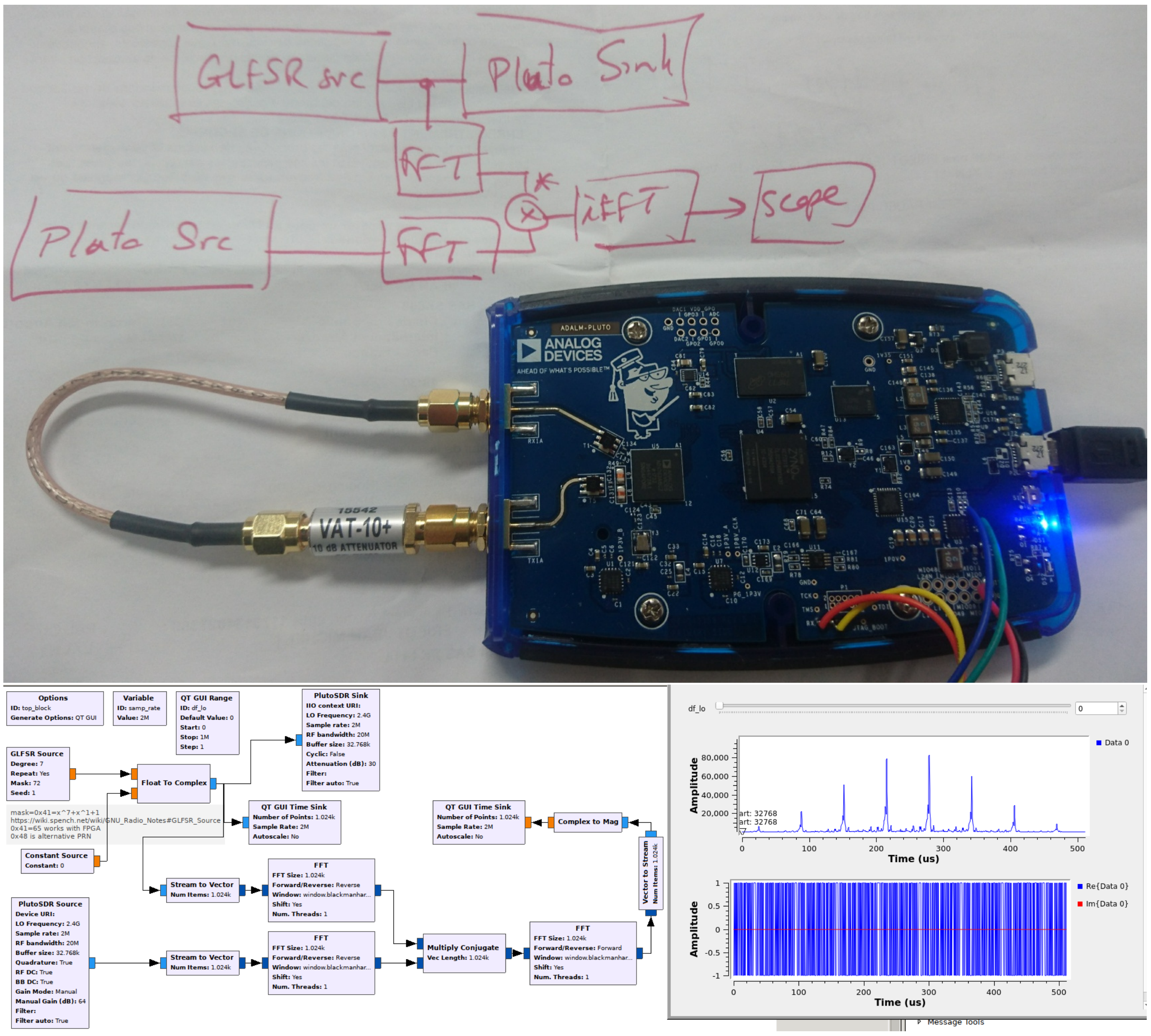


Figure 2: Synthetic signal cross correlation demonstration

The synthetic data example (Fig. 2) is expanded by running on the PlutoSDR transceiver (Fig. 3). In that case, the BPSK modulation is created by running the output of the GLFSR block, with values either +1 or -1 with the pseudo-random sequence, as input of the PlutoSDR sink. Since $+1 = \exp(j0)$ et $-1 = \exp(j\pi)$, indeed the resulting carrier modulation is a constant amplitude and phase shifting from 0 to π, as expected from BPSK. The PlutoSDR Source is then run through the same correlation scheme fed on the other side by the phase sequence generated by the GLFSR. The PlutoSDR output is connected to the PlutoSDR input through an optional 10 dB attenuator to reduce risks of damaging the radiofrequency input frontend (Fig. 3, top).

Again the correlation pattern (Fig. 3, right) repeated every 127 sampling period is well visible. All these signal processing sequences have been run on the host personal computer running GNU Radio with



Options
ID: top_block
Generate Options: QT GUI

Variable
ID: samp_rate
Value: 2M

QT GUI Range
ID: df_lo
Default Value: 0
Start: 0
Stop: 1M
Step: 1

PlutoSDR Sink
IIO context URI:
LO Frequency: 2.4G
Sample rate: 2M
RF bandwidth: 20M
Buffer size: 32.768k
Cyclic: False
Attenuation (dB): 30
Filter:
Filter auto: True

GLFSR Source
Degree: 7
Repeat: Yes
Mask: 72
Seed: 1

Float To Complex

mask=0x41=x^7+x^1+1
https://wiki.spench.net/wiki/GNU_Radio_Notes#GLFSR_Source
0x41=65 works with FPGA
0x48 is alternative PRN

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 2M
Autoscale: No

Constant Source
Constant: 0

Stream to Vector
Num Items: 1.024k

FFT
FFT Size: 1.024k
Forward/Reverse: Reverse
Window: window.blackmanhar...
Shift: Yes
Num. Threads: 1

PlutoSDR Source
Device URI:
LO Frequency: 2.4G
Sample rate: 2M
RF bandwidth: 20M
Buffer size: 32.768k
Quadrature: True
RF DC: True
BB DC: True
Gain Mode: Manual
Manual Gain (dB): 64
Filter:
Filter auto: True

Stream to Vector
Num Items: 1.024k

FFT
FFT Size: 1.024k
Forward/Reverse: Reverse
Window: window.blackmanhar...
Shift: Yes
Num. Threads: 1

Multiply Conjugate
Vec Length: 1.024k

FFT
FFT Size: 1.024k
Forward/Reverse: Forward
Window: window.blackmanhar...
Shift: Yes
Num. Threads: 1

Vector to Stream
Num Items: 1.024k

Complex to Mag

QT GUI Time Sink
Number of Points: 1.024k
Sample Rate: 2M
Autoscale: No

Figure 3: Top: initial idea conception and experimental setup: the 10 dB attenuator is added to try and protect the receiver from excessive emitted power but might be omitted. Bottom: GNU Radio flowchart running the LFSR as a phase modulation of the PlutoSDR carrier, recording the signal through the receiver and cross-correlating with the emitted pattern to demonstrate the concept of CDMA.

the PlutoSDR fitted with the original bitstream. We now wish to run the correlation sequence on the Zynq PL and only recover the cross-correlation output through the DataToRam interface.

The Zynq PL must be appended with two blocks: the PRN generator and the correlator. The PRN generator is implemented as a shift register with tap positions defined by the polynomial non-zero coefficients. The feedback is through XOR gates feeding the register input, while the PRN is observed at the output of the register.

The correlation is implemented in the time-domain which is well suited for massive parallelization on the PL.

Remember for synthesizing the project for the PlutoSDR to export the `ADI_HDL_DIR` variable towards the `plutosdr-fw/hdl/` directory. The `design` directory hold the application demonstration, which is compiled thanks to the provided `Makefile` assuming the Vivado and OscImpDigital variables have been set properly. Because the datastream in the PlutoSDR is clocked at `samp_rate` (as defined by the GNU Radio flowgraph) and so is the GLFSR datastream provided by GNU Radio Companion, the datarate will be consistent during the correlation process. Hence, the VHDL implementation of the PRN is (`fpga_ip/prn/hdl`):

```
--http://users.ece.cmu.edu/~koopman/lfsr/7.txt
xorM_N <= lfsr_s(0) xor lfsr_s(6);
lfsr_next_s <= lfsr_s(BIT_LEN-2 downto 0) & xorM_N;
```

which is the straigtforward implementation of the XORed shift-register taps. The PRN generator block can be tuned for various PRN lengths (`BIT_LEN`) and for various sequences for this given length (`PRN_NUM`). Hence, in a CDMA demonstration approach, we can generate two PRNs and correlate the datastream generated by GNU Radio with both sequences simultaneously: only the matching pattern will accumulate energy in the correlation peaks and hence demonstrate the identification process thanks to the PRN orthogonality. The cross-correlation block must be tuned with matching sequence length: the `xcorr_prn_slow_complex/hdl` holds the implementation which states that `LENGTH` is set to the PRN sequence length.

The cross-correlation block parametrization requires defining how many correlations are run in parallel. Due to the limited resources of the Zynq 7010 fitted on the PlutoSDR, we try to re-use the same processing blocks as many times as possible. If the sequential processing of the input stream becomes too long with respect ot the input stream, then parallel branches must be used. Thus, the definition of `NB_BLK` is as follows:

1. let us assume an input stream at 1 Msamples/s, and the internal clock rate of the PlutoSDR of 100 MHz [1]. Then a single cross-correlation `xcorr_prn_slow_complex` can process 100 correlations before a new input bit is clocked by the datastream
2. one last processing block is always assumed to be added to process the remaining number of samples. In this case, 27 more correlations will be needed, which can be performed by this second implicit processing block (since $27 < 100$)
3. if the clock had only been 50 MHz, then 2 blocks would have been needed: the two blocks would have processed $2 \times 50 = 100$ correlations sequentially, and the remaining 27 could have been taken care of by the additional implicit block since $27 < 50$.

Hence, the TCL file defining the PlutoSDR firmware is appended (`design/system_bd.tcl`) with

```
# prn1
ad_ip_instance prn prn1_gen
ad_ip_parameter prn1_gen CONFIG.PERIOD_LEN 1
ad_ip_parameter prn1_gen CONFIG.BIT_LEN 7
ad_ip_parameter prn1_gen CONFIG.PRN_NUM 1

ad_connect sys_ps7/FCLK_CLK0 prn1_gen/clk
ad_connect sys_rstgen/peripheral_reset prn1_gen/reset

# xcorr1
ad_ip_instance xcorr_prn_slow_complex xcorr1
ad_ip_parameter xcorr1 CONFIG.LENGTH 127
ad_ip_parameter xcorr1 CONFIG.NB_BLK 1
ad_ip_parameter xcorr1 CONFIG.IN_SIZE 16
ad_ip_parameter xcorr1 CONFIG.OUT_SIZE 32

ad_connect duppl_xcorr/data1_out xcorr1/data_in
ad_connect prn1_gen/prn_o xcorr1/prn_i
ad_connect xcorr1/prn_sync_o prn1_gen/tick_i

#dataComplex
ad_ip_instance dataComplex_to_ram data_to_ram
ad_ip_parameter data_to_ram CONFIG.NB_INPUT 1
ad_ip_parameter data_to_ram CONFIG.DATA_SIZE 32
ad_ip_parameter data_to_ram CONFIG.NB_SAMPLE 2048

ad_connect xcorr1/data_out data_to_ram/data1_in
ad_connect sys_rstgen/peripheral_reset data_to_ram/s00_axi_reset

ad_cpu_interconnect 0x43C20000 data_to_ram
```

[1] FCLK_CLK0 of sys_ps7 block is defined as shown by double clicking on the block and selecting Clock Configuration → PL Fabric Clock → FCLK_CLK0=100 MHz. This information is initially defined in the TCL script `system_bd.tcl` as `ad_ip_parameter sys_ps7 CONFIG.PCW_FPGA0_PERIPHERAL_FREQMHZ 100.0`

defining the first (PRN_NUM) sequence of PRN with 127-length (BIT_LEN) clocked by the internal 100 MHz clock (sys_ps7/FCLK_CLK0) and the cross correlation operates on these 127-bits (LENGTH) with a single processing block (NB_BLK) considering than an implicit second block will take care of the remaining 27 cycles.

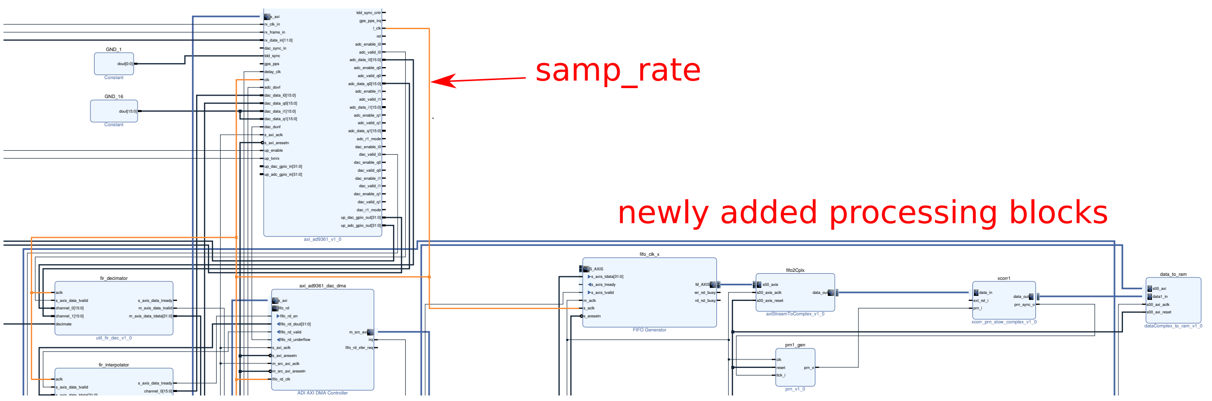


Figure 4: PRN generator and cross-correlation with the received datastream.

We see on this TCL script – or in Vivado by double-clicking on the Data_to_RAM block, that the data size will be encoded as 32-bit words. Since we are handling complex data, we fetch two 32-bit words for each sample, and a total of 2048 samples are stored in the RAM. Hence, the userspace application for fetching the data will be

```
int main()
{int32_t c[2048 * 2 *NUM_PRN]; /* two input (x2) x 2048 sample complex (x2) */
 int fi, fo;
 fi = open("/dev/data3200", O_RDWR);
 fo = open("/tmp/data.bin", O_WRONLY | O_CREAT);
 read(fi, c, 2048 * 2 * NUM_PRN * sizeof(int32_t));
 write(fo, c, 2048 * 2 * NUM_PRN * sizeof(int32_t));
 close(fi);
 close(fo);
}
```

and the resulting /tmp/data.bin is transfered to the host PC for reading by GNU/Octave (Fig. 5) with `fd=fopen("data.bin"); a=fread(fd, Inf, "int32"); prn=a(1:2:end)+i*a(2:2:end); plot(abs(prn1));`.

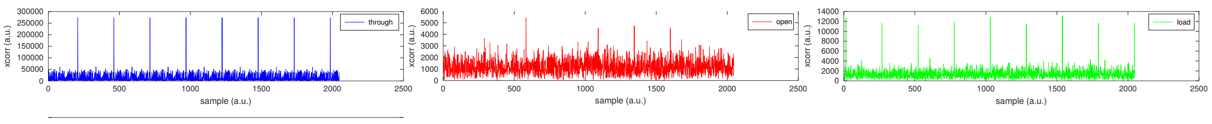


Figure 5: Correlation output from the PL processing, left to right with a through connexion between TX and RX, open circuit or loaded input and output (50 Ω).

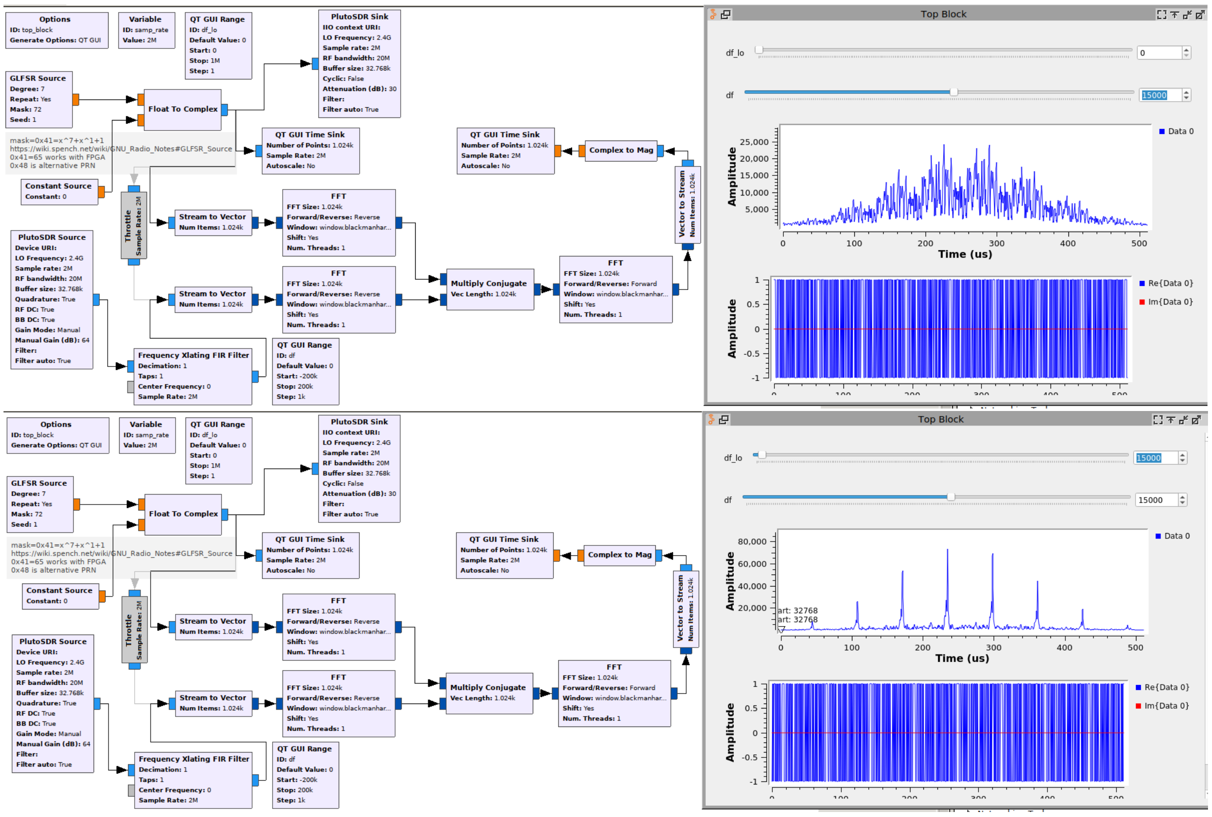
The last challenge is the different datarates between the PRN generation and the cross-correlator. Indeed the cross-correlator is clocked by the 100 MHz AXI bus clock while the input datastream is clocked at the samp_rate rate. Hence, a FIFO block must be included between the output datastream and the cross correlator input (Fig. 4).

Since the 7-bit PRN generator is designed to only generate one pseudo-random sequence out of the 18 possible options (http://users.ece.cmu.edu/~koopman/lfsr/7.txt), namely the taps defined as 0x41=65 (or $x^7 + x^6 + 1$), we can also use this opportunity to demonstrate the orthogonality of the codes needed for CDMA. Indeed cross correlating to null-mean value pseudo-random sequence will not allow for coherent energy accumulation and will not yield discrete correlation peaks when the two sequences match.

Figure 6: Running the same PRN sequence on the Zynq PL (receiver side) than the one emitted by GNU Radio allows for identifying cross-correlation peaks separated by the PRN length of 127 samples. Emitting a different PRN sequence than the one run on the Zynq PL yield close-to-zero cross correlations, hence demonstrating the orthogonality of PRN codes.

2 7-bit PRN on different receive and transmit carrier frequencies

We are convinced the correlator block is operational and fits in the PL remaining resources (although decoding 1023-bit long GPS Gold Codes will require extending the correlator length from 127 to 1023 bits, hence using more resources), and that we understand how to generate PRN sequences matching incoming patterns.



Options ID: top_block; Generate Options: QT GUI
Variable ID: samp_rate; Value: 2M
QT GUI Range ID: df_lo; Default Value: 0; Start: 0; Stop: 1M; Step: 1
PlutoSDR Sink IIO context URI:; LO Frequency: 2.4G; Sample rate: 2M; RF bandwidth: 20M; Buffer size: 32.768k; Cyclic: False; Attenuation (dB): 30; Filter:; Filter auto: True
GLFSR Source Degree: 7; Repeat: Yes; Mask: 72; Seed: 1
Float To Complex
```
mask=0x41=x^7+x^1+1
https://wiki.spench.net/wiki/GNU_Radio_Notes#GLFSR_Source
0x41=65 works with FPGA
0x48 is alternative PRN
```
Constant Source Constant: 0
Throttle Sample Rate: 2M
QT GUI Time Sink Number of Points: 1.024k; Sample Rate: 2M; Autoscale: No
QT GUI Time Sink Number of Points: 1.024k; Sample Rate: 2M; Autoscale: No
Complex to Mag
Vector to Stream Num Items: 1.024k
Stream to Vector Num Items: 1.024k
FFT FFT Size: 1.024k; Forward/Reverse: Reverse; Window: window.blackmanhar...; Shift: Yes; Num. Threads: 1
PlutoSDR Source Device URI:; LO Frequency: 2.4G; Sample rate: 2M; RF bandwidth: 20M; Buffer size: 32.768k; Quadrature: True; RF DC: True; BB DC: True; Gain Mode: Manual; Manual Gain (dB): 64; Filter:; Filter auto: True
Stream to Vector Num Items: 1.024k
FFT FFT Size: 1.024k; Forward/Reverse: Reverse; Window: window.blackmanhar...; Shift: Yes; Num. Threads: 1
Multiply Conjugate Vec Length: 1.024k
FFT FFT Size: 1.024k; Forward/Reverse: Forward; Window: window.blackmanhar...; Shift: Yes; Num. Threads: 1
Frequency Xlating FIR Filter Decimation: 1; Taps: 1; Center Frequency: 0; Sample Rate: 2M
QT GUI Range ID: df; Default Value: 0; Start: -200k; Stop: 200k; Step: 1k

Top: df_lo 0; df 15000
Bottom: df_lo 15000; df 15000

Figure 7: Offsetting the emitted frequency with respect to the received frequency (top) prevents the cross-correlation from accumulating energy due to the null mean power of the trigonometric functions. Compensating for the frequency offset solves the issue (bottom), as will be needed in the case of GPS reception due to the Doppler frequency shift associated with satellite motion.

However, GPS satellites are not geostationary but moving along their orbit at an altitude of 20000 km. Basic celestial mechanics tells us that the period is 12 hours, and dividing the orbit length by the period gives us the satellite velocity from which the Doppler shift is deduced. We expect Doppler shifts in the ± 5 kHz range, while the cross-correlation of the 1023-bit long code running at 1.023 Mb/s will be canceled for any frequency offset of more than 1 kHz (inverse of 1 ms repetition rate) due to the zero-mean value of trigonometric functions. Practically, good signal to noise ratio will be achieved if the receiver frequency is not offset by more than 500 Hz from the emitter frequency. Hence, Doppler shift compensation on the receiver side is needed: the **acquisition** phase of GPS reception means sweeping all possible Space Vehicle (SV$\in$ [1..32]) code and all possible Doppler frequency shifts to try and find which satellite is at what position. This step will be addressed in this document, while the following

tracking phase will not be addressed.

Back to basics, we wish to demonstrate on the 7-bit long PRN sequence how a frequency offset between emitter and receiver can be compensated for by including a PS-configurable Numerically Controlled Oscillator (NCO) and mixing the incoming signal prior to correlating. Again we first start demonstrating the concept using GNU Radio prior to extending the demonstration to the Zynq PL. Fig. 7 demonstrates using GNU Radio's Frequency Xlating FIR Filter to compensate for the frequency offset introduced on purpose between the PlutoSDR emitter and receiver. Fig. 7 indeed demonstrates that the correlation remains null for high frequency offsets and that the correlation peaks re-appear once the frequency offset has been compensated for.

The challenge of implementing the NCO in the FPGA is that we must know what the sampling frequency is, *i.e.* know the clock rate of the NCO block. We have identified that the frequency of the signal propagated in the FPGA is given by reading
`/sys/bus/iio/devices/iio device1/in_voltage_sampling_frequency`
which tells us that the GNU Radio `samp_frequency` is the clock signal frequency in the PL.

The PL implementation is quite similar except that this time the NCO must be included between the datastream – running at `samp_rate` – and the FIFO feeding the correlator (Fig. 8).

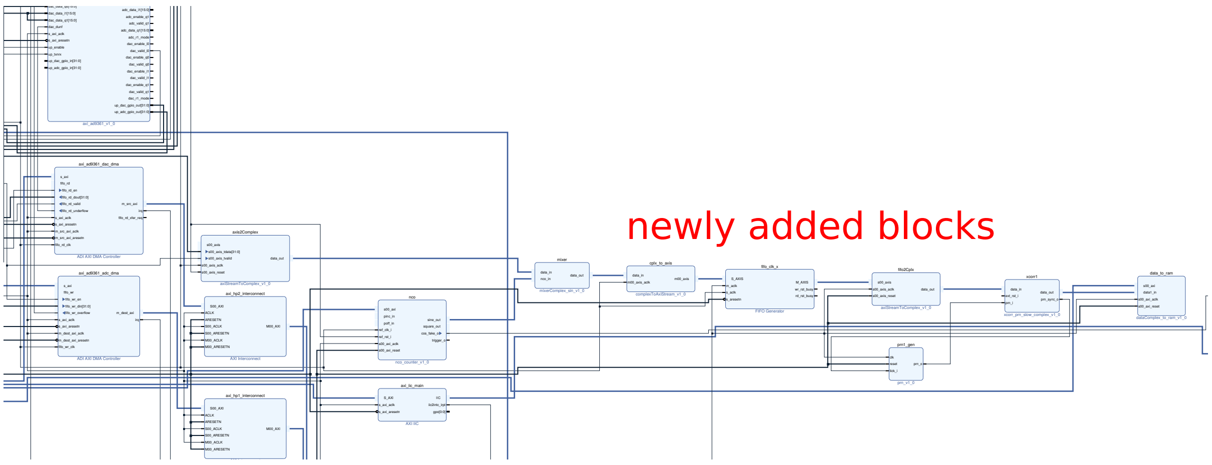


Figure 8: PRN generator and cross-correlation with the received datastream with frequency transposition using the NCO.

In order to demonstrate both frequency shifting capability and dual-PRN detection to select one emitter or the other, two PRN generators and two correlators can be included in the decoding path (Fig. 9).

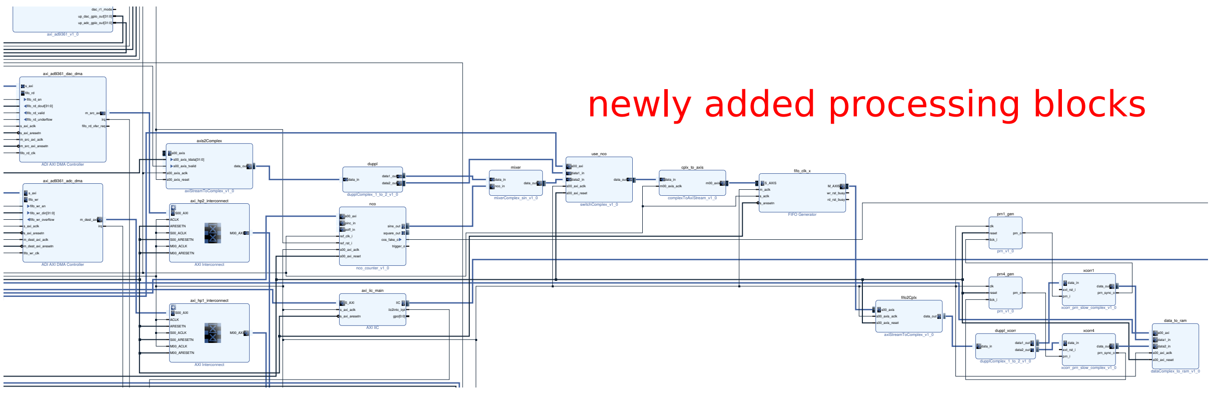


Figure 9: Dual PRN generator and cross-correlation with the received datastream frequency transposed using the NCO.

3 GPS signal reception

The knowledge acquired so far now allows us to tackle GPS acquisition. During this initialization phase in which the receiver does not know what its local oscillator offset is, which satellite is visible and where they are located in the sky, a brute force approach is needed to try all possible satellite identifiers with all possible Doppler shifts[2]. Hence, the NCO is programmed with narrow enough frequency steps to sweep all possible Doppler shifts, and all possible space vehicle PRNs are correlated with the frequency transposed I/Q signal. GPS does not use a basic 10-bit long PRN but the so-called Gold code provided in the `cacode` FPGA IP. Notice that this same function is available for Matlab or GNU/Octave at https://www.mathworks.com/matlabcentral/fileexchange/14670-gps-c-a-code-generator and will be used later in the software-decoding of GPS.

The first step is to assess the hardware setup and being able to collect I/Q streams using the PlutoSDR tuned to the GPS carrier frequency of 1575.42 MHz (Fig. 10). Since our NCO only accepts positive frequencies, we actually slightly offset the carrier frequency so that the Doppler correction is achieved only by programming positive frequencies when configuring the NCO.

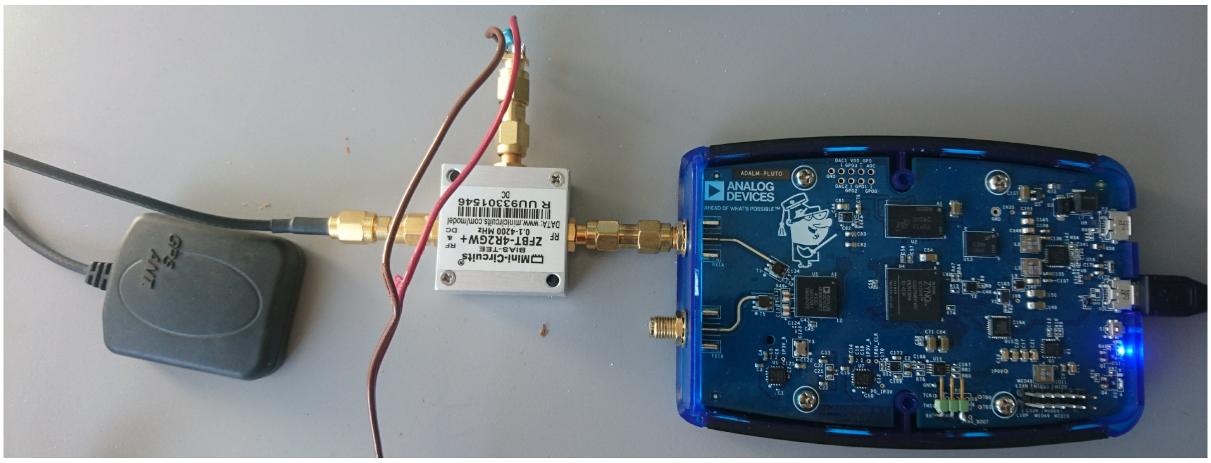


Figure 10: Experimental setup: a bias-T is introduced between the PlutoSDR RX input and the GPS antenna to power the antenna amplifier. A radiofrequency-grade 1 to 100 μH inductor connected to the +5 V power supply and a 1 nF capacitor to the receiver will provide similar results.

While `cacode` outputs all SV sequences, the space available on the Zynq-7010 of the PlutoSDR only allows for synthesizing a single cross-correlating block for 1023-bit long sequences. Hence, the Zynq PS is programmed to configure a single SV identifier, and then sweep NCO values. Since the 1023-bit long is run at 1.023 Mb/s on the GPS satellite, the code repeats every millisecond. To make sure we observe a fine correlation peak, we sweep the frequency with 300 Hz steps.

Running all SV identifiers and for each SV, running all frequency offsets in a ±100 kHz range (to account for possible local oscillator bias) requires 1 minute and 50 seconds for 100 Hz steps and 31 satellites (Fig. 11, bottom right) or 36 seconds for 300 Hz steps. Mapping the constellation of satellites with 500 Hz steps (Fig. 11, bottom left) is hence expected to require only 22 seconds when running on the computation on the PL. The same computation result (500 Hz steps, 31 satellites) requires 108 seconds on a 2.60 GHz i5-3320M CPU as found in the Panasonic CF-19 running GNU/Octave, while the displayed chart (100 Hz steps) required 504 seconds using the following program:

```
pkg load signal
x=read_complex_binary('./gps.bin');
fs=1.023; % MHz % sampling frequency
freq0=[-1.0e4:500:1.0e4]-20000; % frequency range
x=x(1:2e5); % data subset
time=[0:1/fs/1e6:length(x)/fs/1e6]';time=time(1:end-1); % discrete time
for m=[1:31] % PRN number
  a=cacode(m,fs/1.023); a=a-mean(a);
```

[2]the receiver can actually get a hint of the Doppler shifts to be investigated by squaring the BPSK signal and looking at the spectral components of the Fourier transform of the resulting signal. Since squaring a BPSK modulated signal gets rid of the modulation, strong spectral components will be visible at twice the offset frequency.

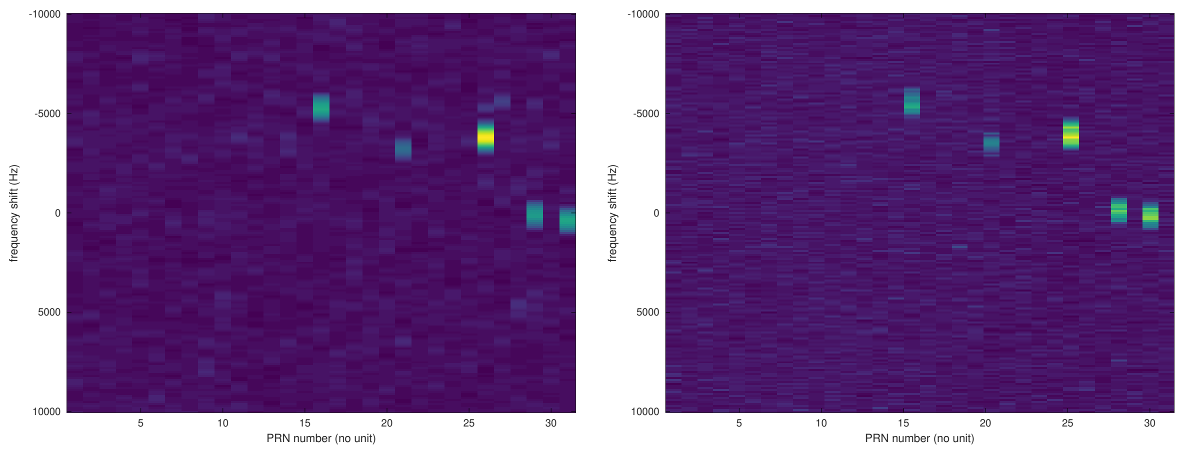


Figure 11: Left: cross-correlation (color code) maximum as a function of SV number (abscissa) and frequency offset (ordinate) as computed using GNU/Octave on a 250 ksamples record collected from the PlutoSDR. Right: same computation as performed on the Zynq PL. The results are consistent with space vehicles 16, 21, 26, 29 and 31 clearly visibles at frequency offsets within the ±5 kHz range expected from the Doppler shift. No local oscillator bias is seen on this PlutoSDR.

```
  l=1; m
  for freq=freq0 % run through possible frequency offsets
   mysine=exp(j*2*pi*(-freq)*time);
   xx=x.*mysine; % frequency shift the signal
   [u(l,m),v(l,m)]=max(abs(xcorr(a,xx))); % check for cross correlation max.
   l=l+1;
  end
end
imagesc([1:31],-freq0-20000,abs(u))
```

References

[1] T.F. Collins, R. Getz, D. Pu & A.M. Wyglinski, *Software-Defined Radio for Engineers*, Artech House (2018), p.171 at www.analog.com/en/education/education-library/software-defined-radio-for-engineers.html