DOI: 10.1515/aon-2019-0010

MEASUREMENTS OF SIGNAL DELAYS IN SOFTWARE DEFINED RADIO WITH USE OF GNSS MODULES

Oskar Mężyk, Michał Doligalski, Ryszard Rybski
Institute of Metrology, Electronics and Computer Science,
University of Zielona Góra

ABSTRACT

In the work a method of latency measurement in software defined radio (SDR) is proposed and validated. The test setup uses customer grade GNSS modules as reference time sources and enables relative delay calculation between signals received directly and those bypassed through SDR platform. The method is hardware agnostic in a sense, that it does not involve any custom software or hardware modifications. Tests that compare reported carrier-to-noise ratio and positioning errors were performed to prove functionality of such system. Additionally, authors measured several *gnuradio* blocks with respect to their impact on total latency introduced into signal path. All tests were performed on a bladeRF low-cost RF front-end. Minimum observed latency for the signal was below 10 ms.

***Keywords* – GNSS, SDR**

INTRODUCTION

Software Defined Radio is a common name for set of software and hardware tools used for research and development in the field of telecommunication. Typically, such set consists of an RF front-end that acquires signal from the antenna and a computer program that realizes data processing algorithm. Such toolset, due to its flexibility, enables rapid implementation of new signal processing techniques without a need of costly development of dedicated hardware.

Architecture of SDR hardware platforms varies between applications, but in all of them the split between “hardware” and “software” processing happens right after signal is down-converted and digitized by analog-to-digital converter. Samples may be sent directly to the PC or pre-processed by an on-board FPGA (hybrid solution), which the middle-level designs usually have. In all cases, when samples leave hardware, they are transported to the operating system (OS) of the PC via USB or LAN interface, where drivers, software libraries and high-level applications take care for manipulation of data. While signal samples are moved through the layers of bus/network protocols and then into the OS, they experience various delays due to buffering, bus congestion, packet routing etc. Each of this step and its delays may be subject to a separate analysis, but since it is rarely possible to control them in a typical PC – the summed latency is of interest.

The *gnuradio* is a one of the most popular software frameworks for RF signal manipulation. It may work as a software only implementation or it can interface hardware front-end and process real, sampled data. It is designed for efficient data processing and became a basis for variety of demodulators and receivers, including also GNSS software receivers [Falone, 2014].

A proposed method of measurement of total SDR processing latency takes an advantage of the global precise timing tool – the Global Navigation Satellite System (GNSS) network. Even with use of consumer grade modules it is possible to measure time of events down to nanoseconds precision and authors exploit that possibility. The main motivation of the work is to establish ability of SDR platforms to manipulate and reconstruct GNSS signals in real-time.

Next section reviews previous works on the *gnuradio* latency analysis. It is followed by the measurement method description and results presentation. Paper ends with discussion on achieved results and conclusion sections.

PREVIOUS WORKS

The root concept of *gnuradio* is to maximize throughput by using high data parallelism. To enable that, a dynamic buffering scheme has been implemented so that a PC system capabilities are utilized at maximum. Such approach leads to a non-deterministic processing workload and resulting processing time by default [Müller, 2017]. Basic implementation has been made with highest throughput in mind, but with advancing processing powers of desktop PCs, the latency aspect had to be addressed.

In [Suganuma et al., 2012] authors design and implement a method of dynamic buffer scaling, optimizing latency for processing blocks and whole signal flow path. This method has been implemented in the *gnuradio* and its impact can be seen in achieved results of this paper.

A comprehensive work about SDR latency is [Truong et al., 2013]. Authors analyse different layers of the SDR system with use of time-stamping data and Round-Trip Time (RTT) measurements. The measurement setup included two SDR hardware platforms, wireless modules and a set of additional software tools to perform packet transmission loop back. In this way all layers of software and hardware have been tested, but the method is rather complicated to repeat without detailed knowledge of said time stamping tools and is also hardware dependant.

Similar approach may be found in [Schmid, 2007] although prior to actual packet communication authors perform a simpler test using function generator and a mixed hardware-software edge detection mechanism. With this setup more accurate measurement of USB latencies were performed, but still – the method is dependent on the available hardware and its modifications.

In this paper, authors propose a different approach to SDR latency measurements, which to authors' best knowledge, may be more universal and simpler to implement. The method is described in following sections.

MEASUREMENT METHOD

Basic idea of the method is shown on the figure 1 – the signal from antenna is split and routed separately into two separate GNSS modules. In one of the paths the SDR block is inserted which can be seen as a delay block in signal path. The GNSS module that is connected directly to RF signal stays in lock and serves as a time reference for the incoming 1 PPS (one pulse per second) signal from the other module that receives signal from SDR block. This way the relative time difference between actual GNSS time and time based on the delayed signal can be measured.

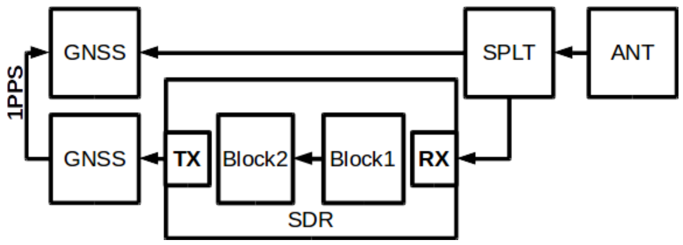


Fig.1: SDR seen as a delay block for antenna signal. Additional equipment not shown.

The method assumes following setup nuances:

- the SDR block contains hardware platform that is a full duplex transceiver,
- said hardware platform has enough bandwidth to process GNSS data,
- the PC computer (included in the SDR block) has enough resources to process data at required sampling frequency,
- manipulation of data is reversible or does not influence signal structure significantly,
- forward and reverse manipulation have similar computing power requirements, as they need to be included both in the signal path.

From the two latter requirements one can clearly see that not all processes may be measured this way – for the purpose of this work following *gnuradio* blocks have been chosen:

- delay,
- FFT and IFFT pair,
- FIR & FFT filter,
- FFT cross-correlation with unit function.

All signal processing algorithms were implemented using *gnuradio-companion* application.

CONCEPT VALIDATION

In order to proceed with measurements, the setup was created and validated for proper operation. For the SDR hardware platform the *bladeRF x40* [Nuand, 2019] has been chosen – it is a relatively advanced platform that allows processing data up to 20Mhz of bandwidth and uses USB3.0. Its input frequency range includes GPS L1 band, and for reason of simplicity only this signal was used during measurements. Since measured delays were in milliseconds range it is assumed that actual GNSS system used, as well as antenna quality, had of little influence on results.

Other equipment included GNSS modules: u-blox NEO-M8T and Telit 873, attenuators, low-noise amplifier (LNA), RF power splitter, GPS active antenna and a bias-tee. Full setup is shown on figure 2. Signal was split from a single antenna, after bias tee and LNA so their influence on relative timing may be omitted. The LNA was introduced to boost signal for the bladeRF front-end as it has much lower sensitivity than GNSS modules. Overall gain of active antenna and LNA was around 50dB and it imposed a need to add a 10dB attenuator on the input of Mod1. The 10dB attenuator on the output of SDR served as a protection from overloading the Mod2 module input. PC computer included in the SDR block was a desktop PC with quad-core CPU, 16GB of RAM and SSD.

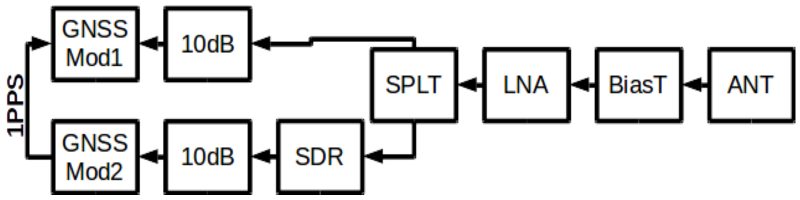


Fig.2: Splitting signal just before SDR eliminated cable delays from measurements.

To validate the concept first direct and SDR-bypassed signals were input into two same (NEO-M8T) GNSS modules and output was monitored for fix quality and carrier to noise ratio (CNo). Sampling frequency of bladeRF front-end was set to 4Msps (IQ samples per second) and input/output bandwidth to 2.8Mhz to cover GPS L1 band with a margin. Since all used GNSS modules had SAW filters on their inputs, the actual margin had little influence on the introduced noise. Transmit gain of SDR was adjusted so that the reported value of AGC of both modules was similar. Figure 3 shows carrier to noise ratio as reported by the Mod2 for visible satellites with value greater than 25dB and status flag "code and carrier locked" available in UBX-NAV-SAT message of UBX protocol.

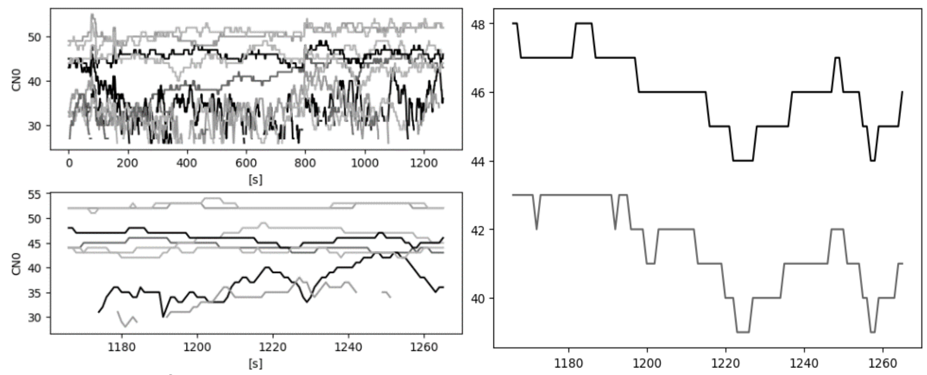


Fig.3: Left: reported CNo of SDR-bypassed satellite signals; right: compared direct and bypassed CNo for GPS satellite SV12 – a visible 5dB drop for bypassed signal. Sampling rate 4Msps.

The CNo reported by GNSS modules was collected and compared. The average loss in CNo of 3.4 to 4.9dB was observed for Mod2, but it maintained lock which proved that the whole setup works correctly.

In order to better understand quality of the bypassed signal, the Mod2 was interrogated for jamming indicator as described in u-blox M8 Receiver Description [u-blox, 2018] in section 14.2. The value of this indicator is relative and it indicates presence of narrowband disturbance on the receiver input versus "normal" conditions. For bypassed signal this value raised significantly exposing known limitations of the chosen RF front-end hardware. Since bladeRF is a zero-IF architecture it suffers from large LO leakage that is visible in the middle of frequency spectrum. This large "spike" saturates AGC of GNSS module and causes drop in overall CNo reported by Mod2. To overcome the issue RF front-end was calibrated using its internal functionality and TX gain was reduced, but still indicator showed significant interference.

Since further, precise calibration of TX path required high quality vector network analysers authors validated possibility to move the centre frequency away from the middle of the input pass-band. To achieve that the RX and TX frequency were set to 1573Mhz, sampling rate was increased to 8Msps and input/output bandwidth extended to 7Mhz. This way GPS L1 was still covered with the increased bandwidth and the LO leakage spike was shifted away from the spectrum of interest. As can be seen on figure 4 reported CNo maintained high values, enabling valid lock of the GNSS module. Average CNo drop for this scenario was measured in range of 1.6 to 2.4dB so it could have been chosen as solution. Unfortunately, at 8Msps the *gnuradio* reported occasional buffer overruns which caused signal discontinuities and variations in NEO-M8T internal clocks, which in turn led to erratic time difference measurement. For further tests authors chose reduced sampling rate and lower CNo reported against higher quality signal, but with difficult to control timing variations.

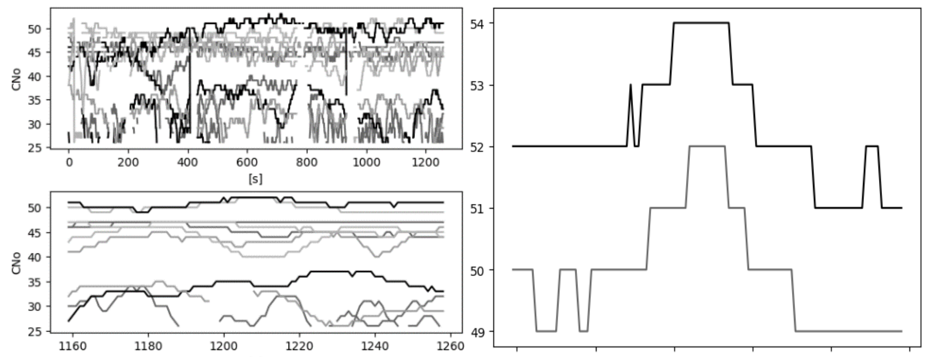


Fig.4: 8Msps scenario; left: reported CNo of SDR-bypassed satellite signals; right: compared direct and bypassed CNo for GPS satellite SV11 – a visible 2-3dB drop for bypassed signal.

As a final confirmation of the proper operation of Mod2 its reported position was compared to Mod1, using data from GPGLL NMEA message. Shown on figure 5 is the position trace within 5 minutes of continuous operation after acquiring fix by Mod2.

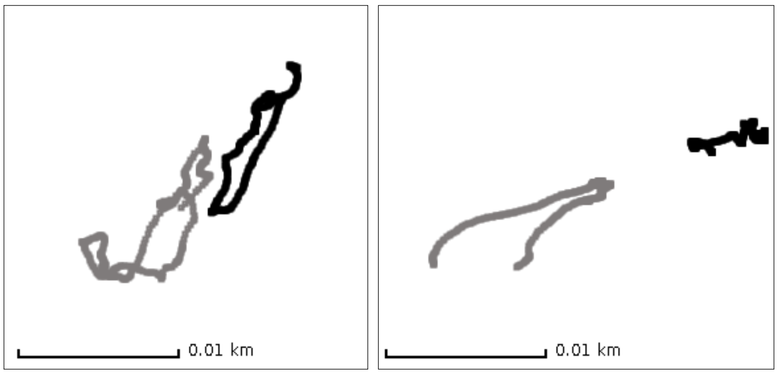


Fig.5: Reported position for direct (black) and SDR-bypassed (grey) signals; left: 4Msps at 1575.42Mhz; right: at 8Msps at 1573Mhz.

Smaller reported position differences and lack of signal discontinuities proved that scenario with 4Msps at 1575.42Mhz is the right choice for conducting latency measurements described in next section.

SDR LATENCY MEASUREMENTS

For the actual SDR latency measurement Mod2 was replaced by Telit 873 module. It has inherent delay on 1PPS signal output which made relative measurements always positive and reduced complexity on calculations. The 400 s test run showed average relative delay (between Mod1 and Mod2) of 110.90 ns with std dev of 10.7 ns with both modules locked directly to satellite signal. The quality of signal was not monitored and it was assumed, that valid 1PPS output signal was enough to prove proper signal structure.

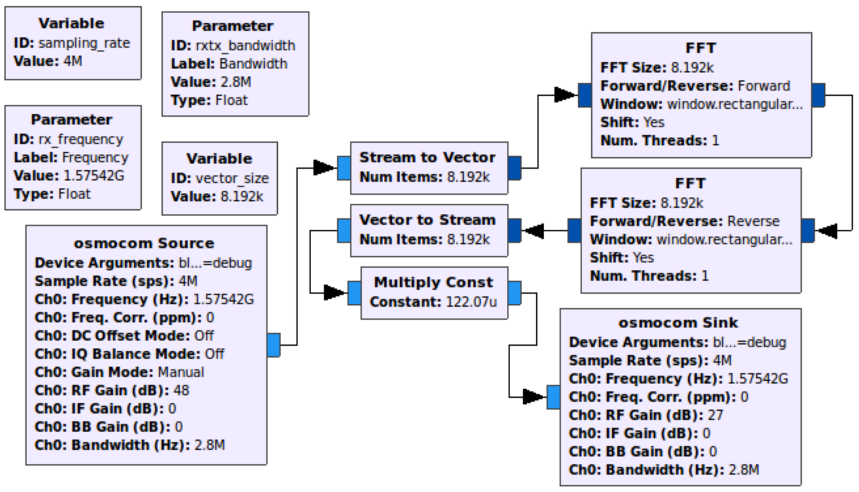
The *gnuradio* flow (block diagram) for basic measurements consisted only of *signal sink*, *signal source* and required minimum variables like *sampling rate* etc. For the SDR-bypassed signal, without any additional block, a latency of 9 ms and 444797 ns was measured in first run. In second run value of milliseconds stayed at 9 ms, but nanoseconds varied and stabilized at value of 472273. Standard deviation of the measured, stable runs was 9.9 ns and 13.35 ns accordingly.

Next the *delay block* was inserted in line with sample stream of *gnuradio*. The block is implemented as a linear buffer of samples with both positive and negative value of delay possible. Since negative delay would require sample dropping and cause discontinuities – only positive values were considered. Each run required power cycling of the Mod2 to perform a “cold start” on the module. Drastic change of delay block value on the fly required from GNSS module to adjust its internal clocks in a linear manner which could take several minutes and lead to occasional loss of lock.

Table 1: Gnuradio “Delay block” latency for different delay settings

Delay block value [ms]	**Measured value [ms]**	**Measured value [ns]**	**Std Dev [ns]**
Disabled	9	850763	9.1
0	12	833664	10.0
100	109	783703	9.7
250	253	661875	9.5

As a next measured block, the FFT was chosen. Since single transformation would change structure of the signal, a corresponding IFFT (Inverse FFT) was also included, as shown on figure 6.



Variable
ID: sampling_rate
Value: 4M

Parameter
ID: rxtx_bandwidth
Label: Bandwidth
Value: 2.8M
Type: Float

Parameter
ID: rx_frequency
Label: Frequency
Value: 1.57542G
Type: Float

Variable
ID: vector_size
Value: 8.192k

osmocom Source
Device Arguments: bl...=debug
Sample Rate (sps): 4M
Ch0: Frequency (Hz): 1.57542G
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 48
Ch0: IF Gain (dB): 0
Ch0: BB Gain (dB): 0
Ch0: Bandwidth (Hz): 2.8M

Stream to Vector
Num Items: 8.192k

Vector to Stream
Num Items: 8.192k

Multiply Const
Constant: 122.07u

FFT
FFT Size: 8.192k
Forward/Reverse: Forward
Window: window.rectangular...
Shift: Yes
Num. Threads: 1

FFT
FFT Size: 8.192k
Forward/Reverse: Reverse
Window: window.rectangular...
Shift: Yes
Num. Threads: 1

osmocom Sink
Device Arguments: bl...=debug
Sample Rate (sps): 4M
Ch0: Frequency (Hz): 1.57542G
Ch0: Freq. Corr. (ppm): 0
Ch0: RF Gain (dB): 27
Ch0: IF Gain (dB): 0
Ch0: BB Gain (dB): 0
Ch0: Bandwidth (Hz): 2.8M

Fig.6: Implementing FFT/IFFT pair in gnuradio requires additional scaling factor of 1/vector_size to restore amplitude of the signal.

During processing the data for FFT/IFFT pair measurements showed more frequent variations of the delay in range of hundreds of microseconds. For the FFT of 1024 length the variation between 5e6 to 9e6 ns occurred but while stable – standard deviation fell to tens of ns.

Table 2: Delays imposed by single FFT-IFFT pair of transformations with different vector lengths.

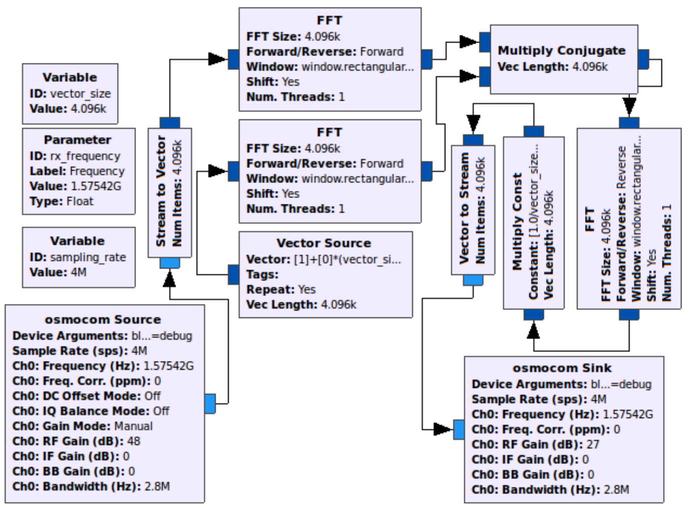
FFT size	Measured value [ms]	Min. value of [ns]	Max. value of [ns]
128	19	585390	585435
1024	15	524768	958053
8192	14	555371	796042

Another block measured was the frequency translating filter (*Frequency Xlating Filter*). Each time a low pass filter was designed with the *Filter Design Tool* present in *gnuradio* with parameters allowing that full GPS L1 band is passed without modification. Complexity (i.e. number of taps) was increased by reducing transition band of the filter, maintaining stop band beyond wanted signal frequency spectrum.

Table 3.: Only filters of highest complexity showed significant increase in latency.

Filter type	Taps	Measured value [ms]	Min. value of [ns]	Max. value of [ns]
FIR	27, Real	12	122554	989851
FIR	55, Real	23	313531	313667
FIR	107, Real	20	219318	633578
FIR	11, Complex	19	840126	840210
FIR	55, Complex	19	691773	691849
FIR	109, Complex	33	140008	712847
FFT	11, Complex	27	935515	935591
FFT	55, Complex	14	471051	859358
FFT	109, Complex	28	415597	415663

Final transformation taken under testing was cross-correlation of signals. According to theorem and Fourier transform characteristics cross correlation can be calculated in frequency domain, with operation flow as shown on the *gnuradio* diagram on figure 7. At first signal is transformed with use of FFT, then the correlated input is multiplied with complex conjugate of the correlating input. At final stage – inverse FFT and scaling is performed.



Variable
ID: vector_size
Value: 4.096k

Parameter
ID: rx_frequency
Label: Frequency
Value: 1.57542G
Type: Float

Variable
ID: sampling_rate
Value: 4M

Stream to Vector
Num Items: 4.096k

FFT
FFT Size: 4.096k
Forward/Reverse: Forward
Window: window.rectangular...
Shift: Yes
Num. Threads: 1

FFT
FFT Size: 4.096k
Forward/Reverse: Forward
Window: window.rectangular...
Shift: Yes
Num. Threads: 1

Vector Source
Vector: [1]+[0]*(vector_si...
Tags:
Repeat: Yes
Vec Length: 4.096k

Multiply Conjugate
Vec Length: 4.096k

Vector to Stream
Num Items: 4.096k

Multiply Const
Constant: [1.0/vector_size...
Vec Length: 4.096k

FFT
FFT Size: 4.096k
Forward/Reverse: Reverse
Window: window.rectangular...
Shift: Yes
Num. Threads: 1

osmocom Source
Device Arguments: bl...=debug
Sample Rate (sps): 4M
Ch0: Frequency (Hz): 1.57542G
Ch0: Freq. Corr. (ppm): 0
Ch0: DC Offset Mode: Off
Ch0: IQ Balance Mode: Off
Ch0: Gain Mode: Manual
Ch0: RF Gain (dB): 48
Ch0: IF Gain (dB): 0
Ch0: BB Gain (dB): 0
Ch0: Bandwidth (Hz): 2.8M

osmocom Sink
Device Arguments: bl...=debug
Sample Rate (sps): 4M
Ch0: Frequency (Hz): 1.57542G
Ch0: Freq. Corr. (ppm): 0
Ch0: RF Gain (dB): 27
Ch0: IF Gain (dB): 0
Ch0: BB Gain (dB): 0
Ch0: Bandwidth (Hz): 2.8M

Fig.7: Cross correlation flow diagram in gnuradio-companion application.

To not change the structure of the signal it was correlated with the unit function. Vector length (length of FFT) was changed between 1024, 4096 and 8192. Measured latency varied between 20 to 31 ms.

DISCUSSION

Achieved results indicate that proposed method may be a valuable tool for latency testing and optimisation for software defined radio. Separate measurement trials show good consistency. Observed variations in nanoseconds range were expected SDR behaviour. The reason for those variations was either the latency adaptation algorithm implemented in *gnuradio* or operating system impact. Since problem of the non-deterministic latency of *gnuradio* framework was not the main scope of this paper, it was not investigated further as well latency was not optimised.

The method itself has some drawbacks, which must be pointed out. First, in poor GNSS satellite visibility conditions the SDR losses ability to properly bypass radio frequencies with such low amplitudes. Increasing LNA gain may not be a wanted solution, as the noise of the whole input section will cover the inherently weak signals. In addition, imperfections of the zero-IF architecture decrease bypassed signal quality and are hard to counteract. This problem is related only to chosen RF front-end, so it may be addressed with better quality hardware or better calibration.

CONCLUSIONS

With proposed method, the latency introduced by SDR can be measured with nanosecond precision. Test setup was created, validated for proper operation and tested on several *gnuradio* blocks with success. In authors' opinion the solution described in the paper may help further optimise performance of software defined radio platforms.

BIBLIOGRAPHY

[1] Falone R., Stallo C., Gambi E., Spinsante S., "SDR GNSS receivers: A comparative overview of different approaches," *2014 IEEE Metrology for Aerospace*, Benevento, 2014, pp. 326-331. doi: 10.1109/MetroAeroSpace.2014.6865943

[2] Müller M., “Behind the Veil: A Peek at GNU Radio’s Buffer Architecture”, The GNU Radio Foundation, 5 Jan 2017, www.gnuradio.org/blog/2017-01-05-buffers/

[3] Nuand LLC, “bladeRF USB 3.0 Software Defined Radio”, 23 Mar 2019, www.nuand.com/bladeRF-brief.pdf,

[4] Schmid T., Sekkat O., B. Srivastava M., “An experimental study of network performance impact of increased latency in software defined radios.”, *Proceedings of the second ACM international workshop on Wireless network testbeds, experimental evaluation and characterization* (WinTECH '07), ACM, New York, 59-66. doi:10.1145/1287767.1287779

[5] Suganuma, H., Suzuki, M., Morikawa, H., “A Buffer Size Tuning Mechanism for Software-Defined Radios”, IEEE INFOCOM 2012, Hilton Orlando Lake Buena Vista

[6] Truong, N.B., Yu, C. "Investigating Latency in GNU Software Radio with USRP Embedded Series SDR Platform," *2013 Eighth International Conference on Broadband and Wireless Computing, Communication and Applications*, Compiegne, 2013, pp. 9-14. doi: 10.1109/BWCCA.2013.11

[7] U-blox AG, “u-blox 8 / u-blox M8 Receiver Description”, 6 Mar 2018, ubx-13003221