ARTICLE TYPE

**GNSS Software Defined Radio: History, Current Developments, and Standardization Efforts**

Thomas Pany$^1$ | Dennis Akos$^2$ | Javier Arribas$^3$ | M. Zahidul H. Bhuiyan$^4$ | Pau Closas$^5$ | Fabio Dovis$^6$ | Ignacio Fernandez-Hernandez$^7$ | Carles Fernández–Prades$^3$ | Sanjeev Gunawardena$^8$ | Todd Humphreys$^9$ | Zaher M. Kassas$^{10}$ | José A. López Salcedo$^{11}$ | Mario Nicola$^{12}$ | Mark L. Psiaki$^{13}$ | Alexander Rügamer$^{14}$ | Young-Jin Song$^{15}$ | Jong-Hoon Won$^{15}$

$^1$University of the Bundeswehr Munich, Neubiberg, Germany
$^2$University of Colorado, Boulder, USA
$^3$Centre Tecnològic de Telecomunicacions de Catalunya, Barcelona, Spain
$^4$Finnish Geospatial Research Institute, Kirkkonummi, Finland
$^5$Northeastern University, Boston, USA
$^6$Politecnico di Torino, Turin, Italy
$^7$European Commission, Brussels, Belgium
$^8$Air Force Institute of Technology, Wright-Patterson AFB, USA
$^9$The University of Texas at Austin, Austin, USA
$^{10}$The Ohio State University, Columbus, USA
$^{11}$Universitat Autònoma de Barcelona, Cerdanyola del Vallès, Spain
$^{12}$LINKS Foundation, Turin, Italy
$^{13}$Virginia Tech, Blacksburg, USA
$^{14}$Fraunhofer Institute for Integrated Circuits IIS, Erlangen, Germany
$^{15}$Inha University, Incheon, South Korea

Summary

Taking the work conducted by the global navigation satellite system (GNSS) software-defined radio (SDR) working group during the last decade as a seed, this contribution summarizes for the first time the history of GNSS SDR development. It highlights selected SDR implementations and achievements that are available to the public or that influenced the general SDR development. The relation to the standardization process of intermediate frequency (IF) sample data and metadata is discussed, and an update of the Institute of Navigation (ION) SDR Standard is proposed. The work focuses on GNSS SDR implementations on general purpose processors and leaves aside developments conducted on field programmable gate array (FPGA) and application-specific integrated circuits (ASICs) platforms. Data collection systems (i.e., front-ends) have always been of paramount importance for GNSS SDRs, and are thus partly covered in this work. The work represents the knowledge of the authors but is not meant as a complete description of SDR history.

**KEYWORDS**

GNSS, software-defined radio

1 | INTRODUCTION

Receiver development has always been an integral part of satellite navigation, ever since the early studies conducted for the U.S. Global Positioning System (GPS). The very first receivers were huge devices, realizing the correlation of the received satellite signal with internally generated code and carrier replicas by a mixture of digital and analog electronics (Eissfeller & Won 2017). Advances in semiconductor technology soon enabled signal processing on dedicated chips. This technology was of course complex to handle and mostly located within the U.S. industry. Despite the success of GPS and its Russian counterpart Globalnaya Navigazionnaya Sputnikovaya Sistema (GLONASS), receiver internal technology was barely accessible to the broader research community for a long time, as it seemed to be impossible to realize GNSS signal processing on low-cost computers. Even in the year 1996 a key receiver design pioneer expressed skepticism that general purpose microprocessors were, or would ever be, a suitable platform for implementing a GNSS receiver (Kaplan 1996).
The situation radically changed when the algorithms of a GPS receiver were first implemented as Matlab software on a desktop personal computer (PC) and estimates of digital signal processor (DSP) resources required to run the algorithms in real-time were encouraging (D. Akos & Braasch, 1996; D. M. Akos, 1997). Soon after, real-time processing was demonstrated even on conventional PCs and the widespread use of software radio technology took off with exponential growth. Interestingly, software radio technology did not replace existing hardware receivers usually realized as one or more ASICs, but complemented these, allowing researchers to easily implement and test new algorithms or to develop highly specialized receivers with reasonable effort. Today, this is a well-established approach for military, scientific, and even commercial applications as described by Curran et al. (2018).

As different research groups developed their own software radios, they used different data collection systems to sample the GNSS signals. Whereas the data format of the digital GNSS signal streams is comparably easy to describe, the widespread use of software radio technology made it necessary to introduce a certain level of standardization, which was finally achieved by a group of researchers as documented by Gunawardena et al. (2021). The result was the so-called ION SDR Standard (ION SDR Working Group, 2020).

As technology further evolved, new GNSS software radios emerged and some deficiencies of the ION SDR Standard became apparent (Clements et al., 2021). These conditions prompted the present paper, whose contributions are four-fold. First, it presents the first history of GNSS SDR development (Section 2). Second, it offers a detailed description of select GNSS SDRs (Section 3). Third, it overviews recent front-end developments (Section 4). Finally, it summarizes the history of the ION SDR Standard and proposes an update thereto (Section 5).

### 2 | GNSS SOFTWARE DEFINED RADIO HISTORY

The history of GNSS SDR requires more than a bit of recollection, which can be fraught with inaccuracies, none of which are intentional in the present work. Corrections would always be welcome.

GNSS SDR traces its roots to Ohio University’s Avionics Engineering Center around 1994. Professor Michael Braasch, a newly-minted faculty member of the Electrical and Computer Engineering Department and already recognized as an expert in GNSS multipath, was interested in creating a high-fidelity simulation of the internal signal processing within GPS and GLONASS receivers. Dennis Akos, a Ph.D. student in the Department, was intrigued by the idea. Already harboring a keen interest in computer science and programming, Akos took on the simulation project at Braasch’s request under the FAA/NASA Joint University Program. Meanwhile, publication of “The Software Radio Architecture” in the 1995 IEEE Communication Magazine (Mitola, 1995) fueled Akos’s and Braasch’s thinking that this “simulation” could instead be targeted toward an actual software radio implementation. The result was the first publication on GNSS SDR, which appeared in the proceedings of the 1996 ION Annual Meeting (D. Akos & Braasch, 1996).

Development of this initial simulation/implementation was significantly furthered through cooperation with Dr. James B. Y. Tsui of Wright Patterson Air Force Base. Well-recognized as an expert in digital receivers, Tsui had recently taken an interest in satellite navigation. In 1995, two summer interns, Dennis Akos from Ohio University and Michael Stockmaster from The Ohio State University, worked under Tsui’s guidance to develop a Matlab implementation of the signal processing required for basic GPS receiver operation. A digital oscilloscope was used to capture the initial IF data that were critical to developing and debugging those early algorithms. Akos was responsible for the lower-level signal processing (acquisition as well as code/carrier tracking), while Stockmaster implemented the navigation solution. The cumulative result was the first ever GPS SDR implementation. Although fully operational, it was “slow as molasses”: processing 30 seconds of IF data required hours of computation time. Tsui published the first textbook on GPS SDR in 2000 (Tsui, 2000). A parallel contribution of this initial effort was the direct radio frequency (RF) sampling front-end, which garnered significant interest and pushed advances in analog-to-digital converter development (D. Akos et al., 1999).

After receiving his Ph.D. in 1997, Akos started his academic career as an Assistant Professor in the Systemteknik Department of Luleå University of Technology in Sweden, where he taught a course on computer architecture. It was here that GPS SDR first achieved real time operation. For a class project, Akos provided a Matlab-based GPS SDR and challenged a group of students to “get it to run as fast as possible” subject to the requirement that the complex accumulation products for each channel were within 10% of those produced by the original Matlab-based GPS SDR. It was in 1999 that the first “real time” operation was possible, processing 60 seconds of IF data in 55 seconds. This was a notable achievement at the time given that renowned GPS expert Philip Ward, who was responsible for some of the first GPS receivers, had recently expressed skepticism about the prospect of a
fully software-defined real-time GPS SDR, writing “The integrate-and-dump accumulators provide filtering and resampling at the processor baseband input rate, which is around 200 Hz [...] and well within the interrupt servicing rate of modern high-speed microprocessors. But the 5- to 50-MHz rates [of intermediate frequency samples] would not be manageable” (Kaplan, 1996). This real-time implementation effort was led by student Per-Ludvig Normark and led to the results published by D. M. Akos et al. (2001).

In the meantime, Kai Borre, a geodesy professor from Aalborg University, had also developed in the mid-late 1990s Matlab code for GPS receivers. Borre’s code focused on the navigation block and including functions for conversion of coordinates and time references, satellite position determination, and atmospheric corrections. The joint efforts of Akos, Borre, and others would later lead to the well-known book (Borre et al., 2007), a primary reference for GNSS SDR over the next years, and the related SoftGPS Matlab receiver.

Upon graduation, Normark continued his GNSS receiver development with the GPS Laboratory at Stanford University and then returned home to Sweden where he co-founded NordNav Technologies, which developed the first Galileo SDR, and helped establish the architecture, together with Cambridge Silicon Radio (CSR), to push GNSS to a price point acceptable to the mobile phone adoption. CSR, at the time a dominant supplier of Bluetooth hardware to the mobile phone market, acquired NordNav in 2006 and they jointly redesigned the CSR 2.4 GHz radio to multiplex to the 1575.42 MHz GPS L1 band, exploiting the fact that most Bluetooth applications have a relatively low duty cycle. This approach, coupled with the real-time software GPS implementation, provided a near-zero-added-cost GPS receiver.

There have been numerous contributions to GNSS SDR development since these early years, many of which are from the co-authors of this paper. Selected developments by the authors are outlined in Section 3, including a survey of achievements by other researchers in Section 3.1. The authors are aware that many other important contributions are missing, and make no claims of establishing a comprehensive description. In order to give the reader a better orientation about the chronological order of all developments, we present Tab. 1, reiterating that the selection of references is partly subjective and often similar developments have been carried out by several research groups. The timeline demonstrates the flexibility of SDR technology, i.e., the same code base is used for GPS L1 C/A code signals and for signals of opportunity (SOP) from cellular terrestrial transmitters or from communication satellites in low Earth orbit (LEO).

## 3 CURRENT STATUS OF GNSS SOFTWARE DEFINED RADIOS

In June 2023, a quick internet search did not reveal any comprehensive listing of all GNSS SDRs and Wikipedia (2023) lists seven entries, which is far below the number of receivers known by the authors, even if the following criterion is applied to limit the scope: a GNSS SDR (or software receiver) is defined as a piece of software running on a general purpose computer converting samples of a received GNSS signal into a position velocity and time (PVT) estimate. It is clearly understood that a front-end including analog-to-digital conversion (ADC) is required to sample the received signal, but other than that no further functionality is allowed to be realized via hardware. With this definition, three categories of software receivers can be introduced:

- **real-time receivers**: monolithic or modular software packages written in an efficient low-level programming language (like C or C++) typically optimized for run-time efficiency and stability
- **teaching/research tools**: software packages written in a high level programming language like Python or Matlab optimized for code readability and flexibility
- **snapshot receivers**: receivers optimized for very short batches of signal samples

Furthermore, the software package shall allow some configuration flexibility and (at least theoretically) support the ION SDR Standard. The following subsections introduce a few selected developments, emphasizing the rationale behind design choices and current status. Each sub-section is represented by one entry in Tab. 2 to give the reader a quick overview of the main characteristics of each development. Section 3.1 describes the work of Psiaki, Ledvina, and Humphreys and their efforts in real-time processing on DSPs with the bit-wise-parallel approach proving to be highly successful even for space applications. Section 3.2 covers work of Pany/others in their efforts with multiconstellation/multifrequency GNSS. Section 3.3 and Section 3.4 cover the efforts of Borre and others in a downloadable open source Matlab GPS SDR started in (Borre et al., 2007), with the most recent GNSS update reported in (Borre et al., 2022). Akos has also continued this academic development of a suite of open source GNSS SDRs (Bernabeu et al., 2021). The widely used open-source receiver GNSS-SDR is described in Section 3.5.
<table>
<thead>
<tr>
<th>Year</th>
<th>Milestone with comment</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1995</td>
<td>Emergence of software radio approach</td>
<td>(Mitola, 1995)</td>
</tr>
<tr>
<td>1996</td>
<td>First publication of a GPS SDR development</td>
<td>(D. Akos &amp; Braasch, 1996)</td>
</tr>
<tr>
<td>1999</td>
<td>First real-time software receiver with GPS L1 C/A code</td>
<td>(D. M. Akos et al., 2001)</td>
</tr>
<tr>
<td>2000</td>
<td>First text book on GPS SDR published</td>
<td>(Tsui, 2000)</td>
</tr>
<tr>
<td>2002+</td>
<td>Use of bit-wise correlation and SIMD instructions</td>
<td>(Ledvina et al., 2003; Pany et al., 2003)</td>
</tr>
<tr>
<td>2002+</td>
<td>GNSS SDRs as commercial products</td>
<td>NordNav, IFEN, Trimble, Locus Lock, ...</td>
</tr>
<tr>
<td>2004</td>
<td>First multi-GNSS/multi-frequency GNSS SDRs</td>
<td>(Ledvina, Psiaki, Sheinfeld, et al., 2004)</td>
</tr>
<tr>
<td>2004</td>
<td>First real-time GNSS/INS integration with SDR</td>
<td>(Gunawardena et al., 2004)</td>
</tr>
<tr>
<td>2005</td>
<td>GNSS SDR consolidation at Politecnico di Torino and LINKS Foundation</td>
<td>Section 3.9</td>
</tr>
<tr>
<td>2005</td>
<td>Demonstration of vector tracking with a GNSS SDR</td>
<td>(Pany et al., 2005)</td>
</tr>
<tr>
<td>2006</td>
<td>First real-time all-in-view embeddable GNSS SDR</td>
<td>(T. Humphreys et al., 2006)</td>
</tr>
<tr>
<td>2006</td>
<td>First use of SDR technology for AM signals of opportunity</td>
<td>(McEllroy, 2006; McEllroy et al., 2006)</td>
</tr>
<tr>
<td>2007</td>
<td>Start of wide-spread adoption of SDR technology in GNSS research</td>
<td>(Borre et al., 2007)</td>
</tr>
<tr>
<td>2007</td>
<td>First development of a snapshot receiver</td>
<td>Section 3.8</td>
</tr>
<tr>
<td>2009</td>
<td>First multi-core GNSS SDR</td>
<td>(T. E. Humphreys et al., 2009)</td>
</tr>
<tr>
<td>2010</td>
<td>Adoption of a computer science best practice collaborative framework</td>
<td>Section 3.5</td>
</tr>
<tr>
<td>2010</td>
<td>First use of GPUs for correlation</td>
<td>(Hobiger et al., 2010)</td>
</tr>
<tr>
<td>2011+</td>
<td>Use of GNSS SDR for ionospheric research</td>
<td>(O’Hanlon et al., 2011; Peng &amp; Morton, 2011)</td>
</tr>
<tr>
<td>2012+</td>
<td>SDR developments at the Finnish Geospatial Research Institute</td>
<td>(Borre et al., 2022; Söderholm et al., 2016)</td>
</tr>
<tr>
<td>2012</td>
<td>Use of a DVB-T ultra-low-cost front-end for GNSS SDR</td>
<td>Section 3.5</td>
</tr>
<tr>
<td>2012+</td>
<td>Use of SDR technology for LTE signals of opportunity</td>
<td>(del Peral-Rosado et al., 2013; Driusso et al., 2017; Shamaei et al., 2018; Lightsey et al., 2014; Murrian et al., 2021)</td>
</tr>
<tr>
<td>2014+</td>
<td>Use of GNSS SDRs in space</td>
<td>(Yang et al., 2014)</td>
</tr>
<tr>
<td>2014</td>
<td>Use of SDRs for mixed cellular 3G GSM/CDMA and DTV SOP</td>
<td></td>
</tr>
<tr>
<td>2015+</td>
<td>Abundance of processing power for GNSS SDR available</td>
<td>(Dampf et al., 2015; Nichols et al., 2022)</td>
</tr>
<tr>
<td>2017+</td>
<td>Use of SDRs for 3G CDMA and 4G LTE SOP</td>
<td>(Kassas et al., 2017)</td>
</tr>
<tr>
<td>2018</td>
<td>First use of Python for dedicated teaching of GNSS SDR</td>
<td>Section 3.7</td>
</tr>
<tr>
<td>2018</td>
<td>First SDR enabling sub-meter-level carrier-phase-based UAV navigation with 3G CDMA and 4G LTE SOP</td>
<td>(Khalife &amp; Kassas, 2018)</td>
</tr>
<tr>
<td>2020</td>
<td>Formal adoption of ION SDR Standard</td>
<td>Section 5</td>
</tr>
<tr>
<td>2020</td>
<td>Use of SDR for stationary positioning with multi-constellation Orbcomm and Iridium LEO SOP</td>
<td>(Farhangian &amp; Landry, 2020; Orabi et al., 2021)</td>
</tr>
<tr>
<td>2021</td>
<td>First SDR for 5G SOP</td>
<td>(Shamaei &amp; Kassas, 2021b)</td>
</tr>
<tr>
<td>2021+</td>
<td>Use of GNSS SDR to support development of new navigation satellite systems</td>
<td>(Miller et al., 2023; Song et al., 2021)</td>
</tr>
<tr>
<td>2021</td>
<td>First SDR enabling vehicle navigation with multi-constellation LEO SOP</td>
<td>(Kassas et al., 2023)</td>
</tr>
<tr>
<td>2022</td>
<td>First SDR enabling aircraft navigation with cellular SOP</td>
<td>(Shamaei &amp; Kassas, 2021b; Kassas, Khalife, Abdallah, Lee, Jurado, et al., 2022)</td>
</tr>
</tbody>
</table>
The AUTONAV receiver used to support the development of Korean Positioning System (KPS) is discussed in Section 3.6 and PyChips (cf. Section 3.7) is the basis for tutorial classes of the ION. The UAB snapshot GNSS software receiver is described in Section 3.8, while Section 3.9 discusses a SDR used e.g. to the authentication schemes, reflectometry or to assess the influence of non-standard GNSS transmissions. Section 3.10 extends the scope of SDR to non-GNSS signals.

Whereas at the beginning of the GNSS SDR development the different receivers were linked to specific persons or research institutes, today often different receivers, tools or code bases are used at the same institute. On the other hand, code bases first developed by a single institute spread into different institutes. For example, the developments of Borre et al. (2007) forked into several branches [e.g. (Bernabeu et al., 2021; FGI, 2022; Zhang, 2022)], as discussed in Section 3.3 and Section 3.4.

### 3.1 Bit-Wise Parallelism and the Emergence of GRID

The original real-time GNSS software radio work by D. M. Akos (1997) inspired an effort within the Cornell GPS group. Psiaki had been working with non-real-time software GNSS signal processing in Matlab for about two years when he started to wonder whether the slow Matlab operations could be translated to run in real-time on a general desktop workstation. The bottleneck in GNSS digital signal processing occurs when doing the operations that initially process the high-frequency RF front-ends samples. RF front-ends typically sample at 4 MHz or faster. A 12 channel receiver would have to perform on the order of 400 million operations per second or more in order do all of the needed signal processing. Psiaki conceived the concept of bit-wise parallel processing as a means of addressing this challenge. He recruited then-Ph.D. candidate Brent Ledvina to make an attempt at implementing these ideas in the C programming language on a Real-Time Linux desktop workstation. Ledvina succeeded in developing a 12-channel real-time L1 C/A-code receiver after about 6 months’ effort (Ledvina et al., 2003).

The main concept of bit-wise parallelism is to work efficiently with RF front-end data that have a low number of quantization bits. If an RF front-end produces a 1-bit digital output stream, then 32 successive sign-bit samples can be stored in a single 32-bit unsigned integer word on a general-purpose processor. Thirty-two successive output samples of a 2-bit RF front-end can be stored in two 32-bit words, one containing the successive sign bits and the other containing the successive magnitude bits. Each channel of the software receiver generates a 1-bit or a 2-bit representation of 32 successive samples of its IF carrier replica, both in-phase and quadrature, and the successive samples are stored in parallel in 32-bit unsigned integer words. Similarly, it generates a 1-bit representation of 32 successive samples of its prompt pseudo-random noise (PRN) code replica and stores them in parallel in a single 32-bit unsigned integer word. It also generates an early-minus-late PRN code replica that requires 1.5 bits per sample, which takes up two 32-bit unsigned integer words to store 32 samples. These replica signals can be generated very efficiently by using pre-tabulated 32-bit words. The software receiver then performs a series of bit-wise AND, OR, XOR, and similar operations that have the effect of performing PRN code mixing and IF-to-baseband carrier mixing. The outputs of the mixing operations are contained in a small number of 32-bit words, the number of which depends on the number of bits in each RF front-end output sample and the number of bits in the IF carrier replicas.
The final operation is accumulation of the results in the 32-bit words. This involves sets of bit-wise Boolean operations, as per Ledvina et al. (2003), followed by summation of the number of 1-bits in the resulting 32-bit unsigned integer words. The bit summation operations proved to be a challenge in terms of minimizing execution time. Ledvina solved this problem by using a pre-computed 1-dimensional data table whose input was the unsigned integer and whose output was the number of 1-bits. In order to keep the table size reasonable, it only counted the bits in a 16-bit unsigned integer word. The original receiver’s 32-bit words were split in half, two table look-ups were performed, and the results summed in order to count all the 1-bits. The original algorithms are defined by Ledvina et al. (2003), Ledvina, Psiaki, Powell, & Kintner (2004), and Ledvina, Psiaki, Powell, & Kintner (2006). When using very long PRN codes, such as the L2C CL code, the original method’s whole-period PRN code tables of the proper 32-bit words at various code phases became impractically large. Therefore, a new method was developed for long PRN codes. It tabulates 32-bit words of short generic PRN code chip sequences, with all possible combinations of a short sequences of chips considered at various PRN code offsets relative to the start of the samples of the 32-bit word. Those methods are described by Psiaki (2006) and by Ledvina et al. (2007). This technique proved invaluable for dealing with long codes.

A processor that can operate on wider segments of data, up to 512 bits for current single instruction multiple data (SIMD) instructions, gains substantial additional signal processing speed increases (Nichols et al., 2022). Note, however, that the speed increase factors over brute-force integer calculations are typically not as high as the number of bits per word. That is, the techniques do not speed up the operations by a factor of 32 when processing 32 samples in parallel by using 32-bit words to represent 32 samples. For a 2-bit RF front-end and a 32-bit processor, the speed-up factor might be only 4 because the bit-wise parallel approach requires multiple operations due to, say, a simple multiplication of one time series by another. If one doubles the number of bits per word, however, then the speed tends to double. A particularly helpful feature of some recent processor designs is their inclusion of a hardwired command to count all the 1 bits in a word. This “popcount” intrinsic obviates the table look-ups that counted 1-bits in the original bitwise parallel design. If the number of bits increases in the RF front-end samples and/or the IF carrier replicas, however, then the bit-wise parallel method of signal processing slows down. Signals represented by 3 or 4 bits might cause the processing speed gains of bit-wise parallel algorithms to be limited or even non-existent.

After getting the basic algorithms working in real-time using 32-bit words, the Cornell group showcased the efficacy of real-time GNSS software radio by using the techniques to develop a dual-frequency L1 C/A and L2C receiver (Ledvina, Psiaki, Sheinfeld, et al., 2004) and a GPS/Galileo L1 civilian receiver (Ledvina, Psiaki, Humphreys, et al., 2006). These real-time software GNSS receivers each required only several person-days to develop them from the original L1 C/A code receiver. Of course, the L1/L2 receiver required a new dual-frequency RF front-end. The GPS/Galileo receiver required knowledge of the civilian Galileo E1 PRN codes, which had not been published at that time. This requirement led to a supporting effort which successfully deduced the Galileo GIOVE-A E1 PRN codes by recording their raw RF front-end samples and post-processing those samples using a suite of custom-designed SDR signal processing algorithms in order to pull the chips out of the noise (Psiaki et al., 2006).

The next development was to re-implement the bit-wise parallel code for embedded (low-power, low-cost) processing. Initially targeting a Texas Instruments DSP, this work was accomplished in 2006 by then-Ph.D. candidate Todd Humphreys (T. Humphreys et al., 2006). Later, as a professor at The University of Texas at Austin, Humphreys and his students—notably Jahshan Bhatti and Matthew Murrian—undertook a sequence of significant expansions and improvements to this receiver. Called GRID, the C++-based UT Austin receiver is by now a highly-optimized science-grade multicore GNSS SDR (T. E. Humphreys et al., 2009; Nichols et al., 2022) with its main features summarized in Tab. 3. It was the first GNSS SDR to be adapted for spoofing (T. E. Humphreys et al., 2008), the first GNSS SDR to operate in space (Lightsey et al., 2014), the first receiver of any kind to show that centimeter-accurate GNSS positioning is possible with a smartphone antenna (K. M. Pesyna Jr. et al., 2014), the first receiver to be used to locate terrestrial sources of GNSS interference from low-Earth orbit (Murrian et al., 2021), and is the basis of the current state-of-the-art in urban precise (dm-level) positioning (T. E. Humphreys et al., 2020; Yoder & Humphreys, 2023). As detailed in Nichols et al. (2022), GRID has also reaffirmed the commercial viability of GNSS SDR in widespread low-cost applications: it was recently licensed by a major aerospace company for use across all company operations, including in the thousands of satellites of the company’s broadband Internet mega-constellation.

3.2 Multi Sensor Navigation Analysis Tool

The Multi Sensor Navigation Analysis Tool (MuSNAT) is an object-oriented but monolithic C++ software receiver maintained by the University of the Bundeswehr Munich (UniBwM) and has been first mentioned in its present form by Pany et al. (2019).
<table>
<thead>
<tr>
<th>Feature</th>
<th>Solution</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>GNU/Linux, macOS, Windows</td>
<td>Proposed ION SDR Standard will accommodate</td>
</tr>
<tr>
<td>Programming environment</td>
<td>C++</td>
<td>See Nichols et al. (2022)</td>
</tr>
<tr>
<td>IF sample file input source</td>
<td>A wide array of formats</td>
<td>Requires PpEngine module</td>
</tr>
<tr>
<td>Real-time sample input</td>
<td>Yes</td>
<td>Nearly all open spreading codes and navigation message streams supported</td>
</tr>
<tr>
<td>Additional sensors</td>
<td>IMU, Cameras, LiDAR</td>
<td></td>
</tr>
<tr>
<td>Supported GNSS</td>
<td>GPS, Galileo, BeiDou, SBAS, QZSS, CDMA</td>
<td></td>
</tr>
<tr>
<td>Acquisition</td>
<td>Multi-threaded and FFT-optimized</td>
<td></td>
</tr>
<tr>
<td>Tracking</td>
<td>Vectorized, multicore, Intel SIMD (SSE2 through AVX-512) and ARM NEON (64-bit and 128-bit) accelerations</td>
<td>Correlation no longer the primary bottleneck under some configurations; see Nichols et al. (2022)</td>
</tr>
<tr>
<td>Measurement output</td>
<td>All standard GNSS observables</td>
<td>Proprietary GBX format plus RINEX, NMEA, RTCM, Matlab MAT-file, KML</td>
</tr>
<tr>
<td>Navigation</td>
<td>Extended Kalman filter based on pseudorange and Doppler measurements</td>
<td>Carrier-phase-based positioning available with PpEngine module</td>
</tr>
<tr>
<td>Further features</td>
<td>Vector tracking, multi-antenna, IMU integration, space-ready, interference mitigation &amp; detection</td>
<td></td>
</tr>
<tr>
<td>Availability</td>
<td>Source code available via commercial license from UT Austin</td>
<td>Turnkey solutions available via Locus Lock</td>
</tr>
</tbody>
</table>

It started as an operational real-time receiver development, but currently it mostly serves to develop and demonstrate innovative signal processing and navigation algorithms. Furthermore, it is used for teaching. It is freely available as executable for academic purposes from UniBwM (2023). Its main characteristics can be found in Tab. 4. In contrast to the bit-wise approach of Section 3.1 (that allows to design very power-efficient implementations), the design idea of MuSNAT and its predecessors was to realize a high-end receiver running on powerful PCs or workstations. The bit-wise approach was replaced by using SIMD instructions of Intel/AMD central processing units (CPUs). This allows to represent samples as 8-bit or 16-bit values and SIMD instructions like AVX-512 currently allow processing of registers of up to 512 bit (i.e. 32 16-bit samples) in parallel.

The GNSS software receiver developments started at UniBwM in 2002 after it became clear that the software radio approach discovered by D. Akos would provide useful insights into GNSS receiver technology and thus will be indirectly very helpful to design and build the Galileo navigation satellite system. The first software receiver at UniBwM was GPS L1 C/A only and was realized as a Matlab/Simulink project working in post-processing. To sample the GNSS signals, a commercial ADC with a peripheral component interconnect express (PCIe) connector from NI was used (PXI 5112) that was connected either to a low-bandwidth GPS L1 C/A code front-end based on the Plessey GP 2010 RF chip set and later on to one GPS L1/L2 high-bandwidth front-end, which was specifically developed by Fraunhofer IIS (Pany, Förster, & Eissfeller, 2004). Soon after, the software to communicate with the ADC (written in C++ making use of the Microsoft Foundation classes) was upgraded to a full GPS L1 C/A plus L2CS (L2 medium length code was supported only, not the long code) receiver. A detailed analysis published by Pany et al. (2003) revealed that not only the SIMD instruction set was important for the real-time capability but also the size and structure of the CPU caches. Memory bandwidth is one of the key issues when representing samples by multiple bits. One of the first achievements with this receiver was the demonstration of vector tracking (Pany et al., 2005).
Based on those results, funding to support a group of five researchers over three years was secured. This allowed starting a new software receiver project, this time making full use of C++ features for object-oriented development, and development of a graphical user interface (GUI) connected to the processing core via a clearly defined interface also allowing to run the core without GUI. The overarching development goal at that time was to realize a high-quality multi-GNSS multi-frequency receiver on a desktop PC or powerful laptop that could potentially be operated on a continuous basis to replace the (at that time) rather inflexible and expensive commercial GNSS receivers at continuously operating reference stations (CORSs). A concise overview of the development during those years was written by Stöber et al. (2010) and shows the improvements compared to the start of the project laid down by Pany, Eissfeller, et al. (2004).

A loose cooperation with IFEN GmbH was initiated that eventually resulted in the SX3 receiver (IFEN GmbH, 2022). IFEN used the processing core as initial basis, improved the core, replaced the GUI, and developed new dedicated front-ends. The C++ code was further optimized to support more channels at higher bandwidth and almost instantaneous high-sensitivity acquisition with the graphics processing unit (GPU) (GPS World staff, 2012). Also, semi-codeless tracking of GPS L2P(Y) (i.e. P-code aided cross-correlation) was implemented. The cooperation of UniBwM with IFEN lasted until 2013 when the development directions started to diverge. IFEN used the software mostly as base receiver platform with an application programming interface (API) to support different applications, whereas UniBwM continued to modify the core, which was not always beneficial for software stability if seen from a commercial point of view.

The focus at UniBwM switched in 2017 as the old GUI could not be maintained anymore. Furthermore, real-time operation became less important as most scientific results were obtained in post-processing. The result was that a new GUI was developed and attached to the proven processing core. Any run-time optimizations within the processing core degrading navigation performance (i.e. mostly causing additional noise in the code tracking loop) were removed. The core’s logging output was directed to a SQL database to store all different kind of intermediate results in a single file (additionally to the legacy ASCII logging into multiple files). A dedicated visualization tool for this database was developed.

The use of Windows and Visual Studio for developing a software radio is a little unusual, but is explained as follows. At UniBwM, most researchers use Windows PCs to allow easy document exchange with each other and most importantly within the European space industry. For this reason, all software receiver developments were done for Windows only. In terms of numerical performance and code optimization, Intel provided and still provides with the Intel C++ compiler and the Intel Performance Primitives the same quality on Windows as for Linux. Over the years it became, however, also clear that the potential use of the processing core on embedded devices and long-term stability might have been easier to achieve on the Linux operating system. IFEN ported part of the core to Linux, but not the full software receiver, and showed that conventional desktop CPUs and embedded CPUs provided an impressive processing capability already in the year 2015 (Dampf et al., 2015).

As already mentioned, code optimization to achieve fast (and real-time) signal tracking was a main research focus in the first years. Different studies on CPU assembler instructions, CPU architecture and bottlenecks resulted in dedicated assembler implementations. Extensive lookup-tables were used and one very efficient correlator implementation with the Intel x86 pmaddubsw-instruction was based on a signal sample representation as unsigned integers (including the necessary rewriting of the correlation formulae due to the switch from the standard representation of samples as signed integers to unsigned integers). FFT based acquisition was already very efficient on the CPU and even more efficient on the GPU. The use of FFT libraries provided by NVIDIA made the acquisition code porting from CPU to GPU comparably easy. The situation is different for signal tracking. The tracking code has been transferred to the GPU, and some optimization have been applied to minimize the amount of data transfer between CPU and GPU. However, since the correlation parameters are slightly different for each signal tracked, the correlation code is called multiple times and the latency to start one thread on the GPU generated significant overhead. GPU-based tracking is thus currently only beneficial if a very large number (several hundreds) of correlators is configured per tracking channel, as pointed out by Pany et al. (2019). As modern desktop and laptop CPUs continue to improve and make use of a many-core structure, the need to port signal tracking to the GPU becomes less important. Furthermore, the use of dedicated assembler code required over the years continuous adaptation to new CPU instruction sets (e.g. from SSE to AVX instructions). The performance gained by using hand-coded assembler routines compared to the use of the libraries provided by Intel (IPP) is not always worth the effort and was not further actively pursued. Instead, dot-product routines (2 x 16-bit signed input to 64-bit output) from the IPP are employed for signal tracking.

The C++ universe is huge, and it is easy to integrate external source code. For example, the famous RTKLIB and the ION SDR sample reader code have been integrated. The current research work with MuSNAT focuses on GNSS/INS/LiDAR integration, support of massive antenna arrays (Dötterböck et al., 2023), vector tracking and deep GNSS/INS coupling, support for LTE/5G-signals and GNSS signal simulation. It has to be admitted that the maintenance of the huge C++ code-base of MuSNAT at a
### TABLE 4 Main Features of MuSNAT

<table>
<thead>
<tr>
<th>Feature</th>
<th>Solution</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>Windows 10/11</td>
<td>Compiles as GUI or as command line version (port of command line version to Linux under preparation)</td>
</tr>
<tr>
<td>Programming environment</td>
<td>Microsoft Visual Studio 2019 C++</td>
<td>CUDA, Intel OneAPI, vcpkg and .net for GUI</td>
</tr>
<tr>
<td>IF sample file input source</td>
<td>ION SDR Standard and proprietary file readers</td>
<td>proprietary readers faster than ION SDR reader</td>
</tr>
<tr>
<td>Real-time sample input</td>
<td>yes, via TCP/IP</td>
<td>server available via LabView for selected NI USRPs</td>
</tr>
<tr>
<td>Additional sensors</td>
<td>LiDAR, IMU</td>
<td>LiDAR uses PCL format, IMU proprietary ASCII format, video formats supported but not yet used nearly all open spreading codes available and at least for each system one navigation message decoder</td>
</tr>
<tr>
<td>Supported GNSS</td>
<td>GPS, Galileo, BeiDou, GLONASS, SBAS, OFDM (LTE, 5G, ...)</td>
<td>nearly all open spreading codes available and at least for each system one navigation message decoder</td>
</tr>
<tr>
<td>Acquisition</td>
<td>optimized fast Fourier transform (FFT) method</td>
<td>CPU and GPU supported</td>
</tr>
<tr>
<td>Tracking</td>
<td>dot-product from Intel Performance Primitives (CUDA version for massive multi-correlator applications)</td>
<td>computational performance mostly limited by memory bus width</td>
</tr>
<tr>
<td>Further features</td>
<td>multi-antenna, signal-generator, primary-secondary tracking, SQL database for logging, vector tracking, GNSS/INS integration, RTKLIB</td>
<td>via Matlab-interface support of Galileo OSNMA/HAS and synthetic aperture processing</td>
</tr>
<tr>
<td>Availability</td>
<td>Executable plus data visualizer downloadable via UniBwM (2023)</td>
<td>Source code available for research projects with UniBwM</td>
</tr>
</tbody>
</table>

University institute with a high fluctuation of researchers is partly demanding. The learning curve for good C++ development in this context is steep and for the purposes of obtaining a PhD degree often an inefficient way. Therefore, interfaces from the C++ code to Matlab were established and for example Open Service Navigation Message Authentication (OSNMA) decoding, PPP-computation for High Accuracy Service (HAS) or LiDAR odometry are implemented in Matlab. Another development is to use MuSNAT to generate multi-correlator values that are then used within a full Matlab based receiver to emulate signal correlation via interpolation (Bochkati et al., 2022). Bochkati et al. (2023) use this for ease of development of synthetic aperture algorithms.

UniBwM has initially used front-ends from Fraunhofer IIS and the software receiver included low-level universal serial bus (USB) drivers for real-time data transfer. The same approach was used to connect the front-ends from IFEN GmbH to the processing core. The effort to write stable high data-rate low-level drivers is significant and introduces a dependency on libraries and support from the USB chip manufacturers. To reduce these kinds of development efforts, the decision to connect front-ends via TCP/IP was felt. This approach is powerful in terms of bandwidth and also generic and a first version of it is described in (Arizabaleta et al., 2021). Furthermore, with e.g. LabVIEW from NI it is comparably easy to develop a simple TCP/IP signal source for universal software radio peripheral (USRP) frontends. At the time of writing this paper, a more efficient firmware for USRPs with direct FPGA programming is being developed and shall allow to synchronously capture data from an inertial measurement unit (IMU) together with the GNSS signal samples.
3.3 | SoftGPS, SoftGNSSv3.0, and Derivatives

As abovementioned, (Borre et al., 2007) and the associated Matlab receiver was a cornerstone for GNSS SDR development. This receiver, initially called SoftGPS, then SoftGNSS (usually referred to as SoftGNSSv3.0), included the basic processing functions for GPS L1 C/A in a readable format, was very useful for educational purposes. These included signal FFT-based acquisition, frequency, carrier phase and code phase tracking, data synchronization and demodulation, pseudorange generation, and eventually PVT. The Matlab code, together with some samples, was provided in a CD with the book, and was also available at Aalborg University’s Danish GPS lab website. Apart from K. Borre and D. Akos, SoftGNSS included relevant contributions by D. Plausinaitis and others. Unfortunately, Kai Borre passed away in 2017 and the Danish GPS Lab was discontinued. However, SoftGNSS and its derivatives remain quite alive. Here are some examples:

- A new SDR GNSS book, (Borre et al., 2022), extending SoftGPS functionality to several frequencies, GNSS and architectures, can be considered as the successor of (Borre et al., 2007). A main building block of this book is FGI-GSRx, described in the following section, but the book also includes other Matlab receivers. In particular, DF-GSRx (Dual-Frequency GNSS Software Receiver), developed by Borre’s PhD student P. Bolla, is a dual-frequency GPS L1/L5 receiver that includes dual-frequency acquisition techniques, measurements combination (iono-free in particular) and positioning. The book also includes a GPS L1 C/A snapshot receiver developed by Borre’s former PhD student I. Fernandez-Hernandez, more modest than that described later in Section 3.8 but simple and quick to execute and therefore possibly useful for educational purposes.

- The Easy Suite libraries (Borre, 2003–2009), still publicly available and used, provide an excellent educational tool to dive into basic functions of GNSS receivers, such as calculating satellite positions from the ephemerides, datum conversions, or computing the receiver position and its accuracy in multiple ways (least squares, Kalman filter, carrier phase ambiguity resolution, etc.)

- (Bernabeu et al., 2021), as above mentioned, provides a collection of open source SDRs developed at University of Colorado Boulder and based on SoftGNSS.

- (Zhang, 2022) provides a repository with adaptations of SoftGNSS for different front-ends.

3.4 | Finnish Geospatial Research Institute’s Multi-GNSS Software Receiver

The software receiver developed by Finnish Geospatial Research Institute (FGI) is famously known as the FGI-GSRx (FGI’s GNSS Software Receiver). The development of the FGI-GNSS Software Receiver (GSRx) software receiver started in 2012 from the open source GNSS software receiver released in 2007 by Prof. Borre and his colleagues (Borre et al., 2007). The software receiver was able to track two IOV (In-Orbit Validation) satellites called GIOVE A and GIOVE B from the European GNSS system Galileo. Since then, the researchers at FGI have been continuously developing new capabilities to the software receiver with the inclusion of Galileo in 2013 (Söderholm et al., 2016), the Chinese satellite navigation system BeiDou in early 2014 (M. Bhuiyan et al., 2015), the Indian regional satellite navigation System NavIC in late 2014 (Thombre et al., 2015), and the Russian satellite navigation system GLONASS in 2015 (Honkala, 2016).

The FGI-GSRx software receiver has been extensively used as a research platform for the last one decade in different national and international research and development projects to develop, test and validate novel receiver processing algorithms for robust, resilient and precise position navigation and timing (PNT). At present, the FGI-GSRx can process GNSS signals from multiple constellations, including GPS, Galileo, BeiDou, GLONASS, and NavIC. The software receiver is intended to process raw IF signals in post-processing. The processing chain of the software receiver consists of GNSS signal acquisition, code and carrier tracking, decoding the navigation message, pseudorange estimation, and PVT estimation. The software architecture is built in such a way that any new algorithm can be developed and tested at any stage in the receiver processing chain without requiring significant changes to the original codes. FGI-GSRx provides a unique and easy-to-use platform not only for research and development, but also for whoever is interested in learning about GNSS receivers. Some of the main features of FGI-GSRx are listed in Tab. 5.

The software receiver was released as open source in February 2022 (FGI, 2022). FGI-GSRx receiver was also accompanied by the book ‘GNSS Software Receivers’, a next edition of one of the fundamental GNSS textbooks, published in 2022 by Cambridge University Press (Borre et al., 2022). The book systematically introduced the software receiver processing functionalities.
### TABLE 5 Main features of FGI-GSRx

<table>
<thead>
<tr>
<th>Feature</th>
<th>Solution</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>Windows 10</td>
<td>Compiles in Windows 10 environment. The software receiver should run in other OS which can host MATLAB or OCTAVE.</td>
</tr>
<tr>
<td>Programming environment</td>
<td>MATLAB</td>
<td>Executes in MATLAB 2019 or any other later version. The software receiver can be also executed in OCTAVE.</td>
</tr>
<tr>
<td>IF sample file input source</td>
<td>ION SDR Standard</td>
<td>Read input data files following ION SDR Standard.</td>
</tr>
<tr>
<td>Processing mode</td>
<td>Only operate as post-processing GNSS receiver</td>
<td>It can read raw IF data for a complete receiver processing, or it can load previously saved acquisition and/or tracking data in order to skip acquisition and/or tracking operation to be able to process navigation solution depending on parameters set in the user configuration file.</td>
</tr>
<tr>
<td>Supported GNSS</td>
<td>GPS L1, Galileo E1, BeiDou B1, GLONASS L1, NavIC L5</td>
<td>Open source FGI-GSRx only supports single frequency multi-GNSS processing.</td>
</tr>
<tr>
<td>Acquisition</td>
<td>FFT-based signal acquisition</td>
<td>Sophisticated research specific implementation for high sensitive acquisition is not published as open source.</td>
</tr>
<tr>
<td>Tracking</td>
<td>Table-based three-stage tracking</td>
<td>Based on the tracking status of each individual satellite, the software receiver switches among three stages: i) PULL IN, ii) COARSE TRACKING and iii) FINE TRACKING. Users can select SNR or elevation cut-off mask in order to decide on the satellites that contribute to the position computation.</td>
</tr>
<tr>
<td>Navigation</td>
<td>Traditional Least Square (LS)</td>
<td></td>
</tr>
</tbody>
</table>

with experimental results for GPS L1 C/A signal in Section 2 (“GPS L1 C/A Receiver Processing”, 2022), GLONASS L1OF signal in Section 3 (“GLONASS L1OF Receiver Processing”, 2022), Galileo E1 OS signal in Section 4 (“Galileo E1 Receiver Processing”, 2022), BeiDou B1I signal in Section 5 (“BeiDou B1I Receiver Processing”, 2022), NavIC L5 signal in Section 6, (“NavIC L5 Receiver Processing”, 2022) and a single frequency multi-constellation solution with three GNSS signals in Section 7 (“A Multi-GNSS Software Receiver”, 2022). The readers can easily follow the fundamental receiver processing chain for each individual GNSS signal with the distinctive changes among those signals discussed and highlighted with figures. One of the noteworthy contributions of the book was the integration method of several GNSS signals to form a single-frequency multi-GNSS PVT solution presented in Section 7.

The FGI-GSRx software receiver can be utilized in universities and other research institutes as a tool for training graduate level students and early-stage researchers for getting hands-on experience on GNSS receiver development. It can also be utilized in the vast GNSS industry as a benchmark software defined receiver implementation. The software receiver is already being used in the ‘GNSS Technologies’ course offered widely in Finland - at the University of Vaasa, Tampere University, Aalto University and the Finnish Institute of Technology.

### 3.5 GNSS-SDR, an Open-Source Software-Defined GNSS Receiver

The software receiver developed by the Centre Tecnològic de Telecomunicacions de Catalunya (CTTC), uncreatively named GNSS-SDR (but not related to the ION SDR Standard), is another example of a multi-band, multi-system receiver. It has been
TABLE 6 Main features of GNSS-SDR

<table>
<thead>
<tr>
<th>Feature</th>
<th>Solution</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>GNU/Linux, macOS, Windows OS through WSL.</td>
<td>Included as a software package in Debian and Ubuntu, and in Macports for macOS. Tested on ArchLinux, CentOS 7, Fedora, OpenSUSE, Rocky Linux. Software linters are automatically run at each code change to ensure meeting high-quality coding standards. It can work in real-time using a wide assortment of commercial RF front-ends, and in post-processing mode with a number of file formats (including input files produced by the ION SDR Standard conversion tools).</td>
</tr>
<tr>
<td>Programming environment</td>
<td>C++</td>
<td>The modular design allows for easy inclusion of new signals.</td>
</tr>
<tr>
<td>Processing mode</td>
<td>Real-Time and Post-Processing.</td>
<td></td>
</tr>
<tr>
<td>Supported GNSS</td>
<td>GPS L1, L2C, L5; Galileo E1, E5a, E5b, E6; Glonass L1 CA, L2 CA; Beidou B1, B3.</td>
<td>The modular design allows for easy inclusion of new signals.</td>
</tr>
<tr>
<td>Acquisition</td>
<td>FFT-based signal acquisition.</td>
<td>A-GNSS capabilities to accelerate the Time To First Fix.</td>
</tr>
<tr>
<td>Tracking</td>
<td>Multicorrelator-based Data and Pilot signal tracking.</td>
<td>Customizable DLL, PLL, FLL. High-dynamics capabilities. SIMD-accelerated both in i686 and ARM CPUs (see Fernández-Prades et al. (2016a)).</td>
</tr>
<tr>
<td>Navigation</td>
<td>Traditional Least Square (LS), code and carrier-based positioning modes.</td>
<td>Positioning engine based on RTKLIB implementation (Takasu &amp; Yasuda 2009). All possible supported GNSS signals combinations are allowed.</td>
</tr>
</tbody>
</table>

constantly evolving since 2010, keeping pace with the newest GNSS algorithms and signals over more than a decade. It originated as a by-product of a CTTC research staff initiative, with the aim of providing a collaborative framework with other researchers seeking to accelerate research and development of software-defined GNSS receiver technology. The receiver particularly focuses on baseband signal processing, although it has the ability to run a navigation engine (refer to Tab. 6). The early stages of development baked slowly under a personal side-project scheme, with no funding, but with the purely exploratory objective of designing an optimal architecture specifically suitable for GNSS signal processing, where concepts such as testability, extensibility, reusability, scalability, maintainability, portability, adaptability to new non-standard requirements, and adoption of Computer Science best practices considered from scratch.

Its first popularity peak came on August 2012, with the reporting of the usage for GNSS of extremely cheap (about $25) DVB-T receivers based on the Taiwan’s Realtek RTL2832U chipset, sold in form of USB dongles that allow users to watch over-the-air DVB-T European broadcast television on their personal computers. Normally, those devices send partially-decoded MPEG transport frames over the USB, but exploiting an undocumented mode of operation of the demodulator chip, the user was able to obtain raw I&Q samples, stream them through USB to a personal computer and then apply the GNSS-SDR software processing, turning the DVB-T receiver into a GNSS receiver and delivering position in real-time (see Fernández-Prades et al. (2013)). On a parallel development, in November 2013, the European Space Agency acknowledged GNSS-SDR as one of the first 50 users worldwide to achieve a successful Galileo position fix.

The project gained momentum and maturity over the years, and today it enjoys a solid and valuable user base continuously providing feedback, enhancements, and new features. Current versions are included in major GNU/Linux distributions, such as Debian and Ubuntu, and in Macports for Apple’s macOS. The software package has been used in several public and private-funded research projects (including EUSPA, European Space Agency (ESA), NSF and NASA activities, as well as in educational programs such as Google Summer of Code), and it has been reportedly used for research purposes worldwide. The authors opened
a discussion of quality metrics and key performance indicators for any generic software-defined receiver (Fernández–Prades et al. 2016b), extended online version available at https://gnss-sdr.org/design-forces/ and proposed the concept of continuous reproducibility in GNSS signal processing (Fernández–Prades et al. 2018). The full project and source code documentation can be found online at https://gnss-sdr.org, a website with over 5000 unique visitors per month, which contributes to raising awareness on GNSS technology. The website content is also on a GitHub repository at https://github.com/gnss-sdr/geniuss-place, hence undergoing public scrutiny. The project is also well-connected to its software ecosystem and existing SDR platforms. It builds on a wide range of GNU/Linux distributions and versions (from most recent releases to those released in 2014), and it provides a Yocto / Openembedded layer, which allows its portability to a wide range of embedded platforms (see Fernández–Prades 2022).

The software produces standard outputs for observables and navigation data (RINEX files and RTCM-104 v3.2 messages as defined by the Networked Transport of RTCM via Internet Protocol, NTRIP), as well as position fixes in application-specific messages (e.g., NMEA 0183), a variety of GIS-oriented file formats (KML, GeoJSON, GPX), and custom binary outputs that allow the observability of internal signal processing sub-products.

3.6 | AutoNav SDR

The AutoNav SDR is a MATLAB-based multi-GNSS and multi-frequency software receiver that was developed by the Autonomous Navigation Laboratory of Inha University, South Korea (Song et al. 2021). Its main features are arranged in Tab. 7. The critical point considered in the design phase of this SDR is the maximization of reconfigurability. Since South Korea is developing its own satellite navigation system, KPS which is targeted to operate from 2035 as reported by Ministry of Science and ICT of Korea (2021), a flexible receiver that can process not yet existent signals is highly required. The AutoNav SDR is profoundly designed to provide full reconfigurability in terms of target signal combinations and signal characteristics, especially for easy addition of the new signal proposals. To do so, a basic framework of software receiver was designed with a well-designed processing functional architecture and data structure in consideration of the expandability of the signals and then applied to realize an SDR for GPS L1 C/A code signal as a first realization example by reconfiguring a configuration file via a GUI. Then, different signals of the other constellations (GLONASS, Galileo, BeiDou navigation satellite system (BDS), Quasi-Zenith Satellite System (QZSS), NavIC) and frequencies (L1, L2, L5) were added quickly by utilizing this expandability. In this way, KPS signal candidates can be easily added to the SDR to evaluate and compare the performance of each candidate in the signal design phase. Similarly, a reconfigurable GNSS simulator was developed at the same time with the same idea. This is a MATLAB-based IF level GNSS/KPS simulator which can be ideally suited to test the navigation performance of any GNSS signals as well as new KPS signals by reconfiguring signal design parameters via a GUI.

Although the AutoNav SDR is targeted for post-processing only, the original correlation operation in MATLAB with variables of double precision was too slow at the beginning of its design phase. So, two simple accelerations were applied to the SDR: a GPU-based acquisition module and a MEX correlator for tracking. The GPU-based signal acquisition module was implemented in a very simple way using the Parallel Computing Toolbox of MATLAB. If the GPU is usable, local variables for the correlation (i.e., code and carrier replicas) are generated in the GPU memory using the gpuArray function. Then, FFT and inverse fast Fourier transform (IFFT), and correlations are performed in the GPU automatically. Finally, the correlation results are extracted using a gather function. With this simple approach, it has approximately 2.12 times faster execution time compared to the general CPU-based acquisition, without the relatively complex development using CUDA.

Since the most time-consuming process of the receiver is the correlator in the signal tracking, a MEX function is employed to reduce the computational burden. The MEX function connects the MATLAB environment to the external function written in C/C++ language with an appropriate wrapper function, so the user can call it within MATLAB. The MEX correlator was written in the standard C language and uses integer-based variables. The SDR pre-generates the code and carrier replica tables at the initialization process with resolutions of 18 bits and 8 bits, respectively. The code and carrier NCOs have a resolution of 32 bits, so the indices of the tables for current code and carrier replica generation are calculated using bit shift operations of 14 bits and 24 bits, respectively. With these implementations, the overall execution time became much faster (approximately 5 times) than the original double precision-based code, but it still cannot operate in real-time. Currently, Inha University is developing the FPGA-based real-time GNSS receiver that only the correlator would be substituted by the FPGA board at the original AutoNav SDR.

To further enhance the flexibility, the AutoNav SDR also provides APIs at each part of the signal processing chain (such as ring buffer, acquisition, tracking, navigation message extraction, position calculation, etc.). The API design was influenced by
### Table 7: Main features of AutoNav SDR

<table>
<thead>
<tr>
<th>Feature</th>
<th>Solution</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating system</td>
<td>Windows</td>
<td></td>
</tr>
<tr>
<td>Programming environment</td>
<td>MATLAB and C</td>
<td>Free selection of signal combination</td>
</tr>
<tr>
<td>Processing mode</td>
<td>Post-processing</td>
<td></td>
</tr>
<tr>
<td>Supported GNSS</td>
<td>GPS (L1 C/A, L2C, L5), GLONASS L1, Galileo (E1, E5a, E5b), BDS (B1I, B1C, B2a), QZSS (L1 C/A, L1C, L2C, L5), NavIC L5</td>
<td></td>
</tr>
<tr>
<td>Acquisition</td>
<td>GPU-based acquisition</td>
<td>Simple implementation using Parallel Computing Toolbox of MATLAB</td>
</tr>
<tr>
<td>Tracking</td>
<td>MEX correlator</td>
<td>18/8 bits code/carrier replica tables, 32 bits code/carrier numerically controlled oscillators (NCOs), bit shift operations</td>
</tr>
<tr>
<td>Further features</td>
<td>API, easy addition of new signals, RINEX observation logging, Radio Frequency Interference (RFI) mitigation based on pulse blanking, direct state tracking Kalman filter</td>
<td></td>
</tr>
</tbody>
</table>

the ipexSR of [Stöber et al. (2010)] and was implemented similarly using the dynamic link library (DLL). Since MATLAB can load a library from DLL and call a function within the library, the API concept of the C/C++-based software can also be used analogously in the MATLAB environment. If the SDR is converted to an executable file (.exe) and provided to a user, the user can freely modify functions or develop algorithms by generating the DLL, without the need for the whole source codes.

### 3.7 PyChips

PyChips is a relatively new object-oriented satnav SDR that has been developed from scratch since 2018. It is based on the experience gained from two previous implementations, namely the MATLAB SDR that was distributed with Wideband TRIGR (see Section 5) and the ChameleonChips GNSS SDR Toolbox for MATLAB [Gunawardena, 2014].

One of the key promises of SDRs is their flexibility, and hence its utility as an education and research tool. In the satnav context, various publicly available SDRs can be used to teach basic courses on satnav systems, signal processing, and receiver design. However, there is an implicit assumption that students have the relevant programming language skills for that particular SDR. Students are expected to understand the inner workings of the SDR in detail, and, more importantly, to make modifications to the code to add advanced capabilities and/or revisions as part of their graduate research projects. While somewhat valid, this programming language proficiency assumption may not always hold true. Further, given the situation, it may be far more efficient and beneficial to have grad students make deeper progress on their research rather than spending time becoming programming language experts. PyChips was developed from the ground up to support this notion. A more detailed introduction to PyChips can be found in [Gunawardena, 2021]. Its main features are summarized in Tab. 8. It is implemented in Python with C++ bindings where performance is absolutely essential for reasonably fast execution.

The current version of PyChips supports the creation and definition of entire constellations of satellites with advanced next-generation signal structures, along with interference sources and channel effects. This simulation portion of PyChips (comprising of numerous source objects) synthesizes these signals at the sample level on to one or more sample streams that are grouped into objects called stream containers. A stream container is an abstraction of a satnav receiver’s antenna(s) and RF front-end subsystem. This could be multi-frequency, multi-element, with different sample rates and bandwidths, IF or baseband sampling architecture, and any and all combinations thereof. If live-sky signal processing is the use case, then one or more sampled SDR
<table>
<thead>
<tr>
<th>Feature</th>
<th>Solution</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>Windows x64</td>
<td>Due to pre-compiled C/C++ bindings that currently use the Windows API for file reading and threading. Linux support under development.</td>
</tr>
<tr>
<td>Programming environment</td>
<td>Signal and ARchitecture</td>
<td>SDR entirely specified using JSON-based SARDL. Assembles pre-built configurable Python and C/C++ objects at runtime according to user SARDL specification.</td>
</tr>
<tr>
<td></td>
<td>Description Language</td>
<td></td>
</tr>
<tr>
<td>IF sample file input source</td>
<td>ION SDR Standard</td>
<td>Parses ION metadata hierarchy to select the appropriate decoder kernel written in C++. Sample decoding is split across multiple threads using a data parallel architecture</td>
</tr>
<tr>
<td>Real-time sample input</td>
<td>Not currently supported</td>
<td></td>
</tr>
<tr>
<td>Additional sensors</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Supported GNSS</td>
<td>Supports all civilian satnav</td>
<td>Spreading codes defined as memory codes. Code replicas specified as an assembly of sequence objects (static, subcarrier, overlay, mux, ..., see Gunawardena (2021))</td>
</tr>
<tr>
<td></td>
<td>signals (GPS, GLONASS,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Galileo, BeiDou, QZSS,</td>
<td></td>
</tr>
<tr>
<td></td>
<td>NavIC, SBAS)</td>
<td></td>
</tr>
<tr>
<td>Acquisition</td>
<td>FFT-based generic acquisition engine with configurable coherent and non-coherent integration settings</td>
<td>Auto-detects and implements circular vs. non-circular frequency-domain correlation based on code length</td>
</tr>
<tr>
<td>Tracking</td>
<td>Generic tracking module</td>
<td>Employs split-sum correlation Gunawardena &amp; van Graas (2006). Always operates on 1-millisecond block of samples and retires current block before operating on next block (no sample shifting to align with SV time-of-transmission). Direct initialization of tracking objects configured for other signals from same SV (e.g. GPS L1 C/A to L1C, L2C, and L5)</td>
</tr>
<tr>
<td></td>
<td>assembled from configurable functional blocks (e.g. carrier-wipeoff, code replica, correlator, gearbox, accumulator, ...) and a generic controller object – all defined in SARDL</td>
<td></td>
</tr>
<tr>
<td>Measurement output</td>
<td>Yes</td>
<td>CSV format</td>
</tr>
<tr>
<td>Availability</td>
<td>Versions distributed at ION conference tutorials</td>
<td>Versions used at ION tutorials</td>
</tr>
</tbody>
</table>

Data files can be specified to instantiate a stream container object that is functionally identical and imperceptible from a simulated one. PyChips uses the ION SDR Standard to determine the appropriate C/C++ decoder/unpacker/re-quantizer kernel to use for reading and parsing these SDR files.

The sample streams contained in a PyChips stream container are processed using numerous sink objects. Currently, implemented examples include virtual oscilloscopes and spectrum analyzers, as well as acquisition engines and signal tracking modules.

The unique feature of PyChips is that, all of the functionality described above is defined/specification using a draft SDR language called Signal and ARchitecture Description Language (SARDL). SARDL is implemented as a grouping of JavaScript Object Notation (JSON) files. Current and next-generation advanced satnav signal structures and the receiver architectures to process them are constructed by assembling together pre-built low-level functional blocks. For example, as described in Gunawardena (2021), the user can build receiver tracking modules to process GPS L1C TMBOC(6, 1, 4/33) and Galileo E10S CBOC(6, 1, 1/11) MBOC signals as simple BOC(1, 1) signals to model a low-cost low-power mass market receiver, or a high-end survey-grade receiver taking full advantage of these ‘dual personality’ signals.
TABLE 9 Main features of UAB Snapshot GNSS Receiver

<table>
<thead>
<tr>
<th>Feature</th>
<th>Solution</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating system</td>
<td>Any supported by MATLAB</td>
<td>MATLAB version 6.0 (R12, 2000) or higher.</td>
</tr>
<tr>
<td>Programming environment</td>
<td>MATLAB</td>
<td></td>
</tr>
<tr>
<td>Processing mode</td>
<td>Post-processing</td>
<td></td>
</tr>
<tr>
<td>Supported GNSS</td>
<td>GPS (L1 C/A, L5), Galileo (E1C, E5a)</td>
<td></td>
</tr>
<tr>
<td>Acquisition</td>
<td>FFT-based signal acquisition</td>
<td>Implementing the double-FFT algorithm for both code correlation and bit synchronization. Long correlations can be implemented by non-coherently combining a set of coherent correlations. Assisted-GNSS (A-GNSS) is used to narrow the acquisition search space. Compatible with 3GPP RRLP-compliant XML data.</td>
</tr>
<tr>
<td>Tracking</td>
<td>None</td>
<td>No tracking is implemented because the receiver architecture is based on snapshot mode (i.e. acquisition-only).</td>
</tr>
<tr>
<td>Navigation</td>
<td>Weighted Least Squares (WLS)</td>
<td>Coarse-time navigation is implemented.</td>
</tr>
<tr>
<td>Further features</td>
<td>Implements near-far detection and interference detection.</td>
<td></td>
</tr>
</tbody>
</table>

Indeed, at this stage, the goal of the PyChips project is to hone SARDL with a vast number of diverse signal specifications, use cases, and applications – in order explore the concept of a ‘satnav signals and systems specification language.’ Today, the reference SDR that implements SARDL is written in Python and is therefore called PyChips. However, the ultimate goal of this effort is to contribute towards satnav SDR implementations that have the performance, power efficiency, and scalability of ASICs with the flexibility, reconfigurability, adaptability, and ease-of-use of software.

3.8 UAB Snapshot GNSS Software Receiver

The UAB snapshot GNSS software receiver (cf. Tab. 9) was originally developed as part of the research activities on indoor GNSS positioning that were carried out by the Signal Processing for Communications and Navigation (SPCOMNAV) group at Universitat Autònoma de Barcelona (UAB), back in 2007. At that time, the group was involved in one of the two parallel contracts that ESA awarded to assess the feasibility of indoor GNSS positioning, under the project named DINGPOS. The proposed strategy was to rely on a combination of technologies such as WiFi, ultra wideband (UWB), 2G/3G cellular networks and GNSS as discussed by Lopez-Salcedo et al. [2008]. As far as GNSS was concerned, UAB was in charge of developing the software implementation of a so-called high-sensitivity GNSS (HS-GNSS) receiver, which could be able to operate under the extremely-weak signal conditions experienced indoors. This involves working under 10 to 40 dB attenuation losses, which drive the effective carrier power to noise power spectral density, i.e. $C/N_0$, down to values where conventional GNSS receivers are not able to operate anymore.

The proposed HS-GNSS receiver implementation was based on a snapshot architecture where a batch of input samples were processed at a time to provide the user’s position. This approach is often referred to in the literature as ‘push-to-fix’ or 'acquisition-only’, since no tracking stage is actually implemented at the receiver. This means that the receiver operates in open-loop mode by providing at its output the observables obtained straightaway from the acquisition stage. The implementation of the HS-GNSS software receiver was strongly influenced by the work already initiated by Gonzalo Seco-Granados before joining UAB, during his period from 2002 to 2005 as technical staff at the European Space Research and Technology Center (ESTEC) of ESA in The Netherlands, where he was leading the activities concerning indoor GNSS and snapshot GNSS.
receivers. Actually, the core of the UAB snapshot GNSS receiver was inspired on the same concept of double-FFT acquisition already introduced by Jiménez-Baños et al. (2006). This algorithm uses two consecutive FFT operations for implementing the correlation of the received signal with the local code replica, and then the simultaneous estimation of the fine Doppler and bit synchronization. Interested readers on the double-FFT algorithm and on a detailed description of the UAB snapshot GNSS receiver implementation will find a comprehensive description written by Seco-Granados et al. (2012).

From a general perspective, the UAB snapshot GNSS software receiver implements a set of specific signal processing techniques that are tailored to the particular working conditions indoors. Nevertheless, the implementation is flexible and it does not prevent the receiver to be operated efficiently in other scenarios, such as outdoors. Regarding the indoor environment, the most important impairment to be counteracted is certainly the severe attenuation due to the propagation through building materials and other obstacles. Attenuation up to 40 dB can easily be experienced, thus requiring a specific action to recover as much of the lost power in order to still be able to detect GNSS satellites. Since it is the received energy what matters from a signal detection and estimation point of view, and energy is nothing but power times the observation time, the only way to compensate for an extremely weak received power is by increasing the observation time. This means processing a longer piece of received signal, which means implementing very long correlation integration times at the GNSS receiver, on the order of hundreds of milliseconds or even a few seconds. Unfortunately, increasing the correlation time is hindered by the presence of the navigation message data symbols, residual Doppler errors and clock instabilities. So the approach adopted in practice by most snapshot GNSS receivers, particularly those intended for high-sensitivity applications, is to split a long correlation into pieces of shorter, but long-enough coherent correlations, whose outputs are then noncoherently accumulated. This combination of coherent and noncoherent correlation has proven to be successful in increasing the receiver sensitivity and thus still be able to detect a few GNSS satellites indoors. Actually, an interesting discussion on how important having long-enough coherent integrations was discussed by Pany et al. (2009).

The correlation between the received signal and the local replica is therefore the most important operation of a snapshot GNSS receiver. The reason is that with such correlation, the most accurate code delay and Doppler observables need to be estimated. This is because no tracking stage is implemented, and thus there will be no chance to further refine these observables in subsequent stages of the receiver. It is for this reason that the correlation must be implemented in the most optimal way, taking into account subtle details that might be ignored in conventional GNSS receiver implementations. Such optimality is actually brought by the double-FFT algorithm implemented in the UAB snapshot GNSS receiver, which implements the optimal joint estimation of the code-delay and fine Doppler over a long period of time, where potentially sign transitions may occur due to the presence of data modulating symbols. Additional considerations such as how to handle a non-integer number of samples when performing the FFT, the interpolation between consecutive correlation peaks, the code-Doppler effect over a long correlation period, etc. can be found in Seco-Granados et al. (2012).

The code delay and Doppler estimates provided by the acquisition stage are then directly used by the navigation module to compute the user’s position. Such code-delay estimates are ambiguous at one code period because no absolute time reference is available, and therefore no other time-delay information can be provided but that contained with a PRN code period. This is because just a batch of received samples is processed, and thus no access to the transmission time encoded onto the navigation message is available in general. As a result, the user’s position needs to be computed without such time reference, which becomes a very specific feature of snapshot GNSS receivers. This problem can be solved thanks to what is known as coarse-time navigation, where the conventional navigation equations are augmented to include an additional unknown that represents the missing absolute time reference. The interested readers will find in Van Diggelen (2009) Ch.4 an excellent description of this method.

Since its development in 2008, the UAB snapshot GNSS receiver has been a key tool for many research activities at the SPCOMNAV group. This software has been used for instance, to characterize the multipath propagation indoors (López-Salcedo et al., 2009), to assess the feasibility of using GNSS receivers in missions to the Moon, where the weak-signal problem is very similar to the indoor one (Manzano-Jurado et al., 2014), to test near-far mitigation techniques that may appear in indoor/Space applications (Locubiche-Serra et al., 2016), to assess the impact of phase noise (Gómez-Casco et al., 2016), and to provide GNSS positioning to internet of things (IOT) sensors in smart cities (Minetto et al., 2020) by means of a cloud-based implementation of the UAB snapshot GNSS receiver that was developed from 2016 to 2018.

The migration of the UAB snapshot receiver into a cloud-based implementation was certainly a major milestone that attracted the interest of the community and opened the door for totally new applications and use cases. The interest in cloud GNSS positioning was motivated by the fact that GNSS software receivers were running at that time in local computers next to the user who collected the samples to be processed. However, with the advent and widespread deployment of cloud computing platforms such as Amazon Web Services (AWS), Microsoft Azure and Google Cloud, such local computers could actually be
placed anywhere, and remote access could be granted to upload and process GNSS samples in a remote server in a scalable manner. Furthermore, this approach fitted pretty well with a snapshot GNSS receiver implementation, where a batch of samples could be sent to a remote server where the user’s position would then be computed using the same tools as in any other snapshot GNSS receiver. That is, using A-GNSS for reducing the acquisition search space, making extensive use of FFT operations and computing the user’s position by means of coarse-time navigation techniques.

This was the idea behind the so-called ‘cloudGNSSRx’, the cloud-based implementation of the UAB snapshot GNSS receiver as described in [SPCOMNAV] (2019). The architecture was based on a dockerized compilation of the Matlab source code implementing the UAB snapshot GNSS receiver. Then a system of job queues, schedulers and load balancers were built on AWS to automate and scale the remote execution of the receiver, and an API was developed for machine-to-machine communication, facilitating the provision of GNSS positioning to small IOT sensors [Lucas-Sabola et al. (2016)]. In this way, IOT sensors requiring GNSS positioning were able to offload most of the computational load to a remote server, thus significantly reducing the power consumption and thus extending their battery lifetime.

Low-power GNSS positioning is actually one of the main applications of cloud GNSS software receivers, since for snapshots shorter than a few tens of milliseconds, the energy spent in sending the GNSS samples to the cloud pays off for the significant energy that is saved at the user’s terminal for not processing such samples, and doing it at the cloud instead [Lucas-Sabola et al., 2017]. This feature was actually acknowledged by the former GSA, now the European Union Agency for the Space Programme (EUSPA), who identified the UAB cloud GNSS receiver as one of the promising technologies for the future adoption of GNSS in the IOT domain [European Union Agency for the Space Programme, 2018]. The cloud GNSS software receiver developed by UAB was then licensed in 2019 to the startup company Loctio, who improved very significantly the initial prototype and made it a commercial product.

It is important to remark that apart from the low-power consumption use case, cloud GNSS software receivers can also be used to provide access to sophisticated signal processing techniques that cannot be implemented in conventional receivers. For instance, advanced signal monitoring techniques, spoofing detection or authenticated/certified positioning, the latter being reported by Rügamer et al. (2016). There is therefore a brilliant future ahead for cloud GNSS software receivers with many exciting new applications still to come.

3.9 The NGene Family of Receivers at Politecnico di Torino and LINKS

The development of NGene, a GNSS software receiver, originated at Politecnico di Torino and LINKS Foundation in the early 2000s. At that time, the Navigation Signal Analysis and Simulation Group (NavSAS) was already engaged in software implementation of various sections of GNSS baseband processing. This endeavor capitalized on the group’s extensive expertise in digital signal processing and specifically in simulating complex communication systems.

The initial focus was optimizing the acquisition and tracking stages, both as post-processing tools and as core processing units on programmable hardware. In 2005, under regional funding, the research team, partially affiliated with the Istituto Superiore Mario Boella (now part of the LINKS Foundation), commenced the development of a fully software-based, real-time GNSS receiver for GPS and forthcoming Galileo signals.

The outcome of the work was the NGene software receiver, as documented by Molino et al. (2009). NGene demonstrated real-time processing capabilities for GPS, Galileo, and EGNS signal components transmitted on the L1/E1 band. Prior to processing, the signals were subjected to IF downconversion and digitalization by an external analog front-end device. Communication between the front-end device and the software receiver occurred via a USB connection. The hardware part of the receiving chain consisted solely of the antenna, and its low noise amplifier (LNA), the A/D converter with front-end filtering, with all other components being software-based. This fundamental architecture has laid the groundwork for subsequent enhancements and has been the essential building block of the NGene receiver family.

NGene, thanks to the reconfigurable and modular structure, has long served as the primary tool for in-lab analysis, development, and prototyping of signal processing algorithms and architectures. Thanks to its flexible implementation, NGene was adapted to process the Galileo In-Orbit Validation signals (GIOVE-A) and subsequently to process the first Galileo signals immediately upon their availability, as detailed in Margaria et al. (2012). As a result, the research team was among the first worldwide to achieve a position fix using the initial four Galileo satellites. Over time, the software receiver continued to evolve and was tailored to address diverse applications, leveraging the advantages of software radio implementation (see Tab. 10).

Today, the NGene family offers configurable support for various RF-to-IF front-ends, which connect to the software processor via USB, meeting the requirements of numerous activities and projects. A simplified, low-complexity version was developed to
TABLE 10 Main features of NGene receiver family

<table>
<thead>
<tr>
<th>Feature</th>
<th>Solution</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>GNU/Linux-based</td>
<td>Since based on standard libraries, it can run on Windows too</td>
</tr>
<tr>
<td>Programming environment</td>
<td>ANSI C and assembly (Intel x86 and ARM SIMD instructions)</td>
<td>Eclipse IDE and GNU Compiler Collection (GCC) compiler</td>
</tr>
<tr>
<td>IF sample file input source</td>
<td>Binary file</td>
<td>It can work in real-time from USB-based RF front-ends and in post-processing mode with binary file formats.</td>
</tr>
<tr>
<td>Processing mode</td>
<td>Real Time and Post-processing</td>
<td></td>
</tr>
<tr>
<td>Additional sensors</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Supported GNSS</td>
<td>GPS L1 C/A, Galileo E1</td>
<td></td>
</tr>
<tr>
<td>Acquisition</td>
<td>FFT-based algorithm</td>
<td></td>
</tr>
<tr>
<td>Tracking</td>
<td>Multi-Correlator based Data tracking loop</td>
<td></td>
</tr>
<tr>
<td>Measurement output</td>
<td>Yes</td>
<td>Acquisition, tracking and PVT results available as binary/text log files</td>
</tr>
<tr>
<td>Availability</td>
<td>Restricted</td>
<td>Licensing of a public release currently under discussion</td>
</tr>
<tr>
<td>Further features</td>
<td>Scintillation monitoring, Interference detection, Galileo OSNMA authentication</td>
<td>Specific modified versions for research purposes</td>
</tr>
</tbody>
</table>

enable GNSS positioning capabilities on ARM-based embedded processors (Troglia Gamba, Nicola, & Falletti, 2015). Additional branches of the software were adapted for GNSS-R receiver deployment in reflectometry tests (Troglia Gamba, Marucco, et al., 2015), evaluation of anti-jamming algorithms, detection of non-standard code transmission and its effects on Galileo positioning (Dovis et al., 2017), as well as being the tool for the study of the 2019 Galileo outage event (Dovis et al., 2019). One of the latest branches of the NGene family encompasses algorithms for authenticating Galileo messages using the OSNMA, as described in Nicola et al. (2022); Troglia Gamba et al. (2020a). Furthermore, a set of functions is being developed to support future GPS Chimera authentication service processing (Troglia Gamba et al., 2020b).

3.10 The MATRIX SDR for Navigation with Signals of Opportunity

MATRIX (Multichannel Adaptive TRansceiver Information eXtractor) is a state-of-the-art cognitive SDR, developed at Kassas’ Autonomous Systems Perception, Intelligence, and Navigation (ASPIN) Laboratory, for navigation with terrestrial and space-based SOPs (Kassas et al., 2020). MATRIX continuously searches for opportune signals from which it draws navigation and timing information, employing signal characterization on-the-fly as necessary. MATRIX could produce a navigation solution in a standalone fashion (Shamaei & Kassas, 2021a) or by fusing SOPs with sensors (e.g., IMU (Morales & Kassas, 2021), LiDAR (Maaref et al., 2019), etc.), digital maps (Maaref & Kassas, 2020), and/or other signals (e.g., GNSS) (Kassas et al., 2017). Figure 1 shows MATRIX’s architecture and Tab. 11 lists the main features.

On one hand, MATRIX has achieved the most accurate navigation results to-date in the published literature with cellular SOPs (3G CDMA, 4G LTE, and 5G NR), achieving meter-level navigation indoors (Abdallah & Kassas, 2021) and on ground vehicles (Maaref & Kassas, 2022) and submeter-level navigation on unmanned aerial vehicles (Khalife & Kassas, 2022). In addition, MATRIX’s efficacy has been demonstrated in a real-world GPS-denied environment (Kassas, Khalife, Abdallah, & Lee, 2022), achieving a position root-mean squared error of 2.6 m with 7 cellular LTE eNodeBs over a trajectory of 5 km (one of which was more than 25 km away), during which GPS was intentionally jammed (Abdallah et al., 2022). MATRIX has also achieved remarkable results on high-altitude aircraft, where it was able to acquire and track cellular 3G CDMA and 4G LTE signals at
### TABLE 11 Main features of MATRIX

<table>
<thead>
<tr>
<th>Feature</th>
<th>Solution</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating System</td>
<td>Linux, Windows</td>
<td>Some SOP modules supports real-time processing</td>
</tr>
<tr>
<td>Programming environment</td>
<td>C++, MATLAB, LabVIEW</td>
<td></td>
</tr>
<tr>
<td>IF sample file input source</td>
<td>Binary file</td>
<td></td>
</tr>
<tr>
<td>Real-time sample input</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Additional sensors</td>
<td>IMU, GNSS</td>
<td></td>
</tr>
<tr>
<td>Supported GNSS</td>
<td>GPS L1 C/A</td>
<td></td>
</tr>
<tr>
<td>Acquisition</td>
<td>FFT-based signal acquisition</td>
<td></td>
</tr>
<tr>
<td>Tracking</td>
<td>DLL, PLL, FLL, Kalman filter</td>
<td>Different tracking loops per SOP module</td>
</tr>
<tr>
<td>Measurement output</td>
<td>Yes</td>
<td>Acquisition, tracking, and PVT results available as text/CSV files and via GUI</td>
</tr>
<tr>
<td>Navigation</td>
<td>Weighted least squares; Kalman filter; Doppler, code- and carrier-based positioning modes</td>
<td>Licensing options available via The Ohio State University</td>
</tr>
<tr>
<td>Availability</td>
<td>Restricted</td>
<td></td>
</tr>
</tbody>
</table>

**FIGURE 1** MATRIX cognitive SDR architecture. The SDR consists of: (i) USRPs to collect different radio signals, (ii) various modules to produce navigation observables from different types of signals (e.g., cellular, LEO satellites, etc.), (iii) external sensors (e.g., IMU, LiDAR, GNSS receivers, etc.), whose measurements can be fused with the navigation observables produced by the signal modules, and (iv) navigation filter that fuses all measurements to produce a navigation solution.

altitudes as high as 23,000 ft above ground level and from cellular towers more than 100 km away (Kassas, Khalife, Abdallah, Lee, Jurado, et al., 2022). What is more, meter-level high-altitude aircraft navigation was demonstrated over aircraft trajectories exceeding 50 km, by fusing MATRIX’s cellular navigation observables with an altimeter (Kassas, Abdallah, et al., 2022).

On the other hand, MATRIX has achieved the first published results in the literature for exploiting unknown SpaceX’s Starlink LEO satellite signals for positioning, achieving a horizontal positioning error of 10 m with Doppler observables (Neinavae et al., 2021) and 7.7 m with carrier phase observables (Khalife et al., 2022). In addition, the first ground vehicle navigation results with multi-constellation LEO (Orbcomm, Iridium NEXT, and Starlink satellites) were achieved with MATRIX (Kassas et al., 2021), upon coupling its LEO navigation observables with an inertial navigation system (INS) in a tightly-coupled fashion through the simultaneous tracking and navigation (STAN) framework (Kassas et al., 2019).
3.11 Other Achievements with GNSS SDRs

Apart from the success stories of the previous subsections a number of other achievements have been accomplished with GNSS SDRs. They are listed in this subsection.

The first real-time GNSS/INS integration with an SDR was achieved by Gunawardena et al. (2004) and one of the first GNSS SDR implementation on a GPU was reported in Hobiger et al. (2010).

GNSS SDRs are known to achieve the highest possible sensitivity as different integration schemes or data wipe-off procedures can be performed in post-processing. This enables very long coherent integration times, which is beneficial for sensitivity or multipath mitigation as reported in Section 3.8. Characterization of the GPS transmit antenna pattern with a 30-second long coherent integration resulting in 0 dBHz sensitivity is discussed by Donaldson et al. (2020). The same sensitivity was achieved by 300 noncoherent integrations, each 1 second long for the purpose of indoor timing by iPosi Inc. (2015).

Graphical programming languages, such as LabVIEW and Simulink, are attractive choices for implementing SDRs, due to their flexibility, modularity, and upgradability. Moreover, since SDRs are conceptualized as block diagrams, graphical programming languages enable a one-to-one correspondence between the architectural conceptualization and software implementation (Hamza et al., 2009; Kassas et al., 2013).

The scope of SDRs was first extended to non-GNSS signals by McEllroy et al. (2006). SDRs became the implementation of choice in numerous studies aimed at exploiting SOPs for navigation purposes (Diouf et al., 2019; Kassas et al., 2017), such as (i) cellular 3G code division multiple access (CDMA) (Khalife et al., 2018; K. Pesyna et al., 2011; Yang & Soloviev, 2018), 4G long term evolution (LTE) (del Peral-Rosado et al., 2017; Ikhtiar, 2019; Kang et al., 2019; Shamaei & Kassas, 2018; Shamaei et al., 2018; Wang et al., 2022); (ii) 5G new radio (NR) (Abdallah & Kassas, 2022; del Peral-Rosado et al., 2022; Fokin & Volgushev, 2022; Lapin et al., 2022; Santana et al., 2021; Shamaei & Kassas, 2021b; Tang & Peng, 2022); (iii) AM/FM radio (Chen et al., 2020; McEllroy, 2006; Psaki & Slosman, 2022; Souli et al., 2021); (iv) digital television (Souli et al., 2020, 2022; Yang & Soloviev, 2020); and (v) LEO satellites (Farhangian et al., 2021; Farhangian & Landry, 2020; Nardin et al., 2021; Orabi et al., 2021; Pinell, 2021; Zhao et al., 2022).

Due to the enhanced analysis possibilities of GNSS SDR they proved to be very useful to understand ionospheric scintillation and the first dedicated SDRs are described by O'Hanlon et al. (2011); Peng & Morton (2011). The authors used a general purpose front-end being reconfigurable for multi-GNSS multi-band signals, and a custom dual-frequency front-end, respectively. The first system further evolved into an intelligent, scintillation event-driven data collection, as reported by Morton et al. (2015).

Commercialization of academic SDR developments was partly discussed in the previous sections. Also, a major receiver manufacturer provides GNSS SDRs, first starting with a timing receiver (Trimble Inc., 2005) and then moving to a flexible narrow-band receiver (Trimble Inc., 2017). Wide-band signals were later added, with some signal processing now done on an FPGA as reported in PR Newswire (2021). The most recent commercial activity can be found in LocusLock (2022) and builds upon the software described in the Section 3.1.

4 SDR FRONT-ENDS

As outlined before, a front-end is required to obtain digital samples for the SDR processing. The front-end’s tasks are to receive, filter, amplify, down-convert, and further digitize and quantize the analog RF signal entering the GNSS antenna. Many different types of front-ends were used for GNSS SDRs. Roughly, five different categories can be identified:

**discrete components:** using RF-connectable components like LNAs, filters or ADCs it is comparable easy to realize the function of a front-end and log IF or baseband samples. Those setups are easy to realize, but often bulky and sometimes prone to interference.

**commercial signal recorders:** several companies offer GNSS signal recorders to allow to record (and often to replay) one or more GNSS frequency band. Usually they do not implement a real-time connection to an SDR.

**generic non-GNSS front-ends:** SDR technology is used in many different fields of electrical engineering and front-ends covering a wide frequency are available. Their price ranges from a few Dollars (Fernández–Prades et al., 2013) to highly sophisticated multi-channel front-ends costing several ten-thousands of Dollars. The oscillator quality, bit-width or RF-filter characteristics is not always optimal for GNSS signal processing.
dedicated GNSS real-time front-ends: built for the purpose to realize a real-time GNSS SDR. A good example is described in Section 4.1. They are compact and built with discrete components.

ASICs: some RF-ASICs seem to target GNSS SDR use and evaluation kits allow streaming of IF samples, e.g. NTLAB, UAB (2022); RF Micro Devices, Inc., Greensboro (2006).

GNSS signals need a comparable high sampling rate of the front-end and when connected to a PC via a USB cable the transfer was not always reliable in the early years. Various optimizations and workarounds have been implemented like watermarking the IF sample stream and skipping lost sample packets as invented by Foerster & Pany (2013). With the advent of USB 3.0 or PCIe those solutions became obsolete.

In the following section, we describe Fraunhofer USB front-ends as an example of user needs, main features and general architectures of GNSS SDR front-ends. For a broader perspective of GNSS-compatible front-ends in the market, the interested reader can refer to (Borre et al., 2022, Ch.12).

4.1 Fraunhofer USB Front-ends

The scientific community, along with some industrial partners, required a multi-band solution for the upcoming civil multi-band signals in GPS and Galileo planning. In 2006, Fraunhofer IIS developed a front-end called the L125 Triband USB (see Fig. 2 a), which allowed recording of fixed frequencies of L1/E1, L2, and L5/E5a. This front-end had one antenna input and could record via two USB 2.0 connectors data streams with up to 40 MSPS sampling rate and a 2 or 4 bit ADC resolution, and one antenna input. However, increasing customer demands for portable and flexible solutions led to a complete redesign of the USB front-end concept. One major request was reconfigurability on the SDR front-end side. To meet these different requirements in one SDR front-end hardware, a new version of the USB front-end was developed that realizes the signal conditioning to an onboard FPGA enabling desired reconfigurability on the fly. This was necessary to allow for a single-band receiver with a low sampling rate for specific real-time SDR projects, as well as a wideband and multi-frequency front-end for other projects.

In 2012 Fraunhofer IIS (Rügamer et al., 2012) introduced the Flexiband multi-system, multi-band USB front-end depicted in Fig. 2 b. Within the last ten years, this front-end has been used and validated in numerous scientific and industrial projects. Furthermore, it has been commercialized and is distributed as the ‘MGSE REC’ product of TeleOrbit GmbH (2022).

A regular Flexiband unit consists of up to three analog reception boards, a carrier board with ADCs and FPGA, and a USB 3.0 interface board. A common antenna input port is supported, and separate front-end input signals for up to three antenna inputs. Three dual-channel ADCs sample the incoming signal with 81 MSPS and 8 bits I/Q. The raw data stream is received by an FPGA in which different digital operations like filtering, mixing, data rate, and bit-width reduction, as well as a digital automatic gain control (AGC) are applied. Finally, a single multiplexed data stream is formed together with a checksum. This multiplexed stream is sent via an USB 3.0 interface to the PC. Data rates of up to 1296 MBit/s or 162 MByte/s raw data stream are supported. The Flexband GUI software receives the raw multiplexed stream, checks its integrity, and demultiplexes it. The data stream can be either written to hard disk or sent to a customer application (e.g., a software receiver). The raw samples can be stored as a multiplexed data stream, in an 8 bit/sample format, or directly as a .mat file for MATLAB. In parallel, the ION Metadata *.sdrx is provided.

Due to its bandwidth, sampling rates, quantization, and multiplexing schema flexibility, the ION SDR Standard was a perfect fit to clearly and unambiguously define the configuration for the user. Therefore, right after the first conclusion of the ION SDR Standard, each binary raw-data output file of the Flexiband front-end was equipped with an ‘sdrx’ metadata file specifying the raw data format.

Finally, a replay variant of this Flexiband exists that reads in the raw IF samples on hard disk using the ION SDR Standard specification and replays the digital data as an analog RF output stream supporting multiple GNSS bands at the same time.

5 ION SDR STANDARD

The previous sections already made clear that data exchange between the various SDRs requires a certain level of standardization. The events that led up to the suggestion to develop what became the ION SDR Standard (also known as ION GNSS Metadata Standard, ION SDR Metadata Standard, GNSS SDR Sampled Data Metadata Standard or GNSS SDR Metadata Standard), can be traced back to circa 1999. Building upon the successful contributions made by Akos, the Ohio University Avionics
Engineering Center undertook several research projects leveraging GPS SDRs. One such project was called the GPS Anomalous Event Monitor (GAEM) (Snyder et al., 1999). This was sponsored by the FAA Technical Center and led by Prof. Frank van Graas. Commercial GPS receivers within prototype LAAS ground facilities were experiencing brief unexplained outages. GAEM kept a continuous 10-second history of IF samples in a circular memory buffer. When an outage occurred, GAEM was triggered to dump this buffer to disk and collect for a further 10 seconds. These sample files were then post-processed in MATLAB to determine the cause of the anomaly. Early versions of GAEM used commercial data collection cards and had numerous issues related to their proprietary drivers. Around 2001, Gunawardena developed a refined version of GAEM that was based on one of the earliest PCI-based dual-ADC-plus-FPGA development cards commercially available. It collected two GPS L1 data streams at 5 MSPS and 2 MHz bandwidth. This version of GAEM was fielded at three airports and operated continuously for over 3 years and helped to characterize numerous anomalous events (Gunawardena et al., 2009). This GAEM also supported a continuous collection mode, and was used for several research projects including the characterization of GPS multipath over water (Zhu & Van Graas, 2009) and GPS/IMU deep integration demonstrations in flight (Soloviev et al., 2004). For the latter, the 2 kHz raw data from a MEMS IMU were interleaved with the SDR samples thanks to the FPGA-based architecture that allowed for such custom capabilities.

Circa 2002, as these research projects progressed, the 2 MHz bandwidth limitation of GAEM became apparent. There was a pressing need to support emerging research opportunities related to GPS L5, as well as high fidelity GPS signal quality monitoring. A multi-band and higher-bandwidth (24 MHz) front-end and SDR data collection system was needed. There were only a handful of vendors selling such systems at the time, and it wasn’t clear if these would serve the purpose for satnav SDR application (sampling coherency concerns, etc.). However, by far, the >$350k price tag of these systems precluded any hope of purchasing them for university research. It was decided to develop this capability in-house. In 2003, a 2-channel L1/L5 front-end with 24 MHz bandwidth and 56.32 MSPS was developed (Gunawardena et al., Winter 2007-2008). It was based on connectorized RF components. The sampling and collection subsystems were carried over from GAEM.

The capabilities of the dual-frequency high-bandwidth system attracted interest from several universities, government research groups, as well as a defense contractor. To support these opportunities, the development of a new system known as Wideband Transform-domain Instrumentation GNSS Receiver (TRIGR) was completed in 2008 (Gunawardena & Van Graas, 2011). The front-end was miniaturized to a single-frequency custom PCB module. Up to 8 such modules (with the required frequency options) were combined with an 8-channel 12-bit ADC to create modular systems for various sponsors. The raw samples from the ADC are transferred to a PCIe FPGA card where the 8 streams are packed in various formats according to the user’s selection in a GUI. Supported formats range from any one stream at 1-bit sample depth, any 2 streams at 12 bits (sign extended to 16), to all 8 streams at 4 bits. The sustained data transfer rate from the PCIe FPGA card to the RAID storage array was limited to 240 MB/sec. As such, the appropriate format had to be selected to balance between the required capability and transfer rate. The generated file names embed a UTC timestamp as well as the packed stream order and sample depth.

The event-based data collection feature of GAEM needed to be incorporated into Wideband TRIGR. However, the >10× data rate meant that a 10-second circular buffer could not be easily implemented in RAM using 32-bit systems of the day. This issue was addressed by writing data as a sequence of smaller files, where a new file was spawned before the current file was closed – with some sample overlap for data integrity – a technique known as temporal splitting. A separate process was used to delete
older files from the RAID array to make room for new ones – unless an event was received – in which case the files surrounding the event were moved to a folder for post-processing.

With the myriad of sample packing formats available with Wideband TRIGR, along with the temporal splitting-based file generation scheme, it became clear that a machine-readable metadata file needed to be included with every collection. An XML schema was designed for this purpose.

Up until this time, apart from the FPGA-based real-time GPS receiver that was developed and used for certain projects, all SDR files generated by GAEM and Wideband TRIGR were post processed in MATLAB. As others have mentioned, this was exquisitely slow – especially for Wideband TRIGR data. To address this issue, as well as to support the rapid emergence of multi-band and multi-constellation sat nav signals, Gunawardena wrote and distributed a MATLAB SDR toolbox where correlation was performed in optimized C code and also leveraged multi-threading in a data parallel architecture. This toolbox, known as ChameleonChips, also read the XML metadata files produced by Wideband TRIGR to determine the appropriate sample unpacking kernel to use. This work was presented at ION GNSS+ 2013 in Nashville, TN (Gunawardena [2013]). During this presentation, it was suggested that the sat nav SDR community should adopt a metadata standard – similar to the one developed for Wideband TRIGR – in order to alleviate the numerous headaches associated with sharing such files. This was met with widespread support and enthusiasm. Longstanding ION members Phillip Ward, Jade Morton, and Michael Braasch helped to pitch this idea to the ION Executive Committee.

During the January 2014 Council Meeting in San Diego, ION approved the process for establishing a formal standard (Gunawardena et al., 2021). The ION GNSS SDR Metadata Working Group (WG) was formed in April 2014 with Thomas Pany and Gunawardena as co-chairs (James Curran was added later as a third co-chair). Membership represented academia, industry (including sat nav SDR product vendors as well as traditional sat nav equipment manufacturers), non-profit research entities, and government agencies spanning countries in Europe, America, Asia, and Australia. The working group developed the standard as well as associated normative software over a course of six years. With regard to the normative software, while many individuals contributed, initial development of the C++ object model was performed by Michael Mathews of Loctronix while James Curran wrote much of the functionality to decode packed samples based on the metadata specification. The draft standard was adopted as the formal ION SDR Standard in January 2020.

5.1 Use of the ION SDR Standard

Today the ION SDR Standard serves as a reference to describe IF formats and is for example useful for public tenders or if for some means an established format is needed. A number of SDRs do include the C++ libraries to read meta-data and IF samples.

The level of exchange of IF samples between research groups is to some extent limited and much less executed compared to e.g. exchange of RINEX files. This is of course related to the huge size of IF sample files and to the fact that for the majority of GNSS use cases, RINEX observation data or PVT exchange is sufficient. Furthermore, GNSS SDRs still tend to use mostly the same front-end and once the respective data format is known, there is obviously no need to describe it via the XML format. A disadvantage of the C++ routines is their generic design, which renders sample reading quite slow, as each sample is isolated via a number of for-loops from the input files. Clements et al. (2021) did propose an algorithm to automatically generate optimized code for sample reading for a given IF format, but this proposal did not yet manifest into a usable implementation.

5.2 ION SDR Standard Extension

Already during the standardization process a number of features for the standard were identified, that appear to be useful, but lack of resources did not allow including them in the formal standardization procedure. Those features are described in the App. II of (ION SDR Working Group, 2020). Within the ION-GNSS+ 2022 meeting in September, the following points have been discussed and will be included in App. II of the next - draft-version V1.1 of the ION SDR Standard:

5.2.1 Flexible bit layout

The ION SDR Standard defines a 'Lump' as the ordered containment of all samples occurring within an interval. The ordered containment is understood in a regular way holding the samples of the individual streams together. Clements et al. (2021) see this as a limitation, as highly efficient SDRs may use efficient bit-packing schemes to optimize data transfer over communication lines that need buffering. They identify a need to distribute the samples of different Streams in interleaved ways over the Lump. This interleaving cannot be described by the V1.0 of the ION SDR Standard. To overcome this limitation, the authors propose a
new but optional attribute for the Lump object, called 'Layout'. In case Layout is present, further information on the bit packing scheme needs to be provided, describing in an explicit way the type of each bit of a Lump. The authors make a detailed proposal for this new Lump layout following the structure of the existing standard. The proposal even includes more advanced bit use cases, like puncturing (e.g. explicit omitting of bits) and overwriting of bits by time markers.

5.2.2 | Refined sample rate/epoch definitions

Clements et al. (2021) note that the V1.0 of the ION SDR Standard makes implicit assumptions about the timing of the sampling process and staggered sampling cannot be described by it. Staggered sampling occurs if the sampling instants of different GNSS signals are delayed with respect to each other, and might be of use to increase observability of GNSS interference in a multi-antenna system. To overcome this limitation, the authors propose to add two new attributes for ‘Stream’ objects to shift the sampling epochs of different GNSS stream with respect to each other.

5.2.3 | JSON format for metadata files

Comment ID 22 of the initial Request for Comments (RFC1) makes a suggestion that the WG considers markup languages other than XML for metadata files, specifically JSON, YAML and TOML (Anonymous, 2017). In 2017, this comment was addressed by asserting that the XML format will be maintained for the time being, since normative software that parses XML had already been developed. However, the WG responded with the assurance that “other markup languages will be considered in the future based on community need and interest.”

As of the time of this writing, and with the experience gained from developing PyChips (which is a satnav SDR that is completely described using a draft signal/system specification language based on JSON, as described in Section 3.7), it is this author’s opinion that JSON may have some distinct advantages over XML for future applications and use cases. For example, JSON streaming is a methodology for transferring object-oriented data over communications protocols (Wikipedia, 2022) and is widely used in well-known applications such as Plotly (2022). Hence, streaming JSON could be one way to parse SDR sample streams whose formats are changing dynamically.

Figure 3 shows a notional listing for a JSON formatted metadata description for the Flexiband front-end XML metadata listing found in Gunawardena et al. (2021).

To maintain compatibility with the existing and formally adopted XML-based metadata specification, it is understood that any adoption of another markup language such as JSON must include open source normative software and tools to convert between these formats. Adoption of JSON based metadata is currently being considered for future versions of PyChips. If and when a successful implementation has been achieved, consideration for adopting JSON as another valid option for representing ION SDR Standard-compliant metadata in a future version of the standard will be requested.

SUMMARY AND CONCLUSION

Since GPS SDR developments started in the mid 90’s, together with the operational declaration of GPS, its feasibility has been widely proven by several platforms and their derivatives. We define GNSS SDR platforms as those implementing the receiver functions in general purpose software and processors, and divide them in real-time receivers, teaching/research tools, and snapshot receivers. We then describe some of them, with focus on those related to the authors but also including other developments. In particular, and based on the pioneering work by D. Akos, we describe the bit-wise parallelism platform by the Cornell GPS group, which led to GRID by UT Austin; the MuSNAT receiver by UniBwM, which also led to IFEN GmbH’s SX3 commercial receiver; The SoftGPS Matlab receiver and associated book, widely used for GNSS teaching and also influencing other platforms, such as FGI-GSRx; the popular C++ open source GNSS-SDR by CTTC; AutoNav SDR by Inha University; PyChips by S. Gunawardena and based on Python; the snapshot GNSS receiver by UAB, leading to cloudGNSSrx; the real-time N-GENE receiver by LINKS, used for early testing of Galileo first signals and OSNMA and the MATRIX receiver by ASPIN for navigation with terrestrial and space-based SOP among others. We provide an overview of the tasks and components of SDR front-ends, and for this purpose we describe Fraunhofer developments from the last years as a reference. Finally, we discuss the ION SDR Standard, officially approved by ION in 2020, and its current extensions.

In view of the impact in the GNSS community and the progress in the last decades, we conclude that GNSS SDR has a promising future and will continue coexisting with FPGA and ASIC receivers for the decades to come.
ACKNOWLEDGEMENTS AND REMARKS

The authors are mainly listed in alphabetical order to reflect the variety and importance of all contributions. The contributions have been partly edited by Thomas Pany, who also takes the responsibility for the structure of the work, the abstract and the introduction and is thus listed as first author. The article is based on a contribution by the same authors to the ION-GNSS+ conference 2022 in Denver, CO.
The GNSS SDR developments at the University of the Bundeswehr Munich of Section 3.2 were supported by via numerous research projects administered by the German DLR and were financed by the German Federal Ministry for Economic Affairs and Climate Action (BMWK). The authors acknowledge the financial support by the University of the Bundeswehr München for this publication.

At Northeastern University, support has been partially provided by the National Science Foundation (NSF) under Awards ECCS-1845833 and CCF-2326559.

The UAB snapshot GNSS software receiver was supported in part by numerous ESA-funded research projects and by the Spanish State Research Agency (AEI) project PID2020-118984GB-I00.

The MATRIX SDR development at the ASPIN Laboratory was supported by the Office of Naval Research (ONR) under Grants N00014-16-1-2305, N00014-19-1-2613, and N00014-19-1-2511; the Air Force Office of Scientific Research (AFOSR) under Grant FA9550-22-1-0476; the U.S. Department of Transportation (USDOT) under Grant 69A3552047138 for the CAR-MEN University Transportation Center (UTC); and the National Institute of Standards and Technology (NIST) under Grant 70NANB17H192.

The authors would like to acknowledge the work of Prof. Kai Borre, who passed away in 2017, for his influential contributions to the GNSS SDR field over the last decades.

The views expressed in this article are those of the individual authors and do not reflect the official policy or position of any state or government entity.

References


BeiDou B1I receiver processing. (2022). In *GNSS software receivers* (pp. 153–163). Cambridge University Press. [https://doi.org/10.1017/9781108934176.007](https://doi.org/10.1017/9781108934176.007)


Foerster, F., & Pany, T. (2013). Device and method for producing a data stream on the basis of data packets provided with packet sequence marks, and satellite receivers for providing the data stream (Nos. US patent no. 8451170, 2011-03-03; German patent no. 102008014981, 2009-10-15).

Fokin, G., & Volgushev, D. (2022, Mar.). Software-defined radio network positioning technology design. problem statement. In *Proceedings of Systems of Signals Generating and Processing in the Field of on Board Communications* (pp. 1–6). IEEE. [https://doi.org/10.1109/IEEECONF53456.2022.9744391](https://doi.org/10.1109/IEEECONF53456.2022.9744391)

Galileo E1 receiver processing. (2022). In *GNSS software receivers* (pp. 140–152). Cambridge University Press. [https://doi.org/10.1017/9781108934176.006](https://doi.org/10.1017/9781108934176.006)

GLONASS L1OF receiver processing. (2022). In *GNSS software receivers* (pp. 126–139). Cambridge University Press. [https://doi.org/10.1017/9781108934176.005](https://doi.org/10.1017/9781108934176.005)


GPS L1 C/A receiver processing. (2022). In *GNSS software receivers* (pp. 108–125). Cambridge University Press. [https://doi.org/10.1017/9781108934176.004](https://doi.org/10.1017/9781108934176.004)


68(4), 673–685. [https://doi.org/10.1002/navi.449]


NavIC L5 receiver processing. (2022). In *GNSS software receivers* (pp. 164–173). Cambridge University Press. [https://doi.org/10.1017/9781108934176.008]


NTLAB, UAB. (2022). *Unique ICs for GNSS Receivers.* [Accessed on 2022-09-26] [https://ntlab.lt/]


**How to cite this article:** Thomas Pany Akos D., Arribas J. M. Zahidul H. Bhuiyan Pau Closas Fabio Dovis Ignacio Fernandez-Hernandez Carles Fernández–Prades Sanjeev Gunawardena Todd Humphreys Zaher M. Kassas José A. López Salcedo Mario Nicola Mark L. Psiaki Alexander Rügamer Young-Jin Song Jong-Hoon Won , GNSS SDR History and Status. *NAVIGATION*, xxxx;yyy:z–t.