

A LabVIEW-Based GPS Receiver Development and Testing Platform with DSP Peripherals: Case study with C6713 DSK

Arpine Soghoian and David Akopian
The University of Texas at San Antonio

Abstract

The modernization of Global Positioning Systems (GPS) and the availability of more complex signals and modulation schemes boost the development of civil and military applications while the accuracy and coverage of receivers continually improve. Recently, software defined receiver solutions gained attention for flexible multimode operations. For them, developers address algorithmic and hardware accelerators or their hybrids for fast prototyping and testing high performance receivers for various conditions. This paper presents a new fast prototyping concept exploiting digital signal processor (DSP) peripherals and the benefits of the host environment using the National Instruments (NI) LabVIEW platform. With a reasonable distribution of tasks between the host hardware and reconfigurable peripherals, a higher performance is achieved. As a case study, in this paper the Texas Instruments (TI) TMS320C6713 DSP is used along with a Real Time Data Exchange (RTDX) communication link to compare with similar Simulink-based solutions. The proposed testbed GPS signal is created using the NI PXI signal generator and the NI GPS Simulation Toolkit.

Keywords: GPS receiver, NI LabVIEW, SDR, A-GPS, RTDX

1. Introduction

The US GPS is one of the existing Global Navigation Satellite Systems (GNSS) that provides the end user with improved position, velocity and time solutions. GPS/GNSS receivers continually evolve by progressively modernizing conventional algorithms and implementation platforms for faster operation and development. In our previous work (Akopian et al., 2011) it was mentioned that major challenges of advanced receiver development, especially in academia, are system development complexity, long development cycles, RF front-end and hardware accelerator interfaces for real-time processing, and access to end-to-end development and testing platforms.

While GNSS systems perform very well in strong signal conditions, their operation in many urban and indoor applications is difficult or impossible due to weak signals and strong distortions. The modernization of existing and new signals and modulation schemes adds to system complexity; challenging the research community to explore new algorithms and methods.

The urban and indoor applications of the GPS/GNSS receiver operation are based on the technology called Assisted GPS/GNSS (A-GPS/A-GNSS) (Misra, Enge, 2001), (Agarwal et al., 2002), (Wireless E911 location accuracy requirements, 2012), (Zhao, 2002). In this approach, wireless channels are used to deliver aiding data to receivers, which they would normally need to receive and demodulate from weak GPS satellite signals. Assisted technology improves sensitivity and operational coverage of receivers. A-GPS and its hybrids are considered the best global positioning technologies for wireless devices; as a result A-GPS is standardized for all wireless networks (Wireless E911 location accuracy requirements, 2012), (Zhao, 2002).

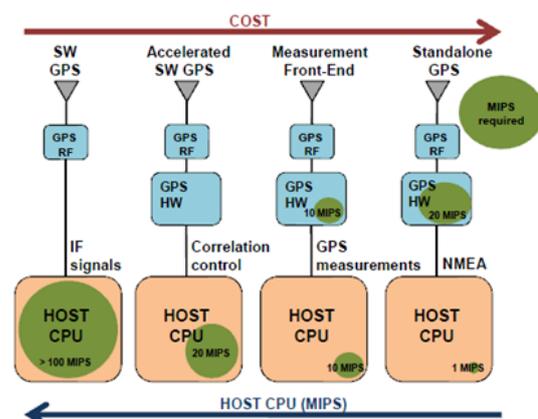


Figure 1: An example of various hardware/ software configurations in a commercial receiver product line (Fastrax, 2012).

With higher-level receiver requirements, one should address more and more complicated phenomena and typically, state-of-the-art receivers should be flexible to

perform multimode tasks for operating in various conditions. The developers need reference receivers, associated software development kits (SDK), development platforms, simulators, and testbeds to accelerate and facilitate their research. Even in assisted mode, GPS receivers need massive correlators to enhance received signals by combining multiple received signal copies and accelerating computations. A popular solution for flexible multimode operations is an SDR receiver (Akos, 2003), where real-time correlators, tracking loops, and navigation calculation are all implemented in software or using general purpose accelerators such as FPGA and DSP peripherals. Massive correlators are implemented using fast algorithmic solutions or deployed on accelerators. Fig. 1 illustrates relative costs and various hardware-software configurations (Fastrax, 2012) for GPS receivers and the increased role of central processing units (CPU) in SDR solutions. Software GNSS receivers require fewer hardware components, offer significant flexibility compared to conventional receivers with correlators implemented on dedicated application-specific integrated circuit (ASIC) technology, and are convenient for faster prototyping and academic research.

The paper elaborates several benefits of using NI LabVIEW (LabVIEW, 2012) as a host platform for fast receiver development, prototyping, simulation, testing, and implementation of A-GPS (Zhao, 2002) support. This paper systematizes and extends an initial feasibility study of LabVIEW-based receiver development (Akopian et al., 2011) where it was shown that a LabVIEW-based receiver is an attractive alternative from a performance point of view and facilitates interfacing with RF front-ends. In this paper, we also demonstrate options on how a LabVIEW-based receiver can connect to DSP accelerators. A dedicated NI LabVIEW-based A-GPS support is also developed to generate assistance data for available satellites (Akopian et al., 2011).

For DSP accelerator implementation, the Code Composer Studio (CCS 3.1) C6713 Device Functional Simulator from TI is used. The peripheral connects to the NI LabVIEW environment through the RTDX (Code Composer Studio, 2012) interface. Comparison of the algorithm performance is done between LabVIEW receiver and complete Simulink receiver implementation as described in (Hamza et al., 2009).

The paper is organized as follows. Section 2 summarizes the GPS system architecture including hardware/software components. Section 3 describes the NI LabVIEW-based testbed including NI GPS Simulator Toolkit and A-GPS support. Section 4 overviews the advanced algorithms implemented in our case study. Section 5 presents DSP target compilation support in the LabVIEW environment. Section 6 gives the detailed implementation description of the GPS receiver with the A-GPS support. Section 7 provides conclusions.

2. Software Defined Radio Concept For GPS Receiver Development

Fig. 2 shows a generic GPS/GNSS receiver architecture design where all of the signal-processing tasks can be implemented on software, eliminating hardware accelerators. The software receiver design offers high flexibility for implementing various algorithms without constraints imposed by fixed hardware architectures and is convenient for multimode operations in challenging environments.

Examples of SDR are the gpsSrx receiver (Akos et al., 2001) and NavX-NSR (Heinrichs et al., 2007), and for the open source PC-based GNSS SDR systems there are examples described in (Borre et al., 2006), (GPS-SDR, 2012). Proprietary or open source toolkits are also developed by incorporating new components in existing GNSS receiver solutions. Examples of toolkits range from Matlab-based (Matlab, 2012) receiver solutions to C/C++ based open source (Borre et al., 2006) and commercial systems (Fastrax, 2012), (Akos et al., 2001), (Heinrichs et al., 2007).

High performance conventional GPS/GNSS receivers rely on ASIC technology to implement massive correlators, as the performance of SDR solutions is still limited. This provides for high performance but limited reconfigurability compared to SDR receivers. Complementing SDR receivers with programmable hardware such as DSPs and FPGAs provides for a trade-off between acceleration and reconfiguration. ASIC hardware implementation requires long development cycles and additional hardware-software interfacing efforts, while FPGA and DSP development cycles are shorter due to convenient development tools and standard interfacing means.

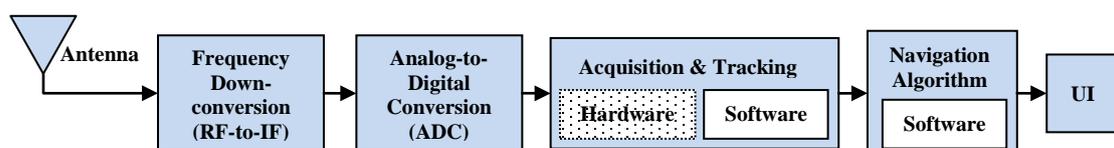


Figure 2: Schematic structure of a GPS receiver. Hardware accelerators are often completely excluded in software defined GPS

Table 1: Advanced Development-Testing Toolkit to Facilitate Research and Dissemination (Akopian et al, 2011)

Components	Benefits
<ul style="list-style-type: none"> • An integrated platform using popular NI RF/PXI equipment, NI GPS simulator and LabVIEW software to facilitate algorithm research and performance analysis • LabVIEW algorithmic libraries: conventional receiver algorithms; Advanced algorithms; Advanced A-GPS support 	<ul style="list-style-type: none"> • Low-cost real-time processing platform with available RF Front-end interfaces and rich algorithmic libraries • Dataflow and/or C/C++ programming, available profiling tools to estimate performances, built-in user interface design and visualization tools • Easy access to RF and I/Q sampled signals from NI LabVIEW-based GPS simulator • UTSA SUPL standard-based A-GPS support • Build-in compilers for target platforms such as DSP processors and FPGAs

Table 2: LabVIEW DSP Hardware and Software Support

DSP Hardware Targets Supported by NI LabVIEW DSP Module	NI Software Support for DSP Targets
<ul style="list-style-type: none"> • TMS320C6711 DSK • TMS320C6713 DSK • NI's SPEEDY-33 	<ul style="list-style-type: none"> • LabVIEW DSP Module • LabVIEW DSP Test Integration Toolkit • TI DSP application as an external application in LabVIEW

This paper focuses on GPS receivers with DSP peripherals specifically optimized for the efficient execution of common signal processing tasks. DSPs are microcontrollers designed specifically for signal processing applications. Unlike ASICs, DSPs are not as efficient in terms of speed and power consumption. However, they are characterized by their flexibility and ease of programming. Examples of the highly sensitive GNSS receivers using DSPs are receivers in (Girau et al., 2007), (Cetin et al., 2007). DSP-based solutions might be slower but still allow for high throughput correlator implementations with a shorter development cycle, using e.g., Fast Fourier Transform (FFT) algorithms.

3. NI LabVIEW-Based Testbed for GPS SDR Development

Table 1 summarizes the benefits for choosing a LabVIEW environment as a GPS receiver development platform. NI LabVIEW provides hundreds of built-in libraries for advanced development, analysis, and data visualization. It is convenient for fast algorithm prototyping and testing, comparative studies, real-time performance evaluation and dissemination. LabVIEW has integrated interfaces to various hardware peripherals. It also supports multithreading and multicore programming, which is useful for real-time applications.

These advantages already have been used in other radio communication systems (Developing an OFDM Transmitter, 2012), (Prototyping Algorithms for Next-Generation Radio Astronomy Receivers, 2012).

The algorithms in LabVIEW can be implemented using LabVIEW library modules, Mathscript nodes using Matlab scripts for fast prototyping, and C/C++ language for fast processing. LabVIEW-based implementation can be transformed to C/C++ implementation for open source and commercial solutions using LabWindows/CVI (NI LabWindows/CVI, 2012). LabWindows/CVI supports convenient interfaces with peripheral instrumentation, and provides graphical user interface (GUI) design tools.

Aside from the instrument control with NI hardware, LabVIEW also has support for external hardware peripherals to accelerate research and simulation of SDR development. As already mentioned, the hardware accelerators such as DSPs and FPGAs can be integrated with LabVIEW (Soghoyan et al., 2011). There are different integration modes. The easiest solution for developers is to transform a LabVIEW data flow schematic into a DSP code using a NI LabVIEW DSP Module. Only a few DSPs are supported by this software as listed in Table 2. Alternatively, the connection with many other DSPs can be performed in a conventional

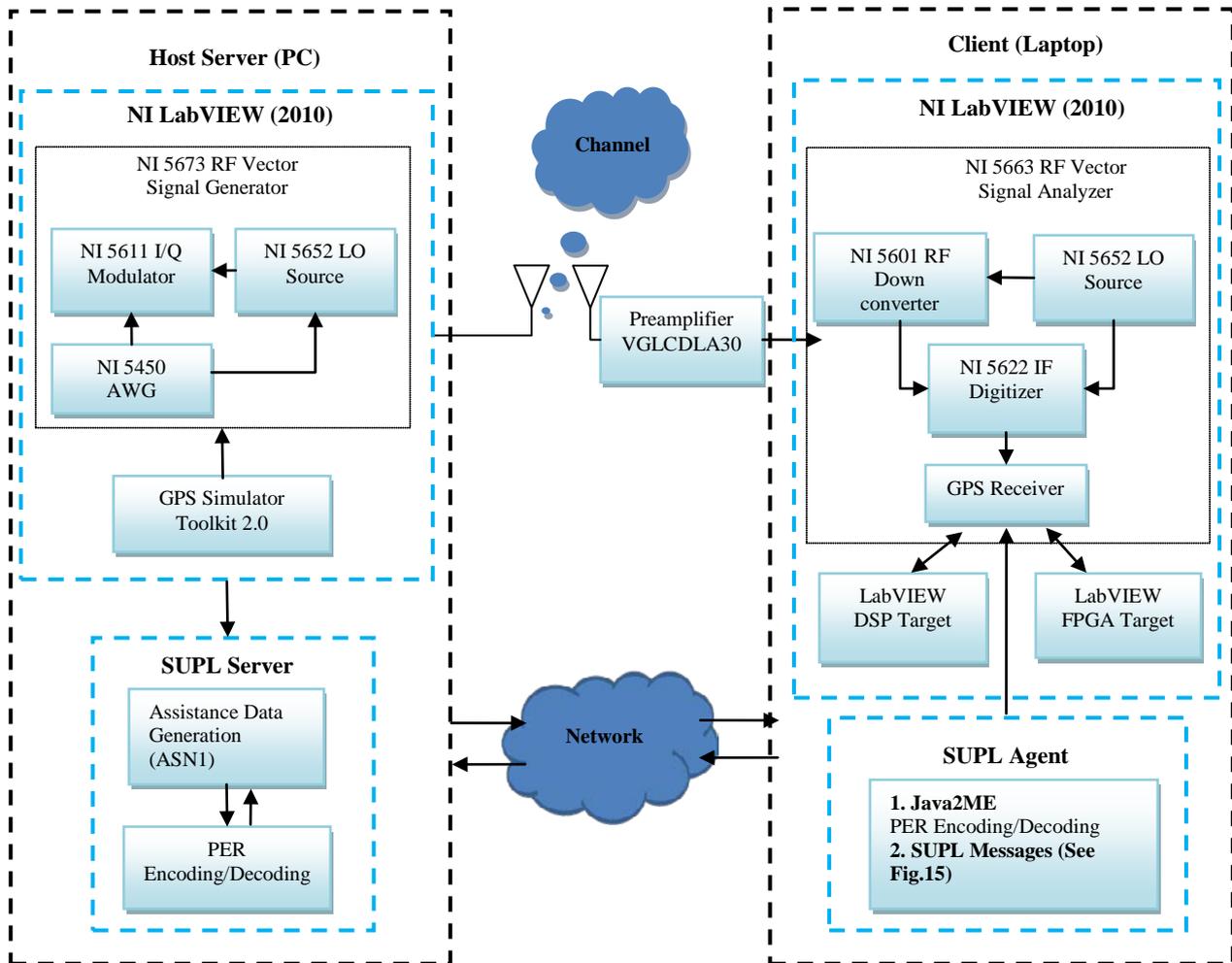


Figure 3: Integrated receiver architecture with hardware front-end, A-GPS support, LabVIEW DSP and LabVIEW FPGA targets

way, connecting a DSP application to LabVIEW using RTDX communication tools. This is accomplished through software support called NI LabVIEW Test Integration Toolkit. Another option is to use dynamic linked libraries (dll) to call the CCS standalone application externally in the LabVIEW environment using “Call Library Function Node” functionality.

The initial system architecture for the real-time GPS receiver LabVIEW-based testbed first has been presented in (Soghoyan et al., 2011). Fig. 3 illustrates an enhanced version of the system, which provides A-GPS assistance delivery support, channel modelling extension for the NI GPS simulator, and FPGA/DSP accelerators. Details about FPGA integration will be described in another paper. The transmitter’s RF front-end consists of an antenna and NI PXIe-5673 RF vector signal generator (VSG), which can be configured to generate various GNSS signals. The GPS signal is generated using a LabVIEW-based NI GPS Simulation Toolkit (LabVIEW, 2012). Receiver RF front-end consists of an

antenna, a variable gain amplifier (variable gain up to 30dB) (GPS Networking, 2012) and NI PXIe-5663RF vector signal analyzer (VSA). It is connected to a computer running a LabVIEW-based software defined GPS receiver. Optional hardware DSP/FPGA peripherals are accessible through LabVIEW interfaces.

The dedicated LabVIEW-based A-GPS support is integrated with NI GPS Simulation Toolkit. The simulator generates signal using ephemeris and almanac orbital parameters for all of the selected satellites. Proposed A-GPS support encapsulates the binary navigation data into assistance data, which are communicated by an assistance server to receivers. The process follows the guidelines of A-GPS assistance delivery according to Secure User Plane Location (SUPL) (Open Mobile Alliance, 2007), which defines the assistance delivery format and communication between the user (client) and SUPL server. SUPL is the Internet Protocol (IP)-based network service to deliver information through a User Plane bearer between a

SUPL enabled terminal (SET) (e.g. mobile devices) and a SUPL location platform (SLP) (e.g. A-GPS servers) for wireless communications developed by OMA (Open Mobile Alliance, 2007). A detailed description of an assistance data delivery solution can be found in (Narisetty et al., 2012).

3.1 Front-End GPS signal simulator and transmitter

The NI GPS Simulation Toolkit (LabVIEW, 2012) along with NI 5673 VSG is enabling the generation of one to twelve satellite signals with a waveform duration of up to 12.5 minutes (25 frames) which is the duration of an entire navigation message.

The main features of the GPS Simulation Toolkit are the following:

- Generation 24 hours of up to 12 satellite C/A codes.
- Power adjustment levels of each satellite based on the testing specifications (-145dBm to +10dBm).
- Direct streaming generation with either of those
 - o Wide Area Augmentation System (WAAS)
 - o Trajectory scripts
 - o On the fly parameters
 - o Stored file
- Use the recorded stream of a simulated signal along with the NI RF vector signal generator for reliable and low-cost GPS testing.

Additional selected simulator features are listed in Table 3. The first example application has a common interface with the rest of the examples. The simulator determines visible GPS satellites based on the almanac and ephemeris files, GPS time, and receiver location specified. The simulator provides for satellite power control and WAAS availability concurrently with GPS signal. The toolkit simulates only those user-specified WAAS satellites that are present in the WAAS GEO file that can be downloaded from (WAAS Test Team, 2012).

The simulator generates baseband GPS L1 band signals with a sampling rate of 1.5MS/sec (“IQ interleaved integer 16” data type). The almanac and ephemeris files necessary for the signal generation contain satellite orbit parameters and related information, which are used to estimate satellite locations, trajectories, and health for a specific date and time (Misra, Enge, 2001), (Kaplan, 1996). The almanac and ephemeris file types, content and location are described in (Almanac information, 2012), (Ephemeris information, 2012). Channel models can be developed in LabVIEW and integrated with the toolkit. The proposed testbed shown in Fig. 3 integrated Urban Three-State Fade Model (UTSFM) channel model (Ma et al., 2001) as described in (Soghoyan et al., 2011).

Table 3: GPS Simulation Toolkit 2.0 Programming and Interactive Applications

Example Application	Description
niGPS Adjust Satellite Power	This example takes as an input Receiver location, Almanac and Ephemeris files, Initial GPS Time of Week, Initial speed of the receiver(m/s), Maximum number of optimal satellites, and dynamically adjustable power of individual satellite and provides as an output GPS L1 band signal (Fig. 4) using the hardware resource specified. WAAS GEO files can be added to the simulation if the control is enabled.
niGPS Direct Streaming Generation with WAAS	This example is similar to the first one in addition WAAS satellites are augmented to GPS L1 signal given the WAAS file path.
niGPS Write Waveform To File (Single Satellite,Manual Mode)	This example takes as an input almanac and ephemeris files, number of frames, PRN number of an individual satellite with its Doppler shift, pseudorange and waveform scaling factor providing as an output the bits of GPS L1 signal in a binary file specified by the user.
niGPS Write Waveform to File (Simple,Automatic Mode)	In this example optimal satellites available are automatically selected based on the inputs of Receiver location, Almanac and Ephemeris files, Initial GPS Time of Week. The output is a binary file containing GPS L1 signal.
niGPS Streaming From File	This example inputs a binary file with GPS L1 band signal which is then streamed using NI RF vector signal generator.

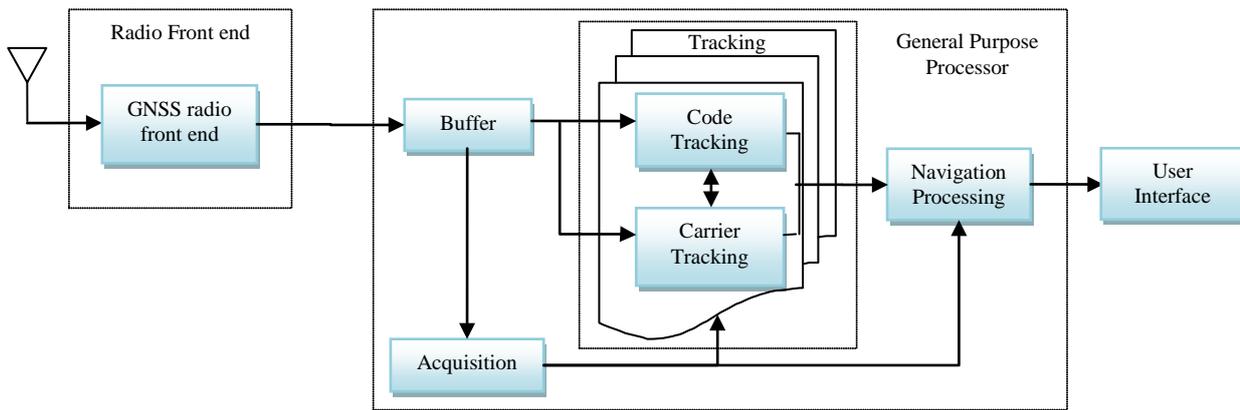


Figure 4: Software GPS/GNSS receiver architecture

4. GPS Receiver Signal Processing

The conventional L1 civilian GPS signal is a direct sequence spread spectrum (DSSS) signal consisting of a multiplication of a sinusoidal carrier at 1.57542 GHz, a binary phase shift keying (BPSK) navigation signal at 50Hz, and BPSK modulated spreading pseudorandom code signal (PRN) at 1.023 Mchips/sec. The spreading sequence is called the C/A (coarse acquisition) code. The generated signal is transmitted through a channel described in (Misra, Enge, 2001).

The ultimate goal of a GPS receiver is to provide position, velocity, and time information. For that, receivers measure range, range-rate, and demodulate navigation data. Range and range-rate measurements are extracted by synchronizing locally generated replicas of the code with the received signal. This synchronization is performed in frequency by estimating Doppler modulation, and in time, by aligning the signal and replica and estimating their relative shift called code-phase. The synchronization typically is performed in two phases: acquisition (coarse) and tracking (fine). These two modules constitute a baseband processing stage. After the synchronization, navigation data are simply obtained through sinusoidal carrier and PRN code wipe-off. Navigation data contain time stamps, which, along with code phases, are used to estimate ranges to satellites. Navigation data also contain orbital parameters, ephemeris and almanac, which are used to find satellite positions provided time. These parameters can alternatively be received using assistance from wireless networks as described later in the paper. Satellite positions serve as beacon locations for trilateration using ranges. The baseband configuration for a software receiver (Fig. 4) follows the processing chain of conventional receivers (Fig. 2).

Receiver operation is implemented using the modified version of the application from the LabVIEW build-in examples' package called "RFSA Acquire Continuous IQ.vi." Here the incoming signal is of integer 16 IQ

interleaved format, which then is converted into integer 8 datatype for faster receiver processing. The settings chosen for the signal acquisition are the following; the IQ sampling rate is 4.092MS/s, the reference power level is adjustable, the GPS carrier signal frequency is 1.57542GHz, and the number of samples to read per each block of the received signal is 81840. Reference power level adjustment was done according to (GPS Receiver Testing, 2010), (GPS Multiple Satellite, 2012).

4.1 Acquisition

The first stage of baseband signal processing is the acquisition of a satellite. A receiver replicates a code and a residual carrier signals matching those to the received signal in a two-dimensional search process. Conventional receivers achieve acquisition by searching over a predicted time-frequency uncertainty zone.

Multiple possible signal replicas are generated and correlated with received signal to find a match and thus to identify input signal parameters. A statistical test is applied to the correlation results to determine if a signal acquisition has been reached or not. If it has been, the acquisition is terminated and the receiver starts the tracking stage for that satellite; if not, the search continues and moves to the next code-phase/frequency option.

4.2 Tracking

In the tracking stage, the residual code and carrier shifts are estimated using correlators, discriminators and a feedback loop to reduce signal misalignment adaptively. The code tracking loop called delay lock loop (DLL) and a carrier tracking loop called phase/frequency locked loop (PLL/FLL) are used for consecutive fine alignment of received and replicated signals. The DLL loop aligns the incoming signal with the local PRN replica for code-phase estimation. In the conventional systems, several replicas are used with shifted relative phases, e.g. three replicas early, prompt and late with a code-phase spacing of $\pm 1/2$ chip are used by slightly shifting code replicas. The correlation outputs are fed into a discriminator,

which defines the relative shift of the received signal with respect to the prompt replica. Then the feedback from the discriminator is used to adjust the code phases of local replicas. In PLL/FLL (Misra, Enge, 2001) the output of the ‘prompt’ correlator connects to the discriminator of the carrier loop with the phase estimation between I and Q components. The PLL loop consecutively estimates phase mismatch and adjusts the locally generated residual carrier replica to minimize Q component. Phase changes indicate the presence of a residual sinusoidal modulation, and the PLL discriminator estimates this presence and instructs to adjust replica residual carrier frequency. Typically, filters are used to smooth discriminator outputs and avoid DLL/PLL loop overreactions.

4.3 Advanced correlators

Both acquisition and tracking algorithms use correlators to synchronize the incoming signal with the local replica. Typically, there is an element-wise multiplication of the received samples with the samples of each replica and eventually the products are integrated for the result. So-called block correlators are implemented to reduce the computational loads performing shared computations. Examples of state-of-the-art block correlators are

(Akopian, 2005) for acquisition (Fig. 5), and (Sagiraju et al., 2008) for tracking (Fig. 6).

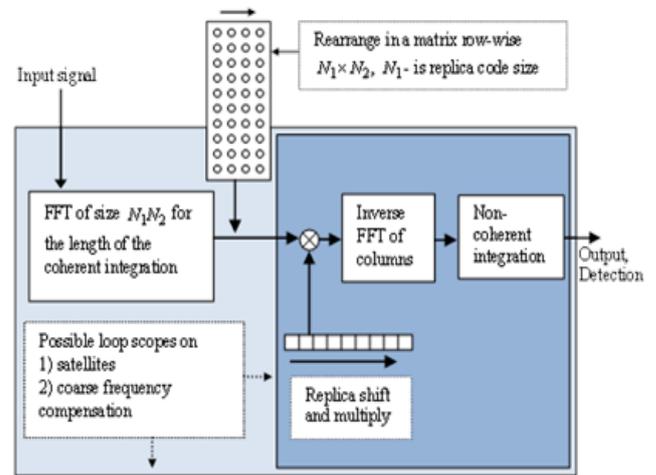


Figure 5: Block correlator for acquisition, coherent integration length is N_2 code periods ($N_1 N_2$ samples for N_1 samples per code period)

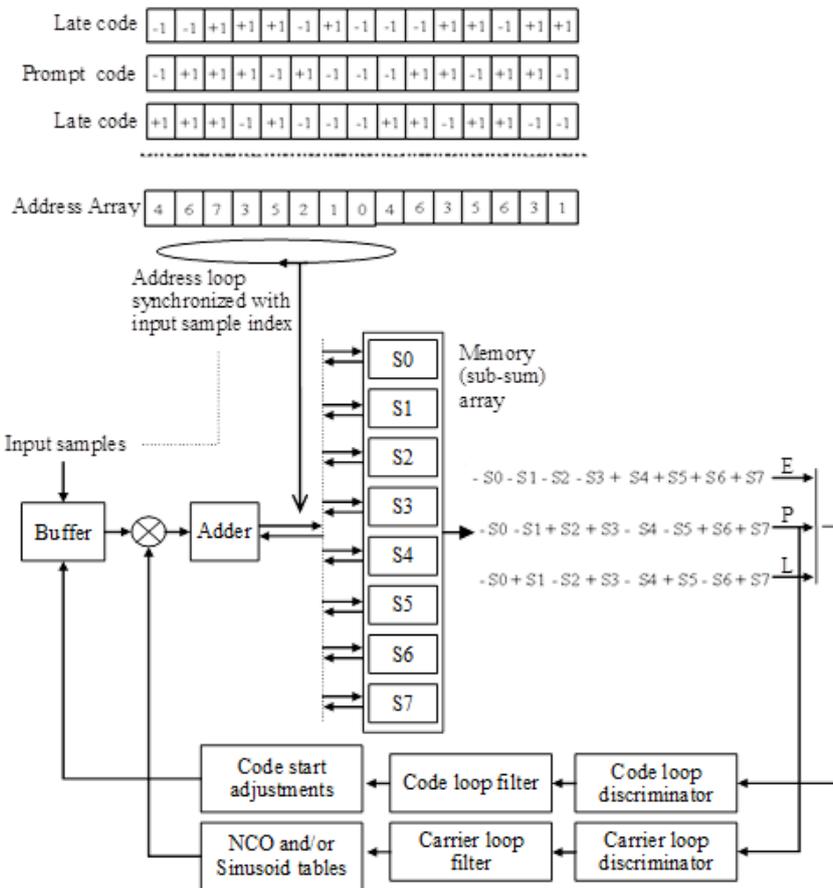


Figure 6: Block correlator structure for tracking. An example with three replica sequences; sub-sum combining

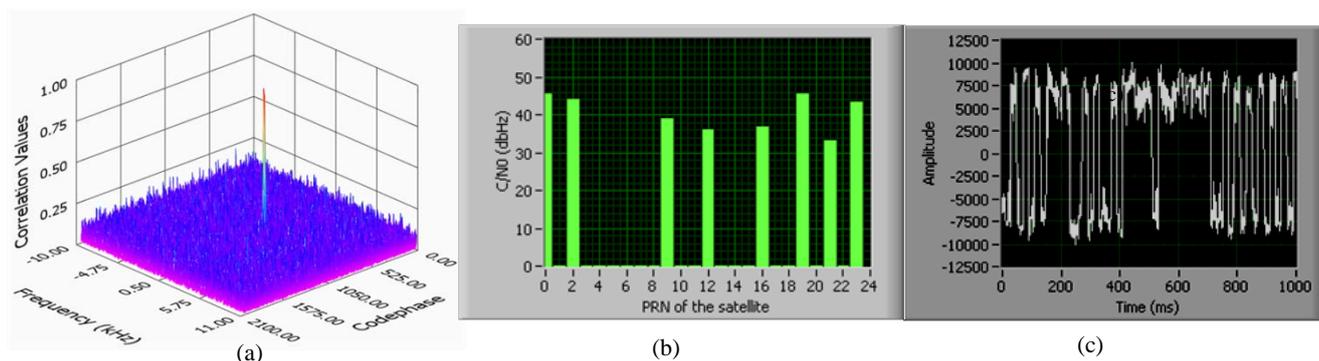


Figure 7: (a) Acquisition result of a single satellite using LabVIEW Visualization Tools. (b) The signal strengths and PRNs of the acquired satellites. (c) Bits of the navigation message for a single satellite

The block correlators for acquisition efficiently perform a parallel joint search in three dimensions: code phase, Doppler frequency, and satellite number using e.g. FFT. In (Sagiraju et al., 2008), many frequency search steps are implemented through iterations in the frequency domain; a long FFT is computed once, while shorter inverse FFTs are computed for each Doppler frequency and satellite.

The tracking block-correlator concept (Sagiraju et al., 2008) implements several correlators jointly in the time domain. Early-Late-Prompt replica sequences are transformed to a set of addresses pointing to a set of registers that accumulate partial correlations. Each incoming sample is added to a register as referred by the address array. Then these partial sums are combined as shown in Fig. 6. The acceleration is achieved by only one addition per sample for all three correlators plus a controlled overhead due to partial correlation integration. Overhead is not significant for three correlators. The tracking loop iterations are modified to make use of the block correlator concept. The DLL discriminator outputs estimate the misalignment of the incoming signal with the prompt code. As replicas are fixed, the code phase adjustments are performed by aligning the received code front edge with the front edge of the prompt replica code. Considering received signal as an array of data, one should shift a pointer to the start of the C/A code back and forth (Winternitz et al., 2004). These block correlators use only additions, which result in essential computational savings. Fig. 6 shows the combined carrier and code tracking loop.

In our case study, coherent integrations of 8ms of signal are used for advanced acquisition implementation. This allows determining the availability of the visible satellites with their code-phases and frequencies. The PRN code replicas for 24 satellite codes are stored in a 2D array for faster access. The correlation result will provide a 2D search over all Doppler frequencies and code phases and whenever there is a match between

incoming signal and the local replica there will be a peak like the one shown in Fig. 7(a). The examples of signal strengths (dBHz) for detected satellites are shown in Fig.7(b). Therefore, once the outputs of the acquisition stage are available we can proceed to the tracking algorithm.

The tracking algorithm implemented in the paper works for the sampling rate of 4.092 MHz. For the advanced tracking, these samples are integrated further to reduce the sampling rate to 2.046MHz. To avoid real time generation of the carrier wave a fixed sinusoid array is created to generate various Doppler modulation compensating sinusoids through the saved sinusoid decimations and cyclic array reading. The generated sinusoids are multiplied to the input signal to wipe-off the carrier. The resulting samples are accumulated into eight partial correlations (subsums). Fig. 7(c) illustrates the navigation message decoded in the tracking stage. The navigation databits and measurements are passed to position a computation module. In this implementation a conventional least squares positioning algorithm is used (Agarwal et al., 2002).

5. DSP as a Hardware Accelerator

As a user-friendly graphical programming language LabVIEW makes it easier to build DSP systems with fast application prototyping and deployment. It also allows creating reusable subvis; code blocks with input(s) and output(s). The NI LabVIEW environment connected with a LabVIEW DSP Module provides a hands on experimental learning environment for novice users, and self documenting, easily maintainable environment for professionals.

TI TMS320C6713 DSP starter kit (C6713 DSK) (TI TMS320C6713 DSP, 2012) is used as a case study. It has 512K Flash and 16MB SDRAM memories, which will be sufficient for the advanced acquisition algorithm

development. The property list of C6713 includes the following (Texas Instruments, 2011):

- C6713 operates at 225 MHz.
- 16 MB synchronous DRAM.
- 512 KB non-volatile Flash memory (256 KB usable in the default configuration).
- Software board configuration through registers implemented in a Complex Programmable Logic Device (CPLD).
- Standard expansion connectors for daughter card usage.
- Joint Test Action Group (JTAG) emulation through an onboard JTAG emulator with USB host interface or external emulator.

The functional block diagram with given features is shown in Fig. 8. Real-time bidirectional data exchange between host and target DSPs is maintained through a JTAG interface using the RTDX channel communication (Chassaing, Reay, 2008).

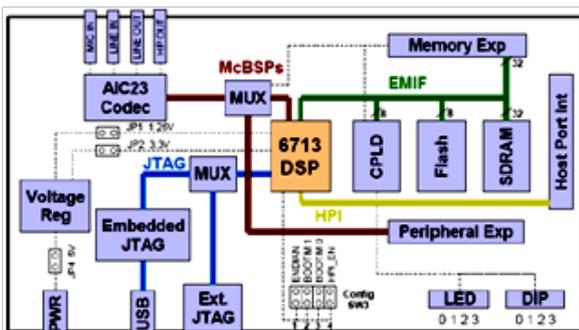


Figure 8: Functional block diagram of the C6713 DSK (Benveniste et al, 2010)

Two different approaches to operate NI supported DSP boards in a LabVIEW platform are described in detail below. The target requirement platform is selected to be NI LabVIEW 2010 version. The DSP hardware/software installation is done according to (TI TMS320C6713 DSP, 2012).

5.1 LabVIEW DSP module

The first option is to use a LabVIEW DSP Module to create a DSP project (shown in Fig. 9) in a LabVIEW environment according to (NI LabVIEW DSP, 2012). Here the idea is to “automatically” map a LabVIEW design into a DSP code. The LabVIEW DSP Module identifies the available hardware I/O points from the supported hardware and it can easily switch between existing DSP targets if needed. The DSP module also allows deploying and running the application in a standalone mode. The NI LabVIEW DSP Module has a limitation to work only with three types of targets; NI SPEEDY-33, TI C6711 and C6713 DSKs. There are also other constraints. For example, by default, the NI LabVIEW DSP Module uses Flash memory of the target.

Due to the memory constraint, only smaller tasks can be delegated to the DSP. In addition to that, target performance and memory profiling are disabled on the specified DSP peripherals. We succeeded at developing a single satellite acquisition with 1ms coherent integration length of an input signal sampled at 1.024MHz sampling rate.

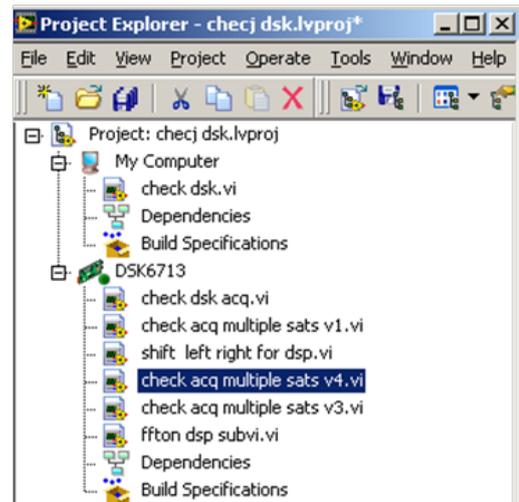


Figure 9: View of a Project Explorer in NI LabVIEW environment

A portion of the acquisition code implementing the FFT on the DSP module is depicted in Fig. 10 to demonstrate the concept. “EMB Real FFT” DSP Module function is used for the application development according to the advanced acquisition algorithm described in Section 4. The inverse FFT is also done using the same “EMB Real FFT” function.

The second approach described below is able to utilize additional external memory resources and is applicable to a broader set of DSP targets.

5.2 LabVIEW DSP test integration toolkit

The more general second option is called a LabVIEW DSP Test Integration Toolkit (NI LabVIEW Test Integration Toolkit, 2012). The supporting libraries can be downloaded from (NI LabVIEW Test Integration Toolkit, 2012). A LabVIEW DSP Test Integration Toolkit allows third party DSP target code integration into the LabVIEW environment, e.g., using a TI CCS Integrated Development Environment (IDE) to create test systems for DSP target development. CCS is the programming, building, and debugging interface of TI DSPs. To configure a platform and make a basic setup of the CCS v3.1 project using RTDX library functions refer to (TI TMS320C6713 DSP, 2012), (TMS320C6000 Code Composer Studio Tutorial, 2012). Once the project is created in *Projects-Build Options-Linker* the configuration is set up in the following way.

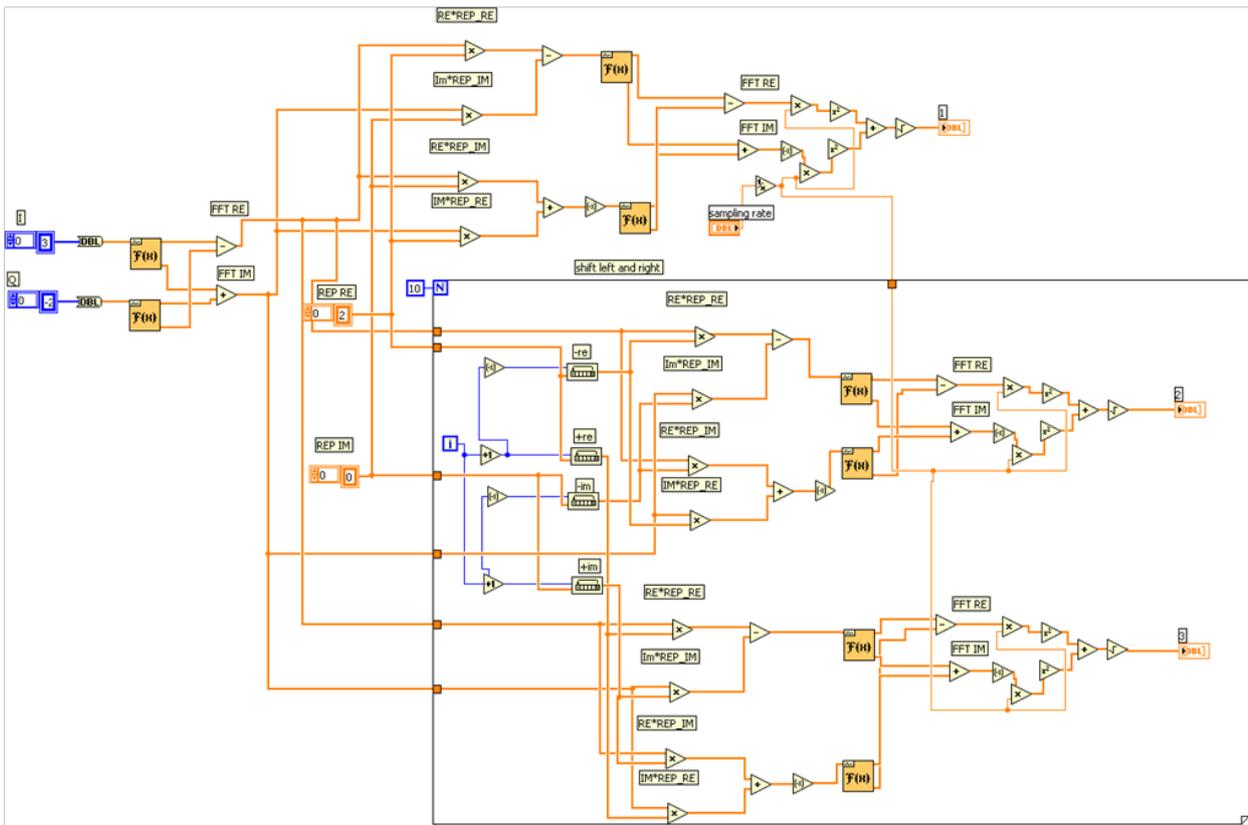


Figure 5: Acquisition algorithm implementation in LabVIEW environment

```
-c -heap10000 -m".\Debug\t2h.map" -
o".\Debug\t2h.out" -stack10000 -w -x -
i"C:\CCStudio_v3.1\C6000\dsk6713\include" -
i"C:\CCStudio_v3.1\C6000\cgtools\include" -
i"C:\CCStudio_v3.1\C6000\cs1\include" -
i"C:\CCStudio_v3.1\c6700\dsplib\support\fft" -
i"C:\CCStudio_v3.1\c6700\dsplib\include" -
-l"rts6700.lib" -l"rt dx.lib" -
l"C:\CCStudio_v3.1\C6000\dsk6713\lib\dsk6713bs
l.lib" -
l"C:\CCStudio_v3.1\C6000\cs1\lib\cs16713.lib"
```

These are the supporting libraries to implement RTDX communication, FFT operations, compilation on the C6713 DSK target, etc.

The source codes must be added in order to run the CCS application. To run the code follow the following steps:

1. File→Load→Program→Select the ".out" under debug folder.
2. Debug→Reset CPU
3. Debug→Restart
4. Debug→Go Main
5. Debug→Run

Alternately, the same process automated in LabVIEW is done using six advanced RTDX libraries: CCS Halt VI, CCS Close Project VI, CCS Open Project VI, CCS Download Code VI, CCS Build VI, and CCS Run VI. The .pjt file path is wired to the CCS Open Project VI to open the .pjt file in CCS IDE. The CCS Build VI builds the .pjt file (created in CCS v3.1) to create the DSP target code .out file. The CCS Download Code VI downloads the .out file to the development board. The CCS Run VI runs the embedded .out file on the development board. The RTDX communication and the memory VIs enable accessing data from the target code. RTDX is available on XDS510 and XDS560 class emulators (TMS320C6000 CCS Tutorial, 2012).

CCS IDE allows advanced debugging options, memory map, and a graphical view of the data in the project. In this paper the acquisition algorithm is heavily based on FFT (see Fig. 5) and a TI's C Callable Optimized FFT Function (Chassaing, Reay, 2008) is used.

For illustration purposes a fragment of the TI's C Callable Optimized FFT Function used in the CCS project is given below:

```
// N ->number of complex samples,
// Radix = 2;
for( i = 0 ; i < N/RADIX ; i++ )
// declare the FFT coefficients
{
W[i].re = cos(DELTA*i); //real component of W
W[i].im = sin(DELTA*i); //neg imag component
}
digitrev_index(iTwid, N/RADIX, RADIX);
bitrev(W, iTwid, N/RADIX); //bit reverse W
//get the real and imaginary of the input signal
for( j = 0; j < N; j++ ) {
x[j].re = values_re[j]; // I component
x[j].im = values_im[j]; // Q component
}
cfftr2_dit(x, W, N ) ; //TI floating-pt complex FFT
digitrev_index(iData, N, RADIX);
bitrev(x, iData, N); //freq scrambled->bit-reverse x
```

TI's radix - 2 optimized FFT function (cfftr2_dit), the function for generating the index for bit reversal (digitrev_index), and the function for the bit-reversal procedure (bitrev) are used.

For the external memory usage of the target a linker file is created (Fig. 11). The linker file provides general memory structure defined in the MEMORY section with designated origin and length. The directive SECTIONS allocate the application code sections into predefined memory locations. More information about the linker file structure is found in (Chassaing, Reay, 2008). The 24 shifted replica codes used by FFT-based acquisition in this paper are stored in the header file and added to the CCS project. The full acquisition algorithm is implemented completely on the C6713 target and called into the LabVIEW using RTDX communication.

The input and output RTDX channels are enabled in CCS using "rtdx.lib" library functionalities:

```
RTDX_CreateInputChannel(cinput);
RTDX_CreateOutputChannel(coutput);
void main()
{
// Target initialization for RTDX
TARGET_INITIALIZE();
/*enable RTDX channels*/
RTDX_enableInput(&cinput);
RTDX_enableOutput(&coutput);
}
```

```
/*C6713dsk.cmd Linker command file*/
-c
-heap 0x10000
-stack 0x10000
-u __vectors
-u __auto_init

_HWI_Cache_Control = 0;
_RTDX_interrupt_mask = ~0x000001808;

MEMORY
{
IVECS:   org=0h,          len=0x220
IRAM:    org=0x00000220, len=0x0002FDE0 /*internal memory*/
SDRAM1:  org=0x80000000, len=0x00100000 /*external memory*/
SDRAM2:  org=0x80100000, len=0x00100000 /*external memory*/
SDRAM3:  org=0x80200000, len=0x00100000 /*external memory*/
SDRAM4:  org=0x80300000, len=0x00100000 /*external memory*/
SDRAM5:  org=0x80400000, len=0x00200000 /*external memory*/
SDRAM6:  org=0x80600000, len=0x00200000 /*external memory*/
FLASH:   org=0x90000000, len=0x00020000 /*Flash memory*/
}

SECTIONS
{
.EXT_RAM :> SDRAM1
.intvecs :> IVECS /*in vector file*/
.text :> IRAM /*Created by C Compiler*/
.bss :> IRAM
.cinit :> SDRAM6
.pinit :> SDRAM3
.stack :> IRAM
.systemem :> IRAM
.const :> IRAM
.switch :> IRAM
.far :> SDRAM5
.cio :> IRAM
.csldata :> IRAM
.rtdx_text :> SDRAM2
.rtdx_data :> SDRAM2
.data :> SDRAM4
}
```

Figure 11: Linker command file for acquisition algorithm with external memory configuration

Once the channels are enabled the incoming IQ signal with the sampling rate of 2.048 MHz is transferred into the target with the RTDX_read() command and then the results of the acquisition stage are obtained, e.g.; the code phases and Doppler frequency shifts within the ± 10 kHz frequency range for all 24 satellites. These results are written into RTDX channel using a RTDX_write() command:

```
{
:
while(!RTDX_read(&cinput, input, sizeof(input)));
:
/* Wait for Target-to-Host transfer to complete */
if ( !RTDX_write( &coutput, &csat, sizeof(csat) ) )
{
fprintf(stderr, "\nError: RTDX_write() failed!\n");
abort();
}
while ( RTDX_writing != NULL )
{
#if RTDX_POLLING_IMPLEMENTATION
RTDX_Poll();
#endif
}
}
RTDX_disableOutput(&coutput);
RTDX_disableInput(&cinput);}
```

Once the results are verified in the CCS environment the NI LabVIEW RTDX support application can be created. The block diagram in Fig. 12 shows how to automate the process of compiling DSP target code and embedding the code on a development board in LabVIEW as explained earlier.

The input signal is transferred to the DSP target using “CCS RTDX Write Array I16.vi.” The streaming from the target to the host is done using “CCS RTDX Read SGL.vi.” To be able to run the RTDX streaming for more than 1024 bytes the RTDX buffer size should be modified as follows (Code Composer Studio, 2012).

1. In CCS, Tools-RTDX-Configuration Control is selected to display the RTDX-Configuration Control window.
2. The *enable* RTDX checkbox is not selected to ensure that RTDX is disabled.
3. The *configure* button is used to access the RTDX Configuration Control Properties page.
4. In the Buffer Size (in bytes) field, the desired buffer size is specified.
5. In the *Number of Buffers* field, the desired number of buffers is entered. By default, the number of buffers is set to 4, which is the minimum. With a multiprocessor configuration, the total number of buffers must be equivalent to or greater than the total number of processors being used with RTDX. RTDX requires a unique buffer for each processor.
6. For configurations to take effect, click *OK*.
7. In the RTDX-Configuration Control window, click the Enable RTDX checkbox to enable RTDX.
8. Build the project.

Another approach is to use TI CCS embedded in NI LabVIEW, which provides full control over the target capabilities. Here a dll is created given the newer versions of CCS (starting from CCS v4.1), which then

can be called in NI LabVIEW using the “Call Library Function Node” functionality that in turn eliminates the use of RTDX communication link. Table 4 summarizes these techniques and ranks their implementation complexity.

Table 4: Summary of the Methods

Tool	Functionality	Development Complexity
NI LabVIEW DSP Module	<ol style="list-style-type: none"> 1. Limited Functionality 2.No Access To the External Memory of the board for more complex algorithm development 3. NI LabVIEW Profiling Tools disabled – only benchmarking is possible 	Low
NI LabVIEW Test Integration Toolkit	<ol style="list-style-type: none"> 1. Enables functionality of TI CCS along with NI LabVIEW environment 2. Requires knowledge of prior CCS development 3.Currently officially not supported by NI 	Medium
TI Code Composer Studio embedded in LabVIEW	<ol style="list-style-type: none"> 1.Full access to the board memory 2. Full functionality of NI LabVIEW 3. Requires knowledge of prior CCS development 	High

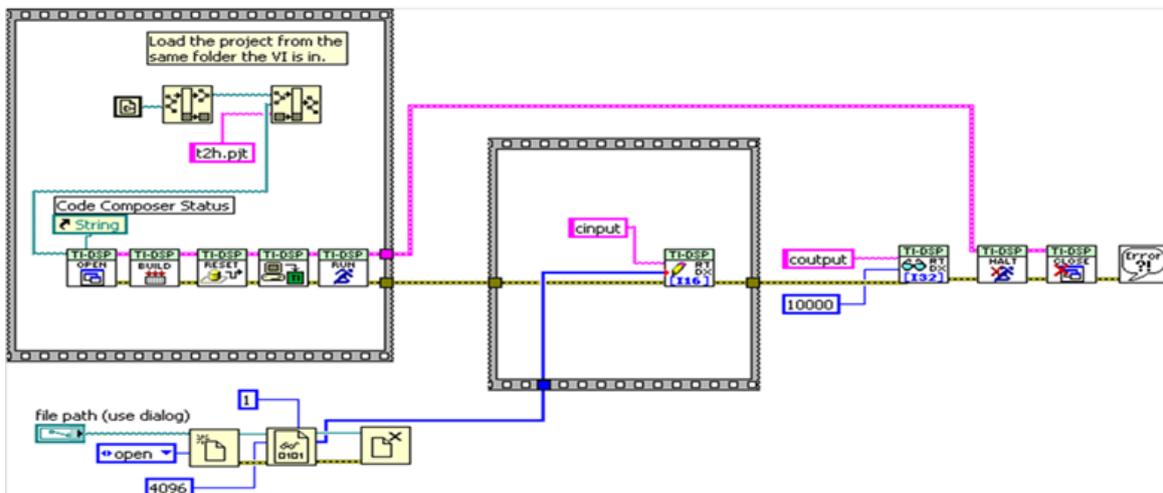


Figure 12: Acquisition algorithm implementation fragment in LabVIEW environment using RTDX streaming to get the FFT of the input signal.

Table 5 shows the timing performance on different platforms for the advanced acquisition algorithm implemented for 24 satellites with the sampling rate of 2.048 MHz. The coherent integration length is 1ms. The code is not fully optimized and mainly depends on the programming platform capabilities (e.g., multithreading, multicore operation, CPU speed). Fig. 13 provides a sample of profiling results on how timing statistics are collected in Table 5. Several implementation options are considered for the analysis. First, an acquisition algorithm is implemented in C/C++. The code is not optimized, and accelerations are only due to a fast block processing algorithm. The algorithm completes in 0.25 seconds. Then the algorithm is implemented as a dll in MS Visual Studio C++ and called in NI LabVIEW using “Call Library Function Node” functionality. This is done for comparison purposes to check the overhead that LabVIEW environment may introduce over C/C++ only implementation. The algorithm runs in 0.28 seconds. The timing performance is critical for software receiver development, and one can see that LabVIEW is not adding significant overhead over the C code implementation, about 10% in this particular scenario.

Table 5: Advanced Acquisition Algorithm Testing

Platform	Acquisition Time (sec)	FFT Time (sec)
Stadalone C/C++. MS Visual Studio 2008	0.25	0.016
LabVIEW+DSP. Single Precision floating point radix-2 FFT with complex input function called in NI LabVIEW using RTDX	0.031	0.001
Implementation using native LabVIEW blocks	0.058	0.004
Calling dynamic linked library in NI LabVIEW created with MS Visual Studio C++	0.28	0.016

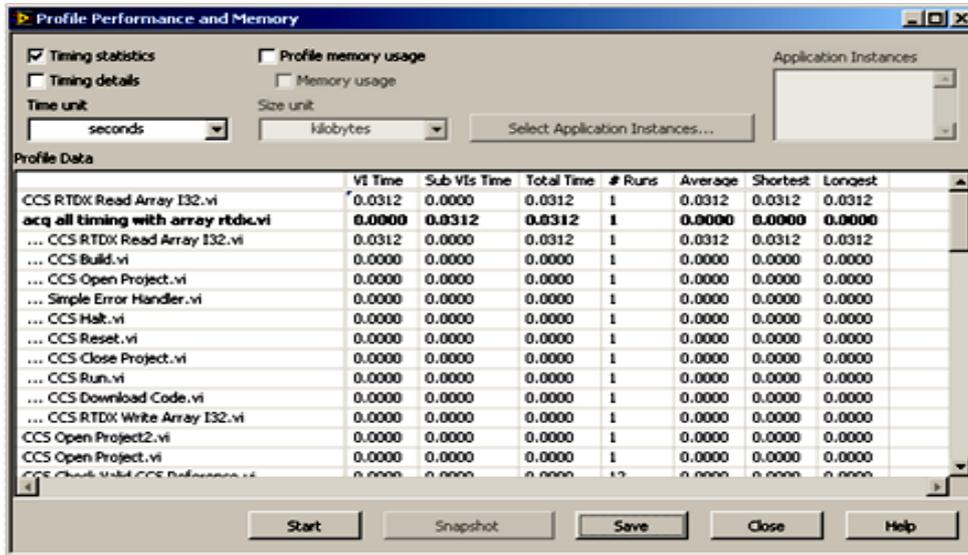


Figure 13: Profiling results for the target compilation timing statistics.

Then, the acquisition algorithm is implemented using LabVIEW native blocks. Interestingly enough the performance improves as the blocks are optimized and all other LabVIEW acceleration factors apply. The algorithm runs in 0.06 seconds, accelerating almost 5 times. When delegating the acquisition algorithm from LabVIEW to DSP peripheral the overall runtime of the .vi that performs the acquisition algorithm in CCS IDE using a bidirectional RTDX communication link is about 0.03s, which is twice as fast as the same algorithm implemented on LabVIEW only. The operation must be even faster, but an RTDX communication link is quite

slow when transferring big arrays of data. In all scenarios, the results achieved in these case studies are exceeding by far the performance of the acquisition algorithm implemented on the C6713DSP through Simulink (Hamza et al., 2009). In (Hamza et al., 2009) the result for the multiple satellite acquisition for the sampling rate of 4.092MHz and the intermediate frequency of 2.046MHz gives the maximum performance of about 17s. The same acquisition algorithm implemented in Matlab by Borre and Akos (2006) takes 183s having the intermediate frequency of the incoming signal as 9.548MHz and the sampling rate

as 38.192MHz. Table 5 shows also the FFT times taken on different platforms. It shows that the performance of the DSP target might be significantly higher if not slowness of RTDX communication.

Table 6 shows the timing statistics with a varying number of coherent integration lengths on the selected platforms when the sampling rate is still 2.048MHz. The LabVIEW application is performing faster according to the tests and still it's not adding an overhead to the same algorithm implemented in MS Visual Studio and called in LabVIEW as a dll.

Table 6: Advanced Acquisition Algorithm Testing (NI LabVIEW 2010 is used)

Platform	Input Signal duration (ms)	Time (sec)
Standalone C/C++. MS Visual Studio 2008	1	0.25
	8	1.25
	16	2.375
Implementation using native LabVIEW blocks	1	0.058
	8	0.47
	16	0.94
Calling dynamic linked library in NI LabVIEW created with MS Visual Studio C++	1	0.28
	8	1.188
	16	2.422

A PC with Intel(R) Core™ i5 CPU M580, 2.67GHz (4CPUs), 3510MB RAM, Windows XP Professional OS is used as a GPS simulator and A-GPS generation platform. NI LabVIEW 2010 version is used as a development environment along with GPS Simulation Toolkit 2.0. MS Visual Studio 2008 Team Suite is used for creating a dll to communicate with LabVIEW. Performance evaluation of the system in the LabVIEW environment is done based on (LabVIEW, 2012).

6. LabVIEW GPS/GNSS Receiver Testing with A-GPS Support

An assisted GPS concept facilitates a GPS receiver operation in a weak signal environment. It is standardized for all wireless networks (Zhao, 2002). As is mentioned in Section 3, one can integrate Labview-based A-GPS with the NI GPS Simulation Toolkit. For the case study of this paper, the implementation follows the guidelines of Secure User Plane Location (SUPL) architecture (Open Mobile Alliance, 2007) for a mobile-base network-assisted scenario. It is assumed that the receivers are equipped with wireless communication capability to receive assistance from a network. In one of the possible configurations, this communication is Internet Protocol (IP)-based to deliver assistance information through a User Plane bearer between a SUPL Enabled Terminal (SET), such as a mobile device, and a SUPL Location Platform (SLP) server. While a detailed description of an assistance data delivery solution can be found in (Narisetty et al., 2012), the following describes a general setup.

Figures 14-16 illustrate the complete experimental setup to test user devices for A-GPS support. First, the NI GPS Simulation Toolkit (NI LabVIEW, 2012) generates GPS binary navigation data for all selected satellites based on user-defined 'location' and 'time' along with almanac files in SEM format (Almanac information, 2012) and the ephemeris files in RINEX 2.0 format (Ephemeris information, 2012). These data are used to generate GPS signals. In our implementation, the simulator is co-located with the A-GPS SUPL server (SLP). The above-mentioned binary navigation data is also provided to an A-GPS SUPL server (SLP), which encapsulates it into textual assistance files and communicates them to receivers following SUPL-defined procedure through the wireless link. Client/Server communication through the wireless data link is implemented in Java as described in (Narisetty et al., 2012).

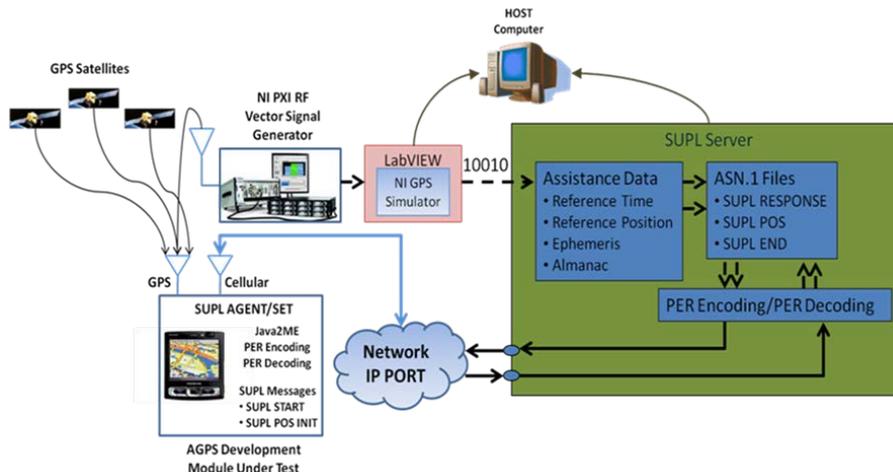


Figure 14: Assistance data generation and flow.

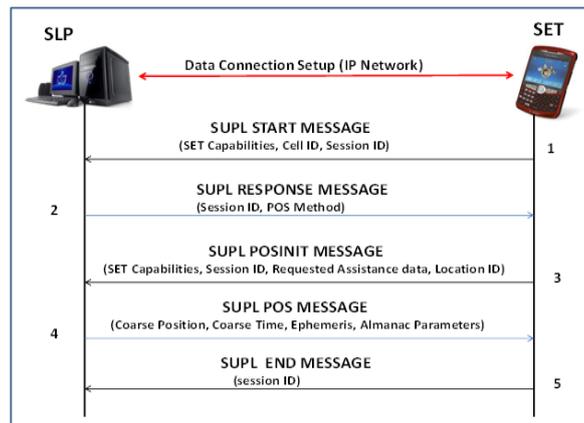


Figure 15: SUPL message flow (SET initiated)

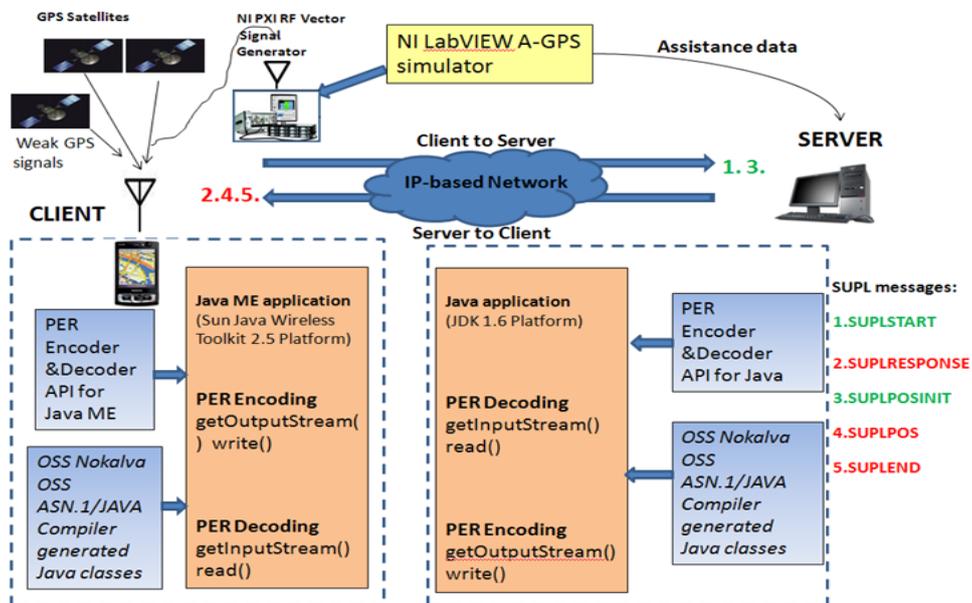


Figure 16: A block diagram of experimental setup for testing A-GPS support (Narisetty et al, 2012)

Along with the orbital parameters (ephemeris and almanac) the assistance also includes ‘reference position’ and ‘reference time’ information. The generation of this data is quite straightforward as the simulator internally possesses the accurate time and location of the user. So one can generate these references by user-defined offsets. The reference location can be alternatively retrieved using a Cell-ID location provided by application programming interfaces (API) of mobile operational systems. Another alternative is to retrieve the reference location using wireless network addresses such as WLAN MAC-IDs and IP addresses and existing databases of network address locations. Details on these alternatives are presented in (Narisetty et al., 2012).

The SUPL textual files are created in Abstract Syntax Notation (ASN.1) as described in (UniGone, 2012). There are five messages going back and forth between the SET and the SLP in the SUPL architecture as shown in Fig. 15. Whenever an application running on the SET requests for position, the SUPL agent on the SET sets up a secure IP connection with the SLP and initially sends the start message to the SLP (SUPLSTART), which contains the user position technology, preference method and position protocol. The SLP replies with the SUPLRESPONSE in ASN.1 format including the session-ID and the positioning method to the SET as a response message. The SET then initializes the position session by requesting for the assistance data sending a SUPLPOSINIT message to the SLP consisting of supported positioning methods and associated

positioning protocol. The assistance data (SUPLPOS message) is delivered to the SET by the SLP wrapped in the form of a RRLP payload. As long as the SET receives the orbital assistance message (SUPLPOS), it can proceed with the calculation of the coarse position based on the estimated assistance data from the SLP. Once the SUPLPOS message is received the SLP informs the SET to end the IP connection by sending a SUPLEND message for releasing resources related to the location session (Open Mobile Alliance, 2007), (Chayapathy et al., 2009).

For wireless communication, the textual files are encoded in Unaligned Packed Encoding Rules (PER) (T-Rec-X, 2012) for minimal encoding size using OSS NOKALVA™ runtime libraries and OSS NOKALVA™ ASN.1 TO JAVA COMPILER (OSS Nokalva, 2012) as shown in Fig. 16.

Without A-GPS support there is a need to collect data for at least three first subframes out of five (6 seconds each), which is sufficient for decoding a navigation data fragment for position calculation. However, with the A-GPS support, we already have the almanac and ephemeris information decoded; thus, we can proceed to the position calculation immediately. The GPS receive should still track the signals to collect code phase measurements and preferably detect time stamp locations.

7. Conclusion

The paper describes an integrated LabVIEW-based platform for GPS/GNSS receiver development and testing using a simulator, hardware accelerators and A-GPS support. It is described how to delegate computing tasks to a DSP peripheral. An advanced acquisition algorithm is tested on C6713 DSP target platform using RTDX channel communication. A LabVIEW-based solution with the DSP target peripheral accelerates computations significantly compared to the Simulink GPS receiver developed on the same target (Hamza et al., 2009). Comparison the GPS receiver implementations using NI LabVIEW and MS Visual Studio C++ platforms is provided where it is shown that due to NI LabVIEW platform's inherent optimization capabilities and embedded multithreading, the same algorithm implementation provides better performance and the overhead is insignificant while embedding the C++ dll into the LabVIEW environment.

Acknowledgments

The authors would like to express their sincere gratitude to Texas Instruments for providing the TMS320C6713 DSP Starter Kit (DSK) which made the design and implementation of this work possible.

References

- Agarwal N. et al (2002), *Algorithms for GPS operation indoors and downtown*, GPS Solutions., Springer-Verlag Heidelberg, Vol. 6, No. 3, pp. 149-160.
- Akopian D. (2005), *Fast FFT based GPS satellite acquisition methods*, Proceedings of IEE, Vol. 152, No. 4, pp. 277-286, 2005. Initial version presented at ION-GPS-2001 Conference, Sep. 11-14, 2001, Salt Lake City, UT.
- Akopian D., A. Soghoyan, S. Chayapathi, GVS Raju (2011), *A Flexible LabVIEW-based GNSS Receiver Development and Testing Platform*, ION-ITM-2011 Conference, Jan. 24-26, 2011, San Diego, CA, pp. 1270-1280
- Akos D. M., P. Normark, A. Hansson, A. Rosenlind, C. Stahlberg, and F. Svensson (2001), *Global Positioning System Software Receiver (gpSrx) Implementation in Low Cost/Power Programmable Processors*, Proc. 2001 ION GPS Conf., Sept. 2001, Salt Lake City, UT, pp. 2851-2858.
- Akos D. (2003), *The role of Global Navigation Satellite System (GNSS) software radios in embedded systems*, *GPS Solutions*, 2003, Springer-Verlag, Vol. 7, No. 1: pp.1-4.
- Almanac information.* <http://www.navcen.uscg.gov/> (Accessed May 7, 2012)
- Benveniste R., B. Sirmacek, S. Unsalan (2010), *A quick start to the Texas Instruments TMS 320C6713 DSK*
- Borre K., D. M. Akos, N. Bertelsen, P. Rinder, S.H. Jensen (2006), *A Software-Defined GPS and Galileo Receiver: A Single-Frequency Approach*. Birkhauser Boston.
- Cetin E., I. Kale, R. Morling (2007), *Analysis and compensation of RF impairments for next generation multimode GNSS receivers*, Proceedings of the IEEE International Symposium on Circuits and Systems IEEE, May 2007, New Orleans, LA, pp. 1729-1732.
- Chassaing R., D. Reay (2008), *Digital Signal Processing and Applications with the TMS320C6713 and TMS320C6416 DSK*, John Wiley & Sons, Inc., Hoboken, New Jersey.
- Chayapathy S.N., A.Kumar, P.Kashyap, D.Akopian, A. Samant (2009), *A SUPL-based A-GPS Simulator Support for Indoor Positioning*, Proc. of the 22nd Int. Technical Meeting of The Satellite Division of

the Institute of Navigation (ION GNSS), Savannah, GE, Sept. 22-25, 2009, pp. 503-515.

Code Composer Studio Development Tools v3.3 Getting Started

Guide <http://www.ti.com/lit/ug/spru509h/spru509h.pdf> (Accessed May 6, 2012)

Developing an OFDM Transmitter and Receiver System Using LabWindows/CVI and PXI

<http://sine.ni.com/cs/app/doc/p/id/cs-12714> (Accessed May 7, 2012).

Ephemeris

information. http://cddis.gsfc.nasa.gov/gnss_dataum.html#brdc (Accessed May 7, 2012).

Fastrax GPS receivers and SDKs, <http://fastraxgps.com/> (Accessed May 7, 2012).

Girau G., A. Tomatis, F. Dovic, P. Mulassano (2007), **Efficient Software Defined Radio Implementations of GNSS Receivers**, Circuits and Systems, ISCAS 2007, IEEE International Symposium, May 2007, New Orleans, LA, pp.1733 - 1736.

GPS Multiple-Satellite Signal Generation, http://www.ni.com/pdf/products/us/cat_gpstollkit.pdf, (Accessed: Oct. 1, 2012)

GPS Networking, <http://gpsnetworking.com/standard-line-amplifiers.asp> (Accessed May 7, 2012)

GPS Receiver Testing, <http://www.ni.com/white-paper/7189/en> (Accessed Sept. 25, 2012)

Hamza G. G., A. A. Zekry, M. N. Moustafa (2009), **Implementation of a Complete GPS Receiver on the C6713 DSP through Simulink Receiver**, Journal of Global Positioning Systems, Vol.8, No.1 : 76-86

Heinrichs G., M. Restle, C. Dreischer, T. Pany (2007), **NavX®- NSR – A Novel Galileo/GPS Navigation Software Receiver**, ION GNSS International Technical Meeting of the Satellite Division, Fort Worth, TX, Sept. 25-28, 2007, pp. 1329-1334.

Internet Protocol Suite,

http://wiki.ask.com/Internet_protocol_suite (Accessed: Sept. 25, 2012)

Kaplan E.D. (1996), **Understanding GPS: Principles and Applications**, Boston: Artech House.

Kashyap P., A. Samant, P. Sagiraju, D. Akopian (2009), **An A-GPS Support for GPS simulators for**

Embedded Mobile Positioning, Proc. of SPIE Multimedia on Mobile Devices, Electronic Imaging, Jan. 18-22, 2009.

LabVIEW, GPS Simulator, RF Signal analyzers and generators from National Instruments, <http://www.ni.com/> (Accessed May 7, 2012)

Ma C., G. Jee, G. MacGougan, G. Lachapelle, S. Bloebaum, G. Cox, L. Garin, J. Shewfelt (2001), **GPS Signal Degradation Modeling**, Proceedings of the 14th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GPS 2001), Salt Lake City, UT, Sept. 2001, pp. 882-893.

Matlab from Mathworks at www.mathworks.com. (Accessed May 5,2012)

Misra P., P. Enge (2001), **Global Positioning System, Signals, Measurements, and Performance**. Ganga-Jamuna Press, Lincoln, MA.

Narisetty J., A. Soghoyan, M. Sundaramurthy, D. Akopian (2012), **SUPL support for mobile devices**, Multimedia on Mobile Devices 2012, Proceedings Vol. 8304, Feb. 2012

NI LabVIEW DSP <http://www.ni.com/pdf/manuals/371581c.pdf> (Accessed May 7, 2012)

NI LabVIEW Test Integration Toolkit, <http://www.ni.com/pdf/manuals/323452b.pdf> (Accessed May 7, 2012)

NI LabWindows/CVI , <http://www.ni.com/twcv/> (Accessed Jan. 13, 2011)

Open Mobile Alliance. UserPlane Location Protocol v1.0, Open Mobile Alliance_, OMA-TS-SUPL-V1_0, 2007.

Open source GPS software radio - GPS-SDR, multithreaded enabled C++ application, <http://www.ctae.org/sdr/doc/html/index.html> (Accessed May 7, 2012).

OSS Nokalva, www.oss.com (Access: May 20, 2012)

Prototyping Algorithms for Next-Generation Radio Astronomy Receivers Using PXI-Based Instruments and High-Speed Streaming <http://sine.ni.com/cs/app/doc/p/id/cs-12972> (Accessed May 7, 2012)

Sagiraju P. K., P. Kashyap, D. Akopian (2008), **Block correlator for tracking GPS/GNSS Signals**, ION-GNSS-2008 Conference, Sep. 16-19, 2008, Savannah, GE, pp. 229-235.

Soghoyan A., G. Huang, J. Narisetty, D. Akopian (2011) **A LabVIEW-Based Assisted GPS Receiver Development, Simulation and Testing Platform**, Proceedings of the 24th International Technical Meeting of The Satellite Division of the Institute of Navigation (ION GNSS 2011), Sept. 20 - 23, 2011, Oregon Convention Center, Portland, pp. 1982-1997

Texas Instruments, www.ti.com, **TMS320C6713 DSP**, <http://www.ti.com/tool/tmdsdsk6713>, (Accessed Oct. 23, 2011)

Third Generation Partnership Project 3GPP TS 44.031; Technical Specification Group GSM/EDGE Radio Access Network; Location Services (LCS); Mobile Station (MS) – Serving Mobile Location Centre (SMLC) Radio Resource LCS Protocol (RRLP), 2010.

TMS320C6000 Code Composer Studio Tutorial, <http://www.ti.com/lit/ug/spru301c/spru301c.pdf>, (Accessed May 7, 2012)

T-REC-X.691 -2008 ASN.1 Encoding Rules: Specification of Packed Encoding Rules (PER) at <http://www.itu.int/ITU-T> (Accessed Sept. 25, 2012).

WAAS Test Team, <http://www.nstb.tc.faa.gov/DisplayNSTBDataDownload.htm> (Accessed, Sept. 25, 2012)

Winternitz L., M. Moreau, G. J. Boegner Jr. and S. Sirotzky (2004), **Navigator GPS Receiver for Fast Acquisition and Weak Signal Space Applications**, ION GNSS 2004, Sep 21-24, 2004, Long Beach, CA, pp. 1013-1026.

UniGone ASN.1 Solutions, <http://www.unigone.com/en/solutions/asn1> (Access: May 20, 2012)

Wireless E911 location accuracy requirements: FCC mandate for Docket 94-102. <http://www.fcc.gov/e911>. (Accessed May 7, 2012).

Zhao Y. (2002), **Standardization of Mobile Phone Positioning for 3G Systems**, IEEE Communications Magazine, July 2002, pp. 109-116.

Biography

Arpine Soghoyan - received her B.Sc./M.Sc. degrees in Radiophysics and Electronics department of the Yerevan State University, and M.Sc. in Computer and Information Science at the American University of Armenia. Currently she is a Ph.D. student in the department of Electrical and Computer Engineering at the University of Texas at San Antonio. Her research focuses on software defined radio, fast prototyping and testing of hybrid hardware/software GPS receivers.