Parallel Software Architecture and Algorithms for GNSS Signal Simulation

Mgr. Iva Bartůňková

Vollständiger Abdruck der von der Fakultät für Luft- und Raumfahrttechnik der Universität der Bundeswehr München zur Erlangung des akademischen Grades eines Doktors der Ingenieurwissenschaften (Dr.-Ing.) genehmigten Dissertation.

Gutachter/Gutachterin

Vorsitzender: Prof. Dr.-Ing. Roger Förstner

1. Berichterstatter: Prof. Dr.-Ing. habil. Bernd Eissfeller

2. Berichterstatter: Prof. Letizia Lo Presti

Die Dissertation wurde am 10.6.2016 bei der Universität der Bundeswehr München eingereicht und durch die Fakultät für Luft- und Raumfahrttechnik am 17.6.2017 angenommen. Die mündliche Prüfung fand am 11.7.2017 statt.

Zusammenfassung

Für die Entwicklung sowie für die Zertifizierung jedes Satellitennavigationsempfängers sind Spezielle Tests und Verifikationen mittels realistischer GNSS-Signale notwendig. Prinzipiell bieten sich im GNSS-Bereich Feldtests zur Evaluierung von Empfängern an, dieses einfache Verfahren birgt jedoch auch eine Reihe von Nachteilen: So ist das Auftreten unterschiedlicher Kombinationen von GNSS-Signalen nicht kontrollierbar oder reproduzierbar. Einen weiteren Nachteil stellt die fehlende Möglichkeit dar, immer wieder dieselben Signale oder Empfangsszenarien zu reproduzieren. Gleichzeitig ist das Durchführen von Feldtests sehr teuer und mit viel Aufwand verbunden. Daher ist es von besonderer Bedeutung für die Validierung und Tests von GNSS-Empfängern, GNSS-Signale mittels spezieller Geräte unter kontrollierten und wiederholbaren Laborbedingungen zu simulieren. GNSS-Simulatoren dienen daher als Testgeräte während des gesamten Entwicklungs- und Lebenszyklus eines Satellitennavigationsempfängers. Preiswerte Mehrfrequenz- und Mehrsystem-GNSS-Simulatoren tragen zusätzlich zur Konkurrenzfähigkeit der modernen Mehrfrequenz- und Mehrsystem- GNSS-Empfänger bei.

Die Architektur der herkömmlichen Mehrfrequenz-GNSS-Simulatoren basiert auf der Generierung jeweils eines einzelnen Frequenzbandes in digitaler Form mittels sog. Field Programmable Gate Arrays (FPGAs) oder digitalen Signalprozessoren (DSPs). Hierauf folgt die einzelne Konvertierung zu den jeweiligen analogen Signalen und das Mischen aller simulierten Signale in die Zielfrequenzbänder. Die Kosten für die speziellen Hochleistungs-FPGAs oder DPSs und teure Hardware für das hochpräzise und hochsynchrone Mischen der analogen Signalbänder tragen zu den hohen Gesamtkosten dieser Simulatoren bei.

In dieser Arbeit wird eine alternative breitbandige Architektur eines Echtzeit- und Mehrfrequenz-GNSS-Signalsimulators erprobt und evaluiert. Die Signalgenerierung ist auf einem Grafikprozessor implementiert. Das digitale Mehrfrequenz-GNSS-Signal wird als ein einzelnes Signalband mit der Bandbreite des ganzen GNSS-Frequenzbereiches generiert. Das breitbandige Signal wird mit einem breitbandigen Digital-zu-Analog Wandler in ein analoges Signal umgewandelt und kann ohne das problematische analoge Zusammenführen und Synchronisieren in das Zielfrequenzband hochgemischt werden.

Eine spezifische und anspruchsvolle Aufgabe der realistischen GNSS-Signalsimulation ist die Generierung von Mehrwegeeffekten. Für eine realistische Modellierung wird eine Vielzahl an Kanälen mit reflektierten Signalen benötigt. Kanalmodelle für Mehrwegeeffekte sowohl in Outdoor- als auch in Indoor- Umgebungen wurden ausgewählt und in einem Softwaresignalsimulator integriert. Hierfür wurde ein Algorithmus für die beschleunigte Generierung der Mehrwegekanäle entworfen und implementiert. Im Anschluss wurden die simulierten Signale mit den Ergebnissen eines Feldtests verglichen und evaluiert.

Im Rahmen dieser Arbeit wurde ein Testsimulator mit breitbandiger Architektur aufgebaut. Dieser setzt sich aus einem leistungsstarken aber kostengünstigen PC und zwei Grafikkarten aus dem Bereich der Unterhaltungselektronik sowie einem DAC-Wandler zusammen. Für die Signalgenerierung und GNSS-Signalmodulierung wurde hochparallele Algorithmik entworfen, welche die hohe Zahl an Prozessorkernen sowie die hierarchische Speicherstruktur der Grafikkarten einsetzen kann. Die Qualität der generierten digitalen Signale und der Datendurchsatz der Algorithmen wurden analysiert. Die Leistungstests bewiesen eine Echtzeitleistung des Simulators für eine breitbandige Generierung,

die alle derzeitigen GNSS-Frequenzbänder einschließt. Hierzu wurden GPS L1 C/A, Galileo E1 OS, E5a und E5b mit jeweils bis zu zwölf Signalkanälen simuliert.

Die digital-zu-anlog Konvertierung wurde für einzelne Signalbänder verifiziert. Die notwendigen Anforderungen an einen passenden breitbandigen DAC-Wandler sowie an einen passenden Hoch-Konverter wurden analysiert und die Verfügbarkeit von entsprechenden Produkten auf dem Weltmarkt nachgewiesen. Zur tatsächlichen Einbindung solcher Komponenten in die Architektur wären noch bedarfsgerechte Anpassungen notwendig.

Abstract

Testing and verification with real and simulated GNSS signals is an essential part of all stages of the lifecycle of a GNSS receiver as well as of the lifecycle of a GNSS-based navigation system. The true GNSS signals can be obtained outdoors in a field test. Disadvantages of the field testing are the lack of repeatability, the logistics costs and the hardship to obtain true signal parameters. For receiver verification, also scenarios that are difficult or expensive to obtain in a field test (scintillation, flight scenario, signal in space, etc.) and scenarios that are impossible to encounter (satellite failure, future signals, selective availability, etc.) must be tested. These drawbacks can be solved by a GNSS signal simulator that synthetizes the GNSS signals according to user specification of system, signal and receiver environment features. The simulator is an indispensable tool in the whole receiver development and production lifecycle. The costs of these devices influence the total costs of the navigation solution development.

High performance multi-GNSS multi-frequency signal simulators that are able to generate signal for scenarios with hundreds of signal channels and multiple frequency bands are required for testing of modern multi-frequency receivers. These devices are costly. This is partly caused by the costs of multiple high-performance digital signal generation engines (FPGAs or DSPs). Another factor is the costs of the hardware for synchronized mixing and up-conversion of the separate signal frequency bands.

In this thesis, an alternative low-cost GNSS signal simulator architecture is proposed. The digital GNSS signal is generated on a pair of graphics processing units (GPU) as a single broadband signal that includes GNSS signal services spreading from L1 to L5 band. This signal is then converted to analog using a single broadband DAC and up-converted to L-band by a broadband upconverter. The simulation definition part describing all signal channels in terms of frequency, phase and amplitude of each epoch can be built up in a conventional way.

A specific issue of the high-performance GNSS signal simulation is the generation of multipath. For a realistic representation of a multipath scenario, numerous additional signal channels of the multipath reflections need to be generated. In this work, satellite-to-indoor channel characterization with a sequence of models is proposed. An algorithm for fast multipath signal generation was designed and implemented. A comparison of simulated multipath with a field test scenario is given.

A prototype of the digital GNSS signal generation module of the simulator was implemented on a test system with low-cost components. Consumer-level GPUs were combined with a mass-market PC system, both designed for the needs of gaming industry. The signal synthesis algorithms and GNSS signal generation concepts were designed for parallel execution on hundreds of processor cores of the GPUs and for utilization of the specific GPU memory hierarchy. Issues of digital signal precision and algorithm throughput with respect to GPU architecture and native instructions were analyzed. The benchmark tests showed that real-time performance of the broadband digital signal generation was reached for simultaneous generation the four implemented GNSS services: GPS L1 C/A, Galileo E1 OS, Galileo E5a and E5b.

The digital-to-analog conversion was implemented in narrowband. Requirements on broadband digital-to-analog converter and upconverter to L-band were analyzed and availability of tailored solution was verified for a possible future deployment.

Contents

List of Acronyms

ADC	Analog-to-digital converter
AltBOC	Alternate BOC
ARNS	Aeronautical radio navigation systems
ASIC	Application-specific integrated circuit
BPSK	Binary phase shift keying
CC	CUDA Computing Capability
CPU	Central processing unit
CRPA	Controlled reception pattern antenna
CUDA	Computing Unified Device Architecture
DAC	Digital-to-analog converter
DAS	Direct analog synthesis
DDR	Double data rate
DDS	Direct digital synthesis
DIF	Digitized intermediate frequency
DLL	Delay lock loop
DP	Double precision
DSP	Digital signal processor
ECEF	Earth-centered earth-fixed
EGNOS	European Global Navigation Overlay System
ENU	East-north-up
FE	Front-end
FLL	Frequency lock loop
FPGA	Field programmable gate array
FPGA	Field programmable gate array
GAGAN	GPS-Aided Geo-Augmented Navigation
GB	Gibabyte
GLONASS	Globalnaya Navigazionnaya Sputnikovaya Sistema
GNSS	Global navigation satellite system
GPS	Navstar Global Positioning System
GPU	Graphics processing unit
Gsps	Giga sample per second
HH	Hand-held
HW	Hardware
ICD	Interface control document
IF	Intermediate frequency
INS	Inertial navigation system
IRNSS	Indian Regional Navigation Satellite System
ISTA	Institute of Space Technology and Space Applications at Universität der Bundeswehr München
KB	Kilobyte
LD/ST	Load / store
LMS	Land Mobile Satellite model

LNA	Low noise amplifier
LOS	Line of sight
MB	Megabyte
MSAS	Multi-Functional Satellite Augmentation System
MSB	Most significant bit
Msps	Mega sample per second
NCO	Numerically controlled oscillator
NMEA	National Marine Electronics Association
NRZ	Non-return-to-zero
OS	Open service
OTAR	Over-the-air rekeying
PAC	Phase-to-amplitude converter
PC	Personal computer
PCIe	Peripheral Component Interconnect Express
PLL	Phase lock loop
PRN	Pseudorandom noise (used for spreading code of CDMA-based GNSS signals)
PVT	Position, velocity and time
QPSK	Quadratur Phase Shift Keying
QZSS	Quazi-Zenith Satellite System
RF	Radio frequency
RMS	Root mean square
RNSS	Radio navigation satellite systems
RP	Replay
SAASM	Selective availability anti-spoofing module
SATA	Serial AT Attachement
SBAS	Satellite-based augmentation system
SFDR	Spurious free dynamic range
SFU	Special function unit
SIMD	Single-instruction multiple-data
SMX	Streaming multiprocessor (of a GPU)
SNR	Signal-to-noise ratio
SR	Sample rate
SS	Signal simulator
SV	Space vehicle
SW	Software
TOW	Time of week
TTFF	Time to first fix
USB	Universal Serial Bus
WAAS	Wide Area Augmentation System
ZOH	Zero-order hold

List of Figures

List of Tables

1 Introduction

1.1 GNSS Signal Simulation – Motivation

Testing and verification with real and simulated GNSS signals is an essential part of all stages of a GNSS receiver lifecycle as well as of a lifecycle of a complete navigation system. Additionally, testing and verification is a key element on the way to the standardization and certification. It ensures correctness of outputs and assesses if and to which extent the desired criteria are fulfilled in terms of reliability, efficiency, portability, maintainability, and usability.

The GNSS signals can be obtained simply outdoors in a field test. Such signals are the only fully realistic ones without the limitations inherently caused by limited features of any generating device. Nevertheless, the lack of repeatability and the logistics costs for field testing create a market for specialized solutions. The simplest solution is offered by so-called record and playback devices. Such devices record the GNSS signals during a field test campaign, store it in digital form and offer a playback of signal in labor at any time. The playback differs from the original signal to an extent given by the abilities of the recording and storage device. However, advantages can be taken from the repeated playback function, since all repeated playbacks provide identical GNSS signals.

The knowledge of true signal parameters of verification signals is important for a crosscheck with receiver measurements. Furthermore, the ability to create a scenario that is in field test hard and expensive to obtain (scintillation, flight scenario, signal in space, etc.) and to generate signals for a scenario that is impossible to obtain in field test (satellite failure, future signals, selective availability, etc.) is needed. Neither the field-testing nor a record and playback device can fulfill these requirements. Therefore, a family of devices called GNSS signal simulators plays an important role in verification and testing of receivers. A GNSS signal simulator synthetizes the GNSS signals according to a user specification of system, signal and receiver environment features.

GNSS signal simulators are indispensable devices in the whole receiver development and production lifecycle. These devices are dedicated tools build up as a combination of software, hardware and analog radio technology. Costs of these devices influence the total costs of the navigation solution development.

1.2 Scope of Work

This thesis introduces an alternative GNSS signal simulator architecture. The digital GNSS signal is generated on a graphic processing unit (GPU) as a single broadband signal that can include GNSS signal services from L1 to L5 band. This single broadband signal is then converted to analog using a single DAC and upconverted to L-band by a broadband upconverter. The simulation definition part describing all signal channels in terms of frequency, phase and amplitude of each epoch can be built up in a conventional way.

The focus of this thesis is the digital GNSS signal generation module of the simulator. Standard signal synthesis algorithms and GNSS signal generation concepts are redesigned for parallel execution on hundreds of processor cores of GPUs and for utilization of the specific GPU memory hierarchy. Issues

of digital signal precision and algorithm throughput with respect to GPU architecture and native instructions are analyzed.

The digital GNSS signal generation module was implemented using consumer-level GPUs and a respective PC system, both coming from the field of gaming industry. The benchmark tests showed that real-time performance of the broadband digital signal generation was reached for simultaneous generation of four services GNSS services, GPS L1 C/A, Galileo E1 OS, E5a and E5b.

The digital-to-analog conversion was implemented in narrowband. Requirements on broadband digital-to-analog converter and upconverter to L-band were analyzed and availability of tailored solution was verified for a possible future deployment.

Additionally, the topic of multipath GNSS signal generation in a software signal simulator is included in the thesis. The satellite to indoor channel is characterized with a sequence of models. Algorithm for fast multipath signal generation on a CPU is proposed and comparison of simulated multipath with a field test scenario is given.

1.3 Receiver Testing

In order to clarify the expectations on GNSS simulators, details about receiver testing procedures, receiver-testing tools and the role of the simulators are given in this section. The subsection about receiver testing procedures is based on [1] and the subsection about receiver testing tools is based on [2]. To the information originating from these sources, this work adds details about the implied expectations on GNSS simulators.

Revenues on the market for GNSS receivers are the key value influencing the range of acceptable costs of a simulator. The price of a receiver can vary tremendously: it ranges from consumer receiver chips with costs around 10 $ to geodetic receivers with price exceeding 10,000 $. Costs of airborne and spaceborne receivers exceed even this value significantly.

The number of units sold follows the opposite pattern. According to the GNSS Market Report 2015 [3], 60 million road units, 1,500 million GNSS-enabled smartphones (equipped with low-end chips) and 250,000 surveying devices (equipped with high-end receiver modules) were expected to be sold in 2015. These assumptions imply the market value of 18.1 million dollar for both low-end and top-end segment per year. This value determines subsequently the market size for receiver testing. A detailed view of the market segmentation for GNSS signal simulators and respective product classes is given in Chapter 2.

1.3.1 Approaches to GNSS Receiver Testing

Testing of a GNSS receiver requires tests of the software (platform tests, runtime tests, and memory tests), the hardware (power consumption) and tests of the analog radio frequency (RF) technology (electromagnetic compatibility, noise figure). Moreover, tests of the correctness, in particular the tests specific to GNSS-based positioning must be carried out.

For the software part of the receiver and software models of the hardware part, standard software testing approaches are applied. Key point of any testing concept is a test plan. As described in [2], a test plan for a GNSS receiver is an outline concept. For each phase of receiver lifecycle, it describes

specific milestones at which testing procedures should be carried out. It specifies also the parts of the software, which should be tested. Each testing procedure is given as a set of individual tests, each with defined inputs and expected correct outputs. Every test has to be documented within a corresponding report. It is important to set up such a test plan in advance to the receiver development.

The testing procedures are scaled according to the desired resolution. Consecutively, three different categories of the testing procedures can be identified:

- Black box: the test inputs and outputs access the tested system only through the end user interfaces.
- Grey box: the test inputs and outputs access individual modules of the tested system and/or use additional interfaces for this access.
- White box: tester has access to the source code and can add code that supports additional input and output.

The type of resolution used for testing procedures in testing loops varies according to the current development phase. If a new functionality is added or changed, white-box testing procedures are applied on the new code first. After the successful pass, grey-box procedures examine the whole module. Thereafter, a black-box testing procedure is applied on the whole system.

In the final verification and product support phase, a reverse order is applied. First, black-box testing procedures are applied on the whole system. If an error occurs, grey-box procedures verify all modules individually. Finally, error localization is carried out using white-box testing procedures.

The role of a GNSS simulator in receiver testing varies according to the phase of the GNSS receiver lifecycle. The phases of the lifecycle are ordered as follows:

- Research and development on new algorithms and solutions
- Receiver system design and validation
- Hierarchical element production
- Consumer evaluation and certification
- Repair and maintenance

The whole navigation system is then built up hierarchically, i.e. starting from a chip, following with the module, OEM board and finally with the complete navigation system integrated in the final user device. At higher levels not only output product must be tested, but incoming components need to be verified as well.

1.3.2 Receiver Testing Tools

The testing tools can be classified as tools generating receiver input data and tools processing the receiver final or partial output.

As mentioned before, input signals can be easily gained by field testing. Figure 1-1 gives a closer look at the application of this method to the receiver. The environment is native; the application of the signal is identical to the real case. Full black box testing of the complete receiver and navigation system can be carried out. The weak point is the source of reference signal parameters for verification. These can be obtained from a reference receiver with higher quality than the tested receiver or from a

GNSS monitoring network. As a further cross-check parameter, also the receiver position known from a geodetic measurement can serve.

Another way to get input signal is the usage of a signal-generating tool. These tools are GNSS signal simulators or record and playback devices.

GNSS signal simulators offer full repeatability of the signal generation and the ability to control the signal generation parameters. Key advantage is the generation of not yet implemented signals or rare special cases. Nevertheless, the signal parameters and the signal itself are not absolutely real and the range of settings to generate special cases is limited.

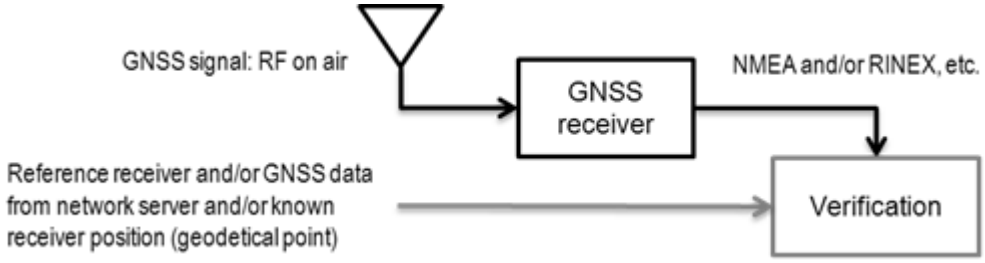

Figure 1-1 Field testing

A GNSS signal simulator is either a RF simulator, generating the analog RF signal as it would leave the antenna of a receiver. Alternatively, it is a digital intermediate frequency (DIF) generator generating the digital signal as it would leave the front end (FE) of the receiver. Details to these two simulator classes and their closer description are given in Chapter 2 GNSS Signal Simulators.

In case of RF simulators, the signal is input in the receiver by cable after the antenna. The antenna pattern and low noise amplifier (LNA) influence must be simulated and therefore cannot be completely real. However, these parameters can be directly used for verification and are fully known. Important is also the ability of the simulator to apply different true parameters and message parameters for a realistic testing of receiver behavior. An overview of the testing with signal simulator is given in Figure 1-2.

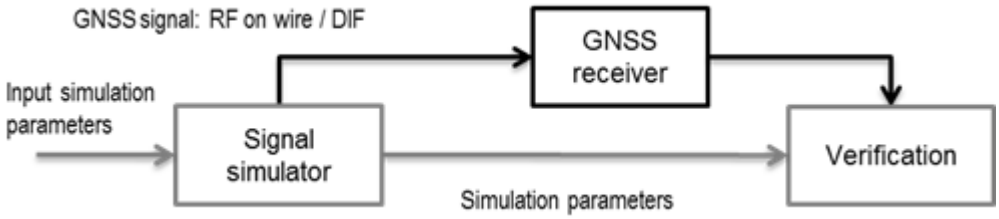

Figure 1-2 Signal simulators

Record and replay devices offer repeatability of the signal generation, real signal parameters, real but costly scenarios and the ability to control signal power. Nevertheless it is difficult to anticipate and impossible to control the signal parameters. The signal is modified through down-conversion, sampling, synthesis and up-conversion procedures.

As in case of a simulator, the generated signal is transmitted through a cable to the front end (FE) of the receiver. It is therefore important that the antenna of the simulator is identical to the antenna that was used for recording of the signal. The parameters of the front end of the replay device should fit the parameters of the front end of the tested receiver.

A special case of a record and playback device is a digital intermediate frequency (DIF) signal recorder. For recording, is uses the same front-end as the target receiver. Then, the digital signal is directed to a storage medium instead of being only processed by the receiver signal-processing module. An IF-replay function of the receiver loads the digital signal from the medium in the same way as in would have been received in real time. The DIF signal recorder is therefore receiver specific and it either comes to market as an add-on of a high-end receiver or it is an in-house tool developed for testing of own receiver products.

As in case of field testing, the signal parameters of a record and replay scenario are obtained directly from a reference receiver and/or from a GNSS monitoring network. An overview of the testing procedure using record and playback device is given in Figure 1-3.

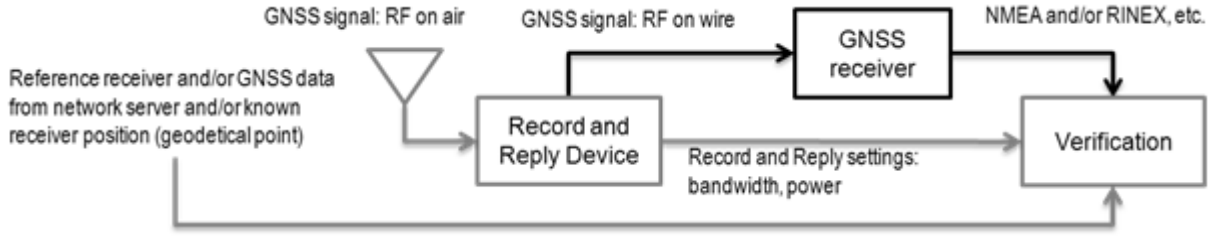

Figure 1-3 Record and playback devices

Beside the signal generating tools and field testing, also hybrid solutions are now wide spread in receiver verification. A typical example is a test bed with special ground based signal transmitting pseudolites. For example, the GATE test bed in Berchtesgaden, Germany, transmits Galileo signals from pseudolites placed on surrounding mountain peaks. A receiver under test receives both pseudolite and standard GNSS signals over an antenna. The signal transmitted from the static constellation of pseudolites is adjusted to position and speed of the tested receiver through a reference receiver to emulate realistic signal parameters of a satellite constellation. Therefore, the testing of the whole receiver system is possible similar to field testing. Configurable and known transmission settings add some advantages of a signal simulator. An overview of the testing procedure in the test bed is depicted in Figure 1-4.

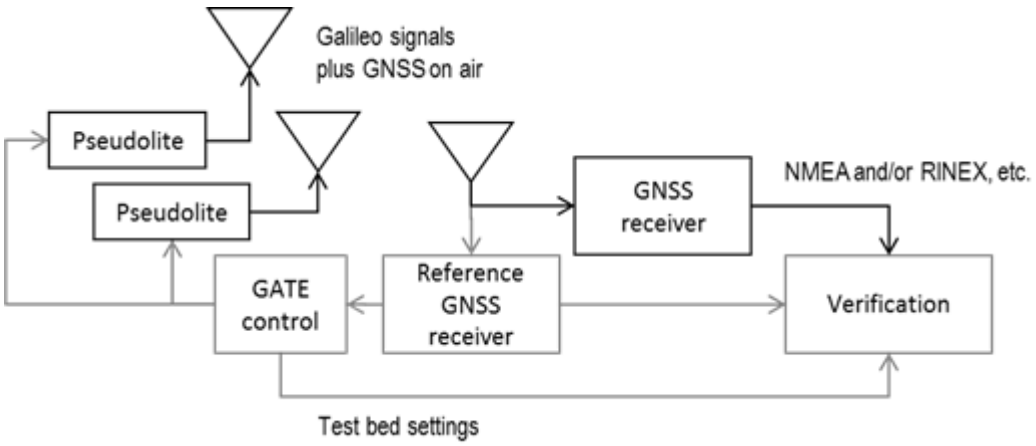

Figure 1-4 GATE test bed in Berchtesgaden, Germany: pseudolites transmitting Galileo signals

Overview of testing infrastructure available in Europe can be found at the European GNSS Simulator and Testing Infrastructure portal [5].

1.3.3 Testing of Receiver Modules

For the specification of the role of a GNSS simulator in receiver testing, an overview of the receiver architecture is given. A generalized architecture of a receiver is depicted in Figure 1-5. This section focuses on the specific parameters of each component of the front end, baseband processor, navigation processor and user interface to be tested and shows, where the simulator input and simulator parameters can be applied.

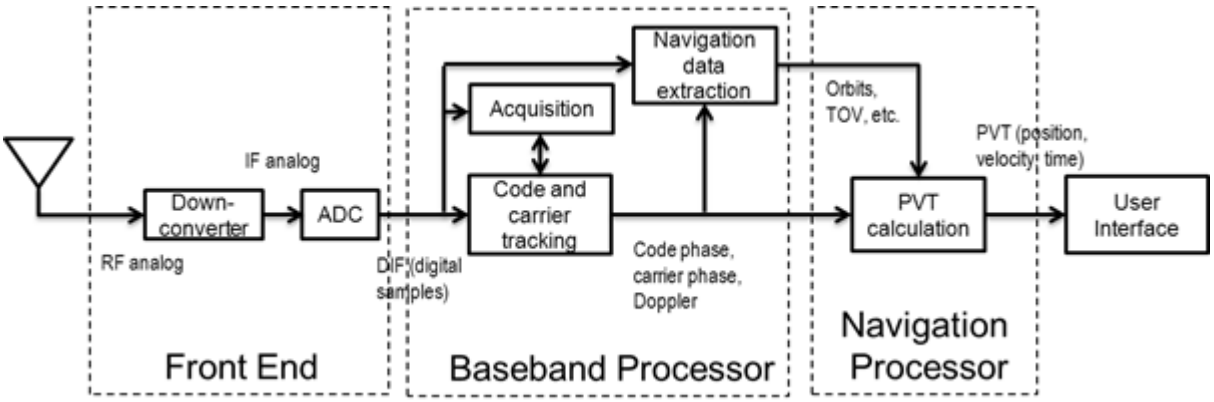

Figure 1-5 GNSS receiver structure

For the grey-box testing of the receiver front end, the signal from field testing or from a simulator can be taken as input. Nevertheless, the knowledge of analog signal parameters of the signal is necessary.

Parameters to be verified are local oscillator's phase noise limits, noise figure of each component, noise figure of the complete front end, second- and third-order intermodulation and characteristics of each filter. Spectrum analyzer and oscilloscope can be used to evaluate the analog inter-stage signals.

For testing of the baseband processor, either a DIF signal simulator or a RF signal simulator can be used to generate test input. In case of the acquisition module, the functionality must be inspected, i.e. the following questions need to be investigated: Are all satellite signal channels acquired? Does the acquisition of pseudolite signals work correctly? Are the signal code phase and Doppler frequency correctly measured and do they work for all possible combinations of both?

Besides its functionality, also the acquisition sensitivity and power profile must be measured. These features can be tested only with a signal generator with configurable power settings. Furthermore, the dynamics behavior, i.e. elevation, speed, acceleration and jitter limits must be evaluated with velocity and acceleration profiles. For these tests, a configurable signal generator is necessary. At this point, nevertheless, a caution must be exercised. Signal simulators have limited configurability in terms of acceleration, jerk and rotation. Limits of the expectation on the simulator must be clarified to choose the simulator with fitting parameter ranges. This topic including a proposal of standardized description of these parameters is discussed in [6].

The tracking module of the receiver baseband processor is tested similar to the acquisition module. Functionality, sensitivity and reactions to the dynamics behavior must be verified. Specifically, the precision and stability of code and carrier tracking must be tested. The input signal from a GNSS simulator is very helpful for profiles and special cases. Nevertheless, the authenticity of the generated signal is limited, because the slight distortions below the limits given in GNSS system specification

differ. The real distortions by individual satellites are not equal to distortions caused by the parameters of simulator components. For these fine-tuning, real tests with real signals in case of already transmitted GNSS signal or test bed signals in case of not yet implemented signals must be included in the test plan.

For verification of the navigation data extraction module, all aspects of data message decoding and interpretation must be thoroughly tested. For the test of all levels of data decoding, both a GNSS simulator and field tests can deliver input: demodulated symbols, decoded message bits, check sums and error correction coding, frame synchronization and data interpretation. Nevertheless for the verification of rare events, specifically concerning time (end of week, week rollover, new leap second, new ephemerides or almanac), niche situations (not-healthy satellite, special bad data cases) and new message schemes a RF or DIF signal simulator is necessary.

The key part of the navigation processor module is the calculation of receiver position, velocity and time (PVT). This module is in comparison to a telecommunication receiver the really unique part of a GNSS receiver.

In this module, the processing of all the input data for time measurement and all the respective niche cases – end-of-week, end-of-year, week number rollover, and leap-second event – need to be verified. Precision of the time measurement plays a key role in the final receiver PVT solution and the resulting output and it must be proved with utmost care. The same must be carried out for position and velocity measurements, where again precision in static and dynamic scenarios and performance in whole range of niche situations: maximum speed, acceleration, conversions of special coordinates (crossing equator, 0° and 180° meridian) must be verified.

Input for this module can come from field measurements or, in case of niche events, from a simulator. Nevertheless, also special end-to-end simulators exist, which simulate tracking loops output together with navigation data and can be used to test PVT algorithms. They are mostly used in the development phase of a receiver.

1.3.4 Testing Specific to Various Receiver Design

For different receiver designs, specific tests must be carried out. In case of software receivers, the real-time capability must be verified and assessed. In case of non-real-time operation systems, very long (> 24 h) tests must be carried out. For multi-threading implementation of a receiver, the unrepeatable computation order must be taken into account. Record and playback devices are limited in length of a test by the capacity of the storage medium. Simulator or field measurements are suitable for this task, nevertheless the repeatability offered by a simulator is convenient for bottleneck pin point.

Hardware and programmable hardware (field-programmable gate array - FPGA) based receivers need special tests of FPGA designs and breadboard prototypes. For these tasks, simulators as well as record and playback devices can be deployed.

In case of multi-frequency receivers with multiple front ends, the inter-frequency biases must be tested and compensated accordingly. Multi-frequency simulators deliver the necessary input; nevertheless, these simulators are quite expensive, especially because the adjustment of inter-frequency biases in the simulator is a very demanding task.

Multi-GNSS receivers, even though only single frequency based, need additional tests that take into account multiple time references and coordinate frames.

Spaceborne and partly also airborne receivers are dependent on simulators for qualification tests as the respective speed, elevation, acceleration, etc. parameters are extremely costly to test in field. Specific tests for spaceborne qualification like radiation tolerance also apply.

1.3.5 Tests for Benchmarking

Testing is also needed for benchmarking of the GNSS receiver performance. Typical tests are time-to-first fix (TTFF) measurements for various start-up modes, acquisition sensitivity measurements (minimum power level at which the receiver is able to correctly identify a satellite signal) and tracking sensitivity measurements for given time limit (minimum power level at which the receiver is able to correctly acquire, track and reacquire a satellite signal). Additionally, tests of assistance information and receiver clock specification apply. The key parameter to be measured is the positioning accuracy. There are several various definitions of this parameter, for example the 1 PPS Test (point per second). For receivers with 1 pps output, it compares position output with 1 pps output from a timing receiver or from the raw data of the simulator and it evaluates the variance of the position accuracy. Further benchmarking tests are time precision test and dynamic positioning test giving profiles of received velocity and acceleration. For these tasks, a GNSS simulator is a necessary tool.

During the benchmarking of a GNSS receiver, also the reaction to the various signal errors is evaluated. These are ionospheric errors and scintillation, tropospheric error, clock error, intentional and unintentional interference, multipath, satellite failure (RAIM testing). For backward compatibility, also selective availability switched off in 2000 can be taken into account. Input for such tests is difficult to obtain in field tests, but signal simulators offer models and settings for generation of such errors.

Benchmarking is done also in general, non-GNSS specific categories, like power consumption, electromagnetic compatibility and in final product specific categories.

1.3.6 Testing of GNSS Receivers for Certification

For many areas of operation of a GNSS receiver, one or more kinds of certification apply. Specific testing procedures must be passed by the receiver to obtain the certification.

A certification can be requested as a prerequisite for introduction at a local market. It can be also bound to a specific market segment. Typical field, where certification is needed, is the area of safety-of-life applications. A specific certification is necessary for airborne receivers and for public-fleet management. In the United States, it is also needed for deployment in mobile phones to follow the E-911 regulation.

The US emergency call service E-911 is regulated by a special mandate of the Federal Communications Commission. The Phase II of this mandate requires caller location to be established for 67% of mobile calls within 50 meters and for 95% of calls within 100 meters for telephone handset-based solutions [7]. In contrast to it, the EU directive E-112 from 2003 requires the mobile

phone service providers to provide whatever information they have about the location from which the emergency call was made, even though no requirements on availability and accuracy are given.

The certification for GNSS receivers in the application area of cellular phones is summarized in [8]. The relevant group of standards are the 3GPP (Third Generation Partnership Program) standards. The 3GPP GPS Performance Standards (TS 34.171) concern the GPS receivers. They define minimum performance requirements in given set of tests cases with given initial conditions concerning among others number of satellites-in-view and signal strength. Besides the GNSS specific standards, also more general 3GPP2 CDMA Performance Standards (TIA 916) apply to GNSS receivers.

More recently, 3GPP2 standards have been developed. Standard TS 37.571-1 covers multi-constellation GNSS systems. It defines minimum performance requirements for GNSS receivers supporting multi-constellation. It includes the same set of test as in case of TS 34.171 standard. For example, for the sensitivity fine time assistance test for multi-constellation receiver the satellite allocation per constellation is stated to be six satellites for single constellation, three and three for two constellations respectively and two, two and two for three constellations. In addition, new standards are being rolled out for over-the-air (OTA) testing, namely the CTIA TIS requirement.

Standard certification equipment include GNSS signal simulator and multi-GNSS reference receiver. The test equipment must be certified itself before it can be employed for the certification.

2 GNSS Signal Simulators

GNSS signal simulators generate input signal for receiver testing at all stages of receiver lifecycle. This chapter gives categorization of the simulators and overview of the features of the contemporary commercial devices.

2.1 Categorization of GNSS Signal Generating Devices

The family of simulators of GNSS signal comprises various devices with confusing naming conventions. The term "GNSS simulator" stands typically for a tool that simulates operation of a GNSS receiver. It produces as an output receiver coordinates or pseudorange data called also raw data. Sometimes this term is also used for a tool simulating the receiver operation together with operation of the GNSS system. In this case, the tool should be called unambiguously an end-to-end GNSS simulator. For devices generating analog or digital GNSS signal as an output, the term GNSS signal simulator should be used.

Ambiguity surrounds also the usage of terms GNSS signal generator and GNSS signal simulator. In the literature consensus was reached [2] that the term GNSS signal generator labels a device generating one channel (i.e. signal from one satellite) of RF GNSS signal. In contrast to it, a GNSS signal simulator, called equivalently also GNSS constellation simulator, generates RF GNSS signal with multiple channels and simulates the geometry of the satellites in motion. It applies additional signal errors including satellite and propagation induced errors with respect to the given geometry.

A device generating digital intermediate frequency GNSS signal is also sometimes called a GNSS simulator. The precise term is nevertheless DIF GNSS generator or GNSS IF (software) signal simulator.

2.1.1 DIF GNSS Generator

A digitized intermediate frequency generator is the simulator type with lowest internal technical complexity. It is a software application that generates the DIF signal as it would be output out of the specific receiver front end. It is therefore a receiver-specific or at least receiver front-end specific tool. A complete constellation of satellite signals is simulated in a defined operating environment with all special cases and signal errors. The range of simulated features resembles a GNSS RF signal simulator. Antenna pattern and front end filters are added. The principle of application of a DIF generator is given in Figure 2-1.

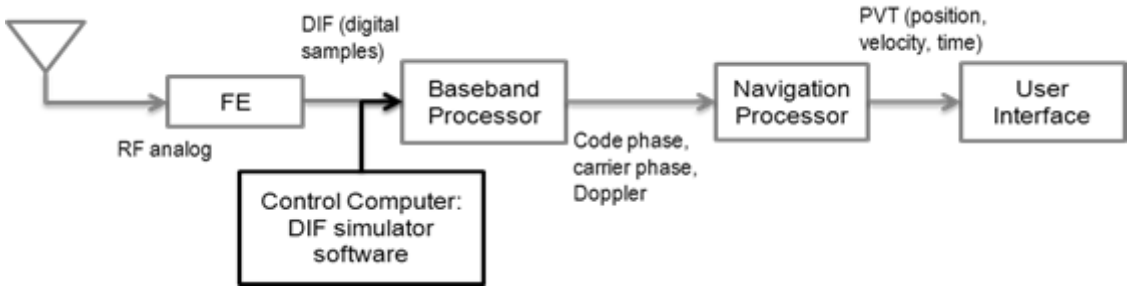

Figure 2-1 Application of DIF generator for receiver testing

2.1.2 GNSS Signal Generator

A GNSS signal generator offers full control over parameters of one generated RF GNSS signal channel. Power level can be typically adjusted over the whole range from ca.-90 dBW to ca. -185 dBW. Doppler profile can be set and full navigation message configured. Products on the market offer choice of one channel from range of one to four signal services. Devices from Spirent, Meguro and Spectracom are depicted in Figure 2-2.

GNSS signal generators are used for testing of front end and baseband processor, especially for optimization of acquisition and tracking loop. They are handy for measuring power and Doppler profiles. They are mostly utilized in development and production stages of receiver lifecycle. They are similar to and usable as pseudolites. The functionality of a GNSS signal generator can be a subset of GNSS signal simulator functionality, but in its application field the advantage is the low cost below ca.10,000 EUR.

Figure 2-2 Single-channel GNSS signal generators: Spirent GSS6300 (L1/E1/G1/B1) [9] - left, Meguro MSG-2051A (L1 C/A) [10] - middle, Spectracom GSG-51 (L1 C/A) [11] - right

2.1.3 GNSS Signal Simulator

The GNSS signal simulators on the market are mostly sorted according to available signal services into three categories as single-service, single-frequency multi-service and multi-frequency multi-service signal simulators.

A single-service simulator generates RF signal of the full constellation of satellite channels of one GNSS signal service for a receiver in motion. It offers mostly 8 to 12 channels for the maximum number of satellites in view. Generation errors (clock error, evil waveforms, and satellite failure), propagation errors (ionospheric error, tropospheric error, and multipath error), interferences (unintentional and jamming) as well as receiver errors prior to simulator input to the receiver (antenna pattern and low-noise amplifier gain) can be added. Simulators use two-levels of modeling. True models describe the simulated scenario. They are applied on the signal being generated. Broadcast models describe the scenario as it would be modelled and broadcast by the respective GNSS or SBAS service provider. They are used to generate the data transmitted in the respective message. The relationship between these two levels of modelling is nevertheless to be set by the user. Price of these devices ranges between 50,000 and 100,000 EUR.

A single-frequency multi-service simulator adds one or more signal services located in the same frequency band, typically in the L1 band. User assigns the available channels to the satellites of the signal service of his choice and adjusts the intersystem time delay.

The multi-frequency multi-service simulators, also called professional simulators in the literature, strive to offer the full scale of GNSS current and planned signal services. Such simulators comprise several individual single-service simulator modules on multiple frequency bands with the intersystem time settings. These devices are expensive as it is hard to reach full inter-frequency synchronization and generate so many channels at once. Price of these devices starts at around 200,000 EUR and rises open-end along with the number of services and frequencies included. Exemplifying products on the market in 2013 are depicted in Figure 2-3.

Figure 2-3 Spirent GSS8000 [9], 3 carriers – left; IFEN NavX-NCS Professional [12], 9 carriers – middle; Spectracom GSG-64 [11], 4 carriers – right

As a special subcategory of the GNSS signal simulators, the military simulators should be mentioned. They are typically available as an extension to a single- or multi-frequency simulator. In case of GPS, all three GPS L1, L2 and L5 frequency bands are available and the military (classified) Y – codes and M – codes can be generated. A military simulator also delivers data for testing the receiver SAASM module - red keys, black keys and over-the-air rekeying (OTAR). Additional functionality are also up to 16 RF outputs for CRPA antenna testing, input to INS testing on IF (mainly Honeywell H764G and Northrop Grumman LN-100G) and the tools for integrated GPS/INS tests (delivery of attitude and velocity increments). Equipment for receiver clock *g*-sensitivity testing is on board. Costs for such a military simulator exceed 1,000,000 USD. Examples of such devices are given by the military extensions to Spirent and the CAST-2000 simulator, depicted in Figure 2-3 left and Figure 2-4 left.

Figure 2-4 GPS Signal Simulator CAST-5000 [13] with 2 carriers – left, Cobham GPSG-1000 [14] with 1 carrier – right

2.2 Features of GNSS Signal Generating Devices

The usage of simulators implies also several issues. The advantages and disadvantages are closely described in [2]. These devices can be easily used as a jammer or a spoofer. Except scientific experiments, no observation of such a misuse has been published to date. Another topic is the copyright infringement caused by further utilization of simulator products, either selling or distributing

of files recorded from a simulator. This data can be easily generated and used with help of a record and playback device.

The quality of the generated signal is the main figure of merit of a simulator. The signal should be as similar as possible to the real signal with all possible simulator settings. The author of [15] names key parameters for good performance to be:

- Quality of code and carrier phase generation. This means that pseudorange root mean square (RMS) error should be below 0.3 m including the inter-channel bias. The uncertainty of pseudorange rate (RMS error) should not exceed 0.03 m/s. Inter-channel code and carrier alignment must be one order of magnitude better than the expected performance of the receiver under test.
- Precision of the power level. Uncertainty of the overall simulated power level should not exceed 1.0 dB or at most 2.0 dB.
- Quality of carrier signal in terms of phase noise and frequency stability. Carrier frequency should be centered with an accuracy of not worse than a few Hz even after a few years of operation. Master clock stability over one day should stay within $\pm 5 \times 10^{-10}$.
- Physical dimensions. For lab test, limitations on dimensions are flexible. The device mostly needs only to fit in a standard lab rack mount. For special on-site tests, a need for a handheld device, like GPSG-1000 from Cobham depicted in Figure 2-4, may exist.

2.3 Overview of RF Simulators Market

GNSS signal simulators are niche electronic and software-driven products on the special electronics market. The development in the field of GNSS system towards multiple systems and signals paired with high precision pose very high and steadily increasing expectations on the devices. In this chapter, a market overview of the RF GNSS signal simulators as a category defined in section 2.1.3 is given together with insight in their capabilities. Table 2-1 gives the overview of implemented signals and systems. Table 2-2 lists the maximum number of simultaneously generated channels and frequency bands. The ability to simulate data for differential corrections and advanced signal perturbations is also listed according to the datasheets of the products. Both tables show the status in 2014.

The Spirent Communications has been developing and manufacturing GPS signal simulators since 1985 [9]. The company is the market leader covering all segments of the GNSS simulator market. Only the flagship of their products, the model GSS9000 introduced in 2014, is listed in the overview. In the full configuration with all additional modules, it covers practically all current and planned GNSS and SBAS signals and offers modelling of all effects influencing the signals. Even the military signals, SAASM modules and IMU testing are included.

The key competitor in the segment of high-end scientific simulators is IFEN GmbH [12] followed by Spectracom [11]. Until 2014, the NavX-NCS by IFEN outperformed Spirent's top product in the number of simultaneously generated channels and frequency bands. However, this Europe based company does not participate in the segment of military simulators, which in US generates a great deal of revenues. In this segment, the CAST Navigation [13] is involved featuring strong GPS/INS focus. This company entered the market of non-military simulators first in 2007 and it is now present even on the opposite side of the product spectrum with their simple hand-held devices.

The hand-held devices, marked as HH in Table 2-1, are a young segment of the market. Conventional simulators use the standard rack form factor with an additional laptop. In contrast to it, the hand-held devices joined all functionality of one signal service simulation, display and steering in a compact, several pounds heavy box with rugged coating. Besides the already mentioned CAST, also the company Cobham [14], formerly Aeroflex Test Solutions, markets such a hand-held simulator.

Manufacturer	Model	Type	GPS	Galileo	GLONASS	QZSS	Beidou	IRNSS	SBAS
Spirent	GSS9000	SS	L1 C/A, L1C, L2C, L5, L2C, L1P(Y), L1M, L2M	E1, E5ab, E6	G1, G2	L1	B1, B2, B3	L5S	WAAS L1, L5, EGNOS, MSAS, GAGAN
IFEN	NavX-NCS	SS	L1, L2/L2C, L5	E1, E5ab, E6	G1, G2	L1	B1, B2, B3	L5S	WAAS L1, L5, EGNOS, MSAS, GAGAN
Spectracom	GSG-64	SS	L1C/A, L1P,L2P, L2C, L5	E1, E5ab	G1, G2	L1	B1, B2	L5S	WAAS, EGNOS, MSAS, GAGAN
CAST Navigation	CAST-SGX	SS HH	L1 C/A, L1P	No	No	No	No	No	WAAS
CAST Navigation	CAST-2000	SS	L1 C/A, L1P, L1M, L2P, L2M	No	No	No	No	No	WAAS
Rohde & Schwarz	SMBV100A	SS	L1 C/A, L1P	E1	G1, G2	L1	B1	No	No
National Instruments	NI GPS Simulator	SS	L1 C/A	No	No	No	No	No	No
Cobham	GPSG-1000	SS HH	L1 C/A, L1C, L2C, L5	E1, E5ab	No	No	No	No	WAAS, EGNOS
Racelogic	LabSat3	DIF + RP	L1 C/A	E1	G1	L1	B1	No	No
NavSys	AGHS	DIF + RP	L1C/A, L1P, L1M, L2P, L2M	No	G1	No	No	No	No

Table 2-1 GNSS signal simulator market products – type, signals and services. SS – real-time signal simulator, DIF + RP – DIF simulator and replay device, HH – hand-held device

GNSS simulators build as software modules on the top of a general purpose vector signal generators are another new segment of the market. They do not offer the high-end features of a specialized GNSS simulator, but they can add full scale of radio and telecommunication signals for development of combined chips and applications. Rohde & Schwarz offers broad GNSS functionality for their vector signal generator [16]. National Instruments also sells a simple GPS L1 C/A module for their modular signal generation and processing system [17].

Manufacturer	# Channels (signal + multipath + interference)	# Simultaneous carriers	Differential corrections modelling	Multipath modeling	Lever arm effect	Antenna reception gain pattern	Interference emulation	Atmospheric models	Dynamic model
Spirent Communications	160 + 640 + 4	10	Yes	Yes	Yes	Yes	Yes	Yes	Yes
IFEN	108	9	?	Yes	No	Yes	No	Yes	Yes
Spectracom	64	4	No	Yes	No	No	Yes	Yes	Yes
CAST Navigation	16	1	No	No	No	Yes	No	Yes	Yes
	60 + 8	2	Yes	Yes	No	Yes	Yes	Yes	Yes
Rohde & Schwarz	24	2	No	Yes	No	Yes	No	Yes	Yes
National Instruments	12	1	No	No	No	No	No	No	Yes
Cobham	12	1	No	No	No	No	No	Yes	Yes
Racelogic	36	3	No	No	No	No	No	Yes	Yes
Navsys	unlimited	?	No	No	No	No	No	Yes	Yes

Table 2-2 GNSS signal simulator market products – performance and signal error effects

Most of the devices pre-generate some constellation description first, and then generate the digital and analog signal in real time. Devices with such architecture are labeled as SS (Signal Simulator) in Table 2-1. The devices that work as a non-real-time DIF signal generator, which saves the DIF signal to a storage device and replays it on demand in real-time, are labeled as DIF + RP. Racelogic offers such a device with focus on multi-system single frequency receiver testing [18]. NavSys markets a modular system combining Matlab-based DIF generator with playback function [19].

Besides the listed manufacturers and their products, also American GNC Corporation offers a simulator product called AGNC-2000RTGIS GPS/IMU Real-Time Simulator [20]. Little information is published about this device. This and the fact that it was developed in the frame of GSA (US General Service Administration) could indicate that the device was not developed with the focus on continuous commercialisation.

As the simplest simulator in the comparison the product by National Instruments can be seen. It simulates from the all the features listed in the tables only the receiver dynamics. The most complex contemporary device by Spirent offers full scale of transmitted, propagation and receiver errors. Table 2-2 shows a simplified overview of chosen abilities, but the implementation of the features and settings they offer differ tremendously among the devices. Data sheets of the simulator rarely state closer information about the implemented models, so the direct contact with the manufacturers in necessary for detailed comparison of the products.

2.4 Previous Work on GNSS Signal Simulation and GPU-based Signal Processing in GNSS Technology

The GNSS signal simulation is a niche topic in the GNSS field. The manufacturers of the commercial devices give the information about the signal features and quality in the product description, but do not publish information about the architecture and algorithms. Publications of the manufactures focus either on verification of the simulator or testing procedures with usage of their products. Verification of the Galileo signal simulation was published by Spirent [21] and performance evaluation of the multi-frequency multi-service simulator was published by IFEN [22].

Literature about experimental modelling on top of a commercial simulator is a topic where multiple publications with participation of the simulator manufacturers exist. Ionospheric scintillation modelling on top of Spirent simulator was described in [23]. This is possible thanks to the signal description input interface with 1 kHz update rate (model GSS9000) [9].

Overview of the GNSS signal simulators was at latest the topic of GNSS literature in 2013 as series about GNSS simulator products and technology [24].

The publications about the state-of-art of the simulator architecture and algorithms can be found in the field of experimental simulation systems developed at research institutions and universities. These focus mostly on DIF simulators. These tools are needed by the research groups themselves and they have to fit their research projects and to be compatible with their experimental GNSS receivers.

In [25] a CPU-based DIF simulator for GPS and Galileo with multipath simulation was introduced. Another CPU-based DIF simulator of L1 C/A and E1 signal with conversion of 6 Msps signal to analog using a PCIe DAC board was described in [26]. A multiservice DIF simulator on top of NI PXI system is introduced in [27]. Architecture of an experimental RF simulator with FPGA and DSP is roughly described in [28].

The deployment of GPU in GNSS signal processing was already demonstrated in GPU based GNSS receivers. The first work, [29], was published in 2008 and it described a GPU-based tracking. The authors of [30] followed in 2009 with acquisition and tracking, using CUDA to program the GPU. Thereafter, authors of [31] published a deployment of GPU for tracking. A SW receiver with GPU based tracking was introduced also by [32], [33], [34] and [35]. A later version of the GPU-using SW receiver [29] switched later to CUDA technology as well [36].

In parallel to the first publication of single service version of the GPU based GNSS signal simulation architecture [37], which is subject of this work, another work on GPU based GNSS signal simulator was published [38]. As only the presentation is given in English and the paper itself in Chinese, closer comparison of the concepts is not possible to the author of this thesis.

3 GNSS Signal Simulation Theory

3.1 Digital Signal Generation

3.1.1 Digital Signal

Digital representation of an analog signal characterizes the signal voltage levels as a sequence of digital numbers. Analog signal voltage level is a continuous variable defined at any time point. Hence, its complete digital representation would have to include infinite number of points with infinite precision. Such a representation would not be appropriate to be processed by any real world device, since it would require an infinite amount of memory and infinite amount of bits per number. Sampling solves such a problem by taking samples at the fixed time intervals, as shown in Figure 3-1, where T represents the interval between two successive samples in seconds and $x(t)$ represents the voltage level at time point t.

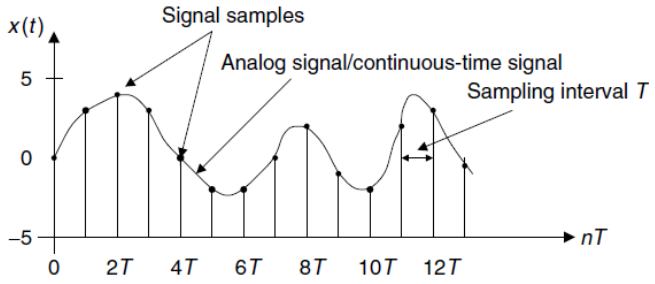

Figure 3-1 Analog (continuous) signal and digital samples of voltage at the sampling time instants [39]

For the sampling interval T, we can define the sampling rate in samples per second or equivalently in Hertz by

$$f_S = \frac{1}{T}. \tag{3-1}$$

A signal period of t seconds is thus represented by a series of n amplitude values, where the length of the series evaluates to

$$n = tf_S. \tag{3-2}$$

Additionally, we have to represent the continuous analog signal with a sample rate, which is high enough so that the analog signal can be constructed from the digital series by a digital-to-analog converter correctly. Thus we need to set the minimum sample rate so that no unwanted signals occur in the desired frequency band. The sampling theorem guarantees perfect construction of the analog signal from its digital representation, as long as the sample rate is more than twice as large as the highest-frequency component f_{max} of the analog signal to be represented. The minimum sample rate must fulfill the criterion

$$f_S > 2f_{max}. \tag{3-3}$$

This fundamental rule is known as the Nyquist-Shannon sampling theorem. The half of the sampling frequency $f_S/2$ is called the Nyquist frequency, Nyquist limit or folding frequency.

If the Nyquist-Shannon criterion is not fulfilled, unwanted signals called aliases will occur in the desired frequency band. In practice, the high frequencies that would cause aliasing need to be removed before sampling using a lowpass filter. When converting the digital signal to analog, the image frequencies occur and a reconstruction lowpass filter smoothing the recovered sample-and-hold voltage levels to an analog signal have to be applied after digital-to-analog conversion [39].

The relation of the spectrum of the sampled signal $X_S(f)$ to the original baseband spectrum $X(f)$ of the analog signal is given by

$$X_S(f) = \frac{1}{T}\sum_{n=-\infty}^{\infty} X(f - nf_s). \tag{3-4}$$

Where $X(f \pm nf_S)$ is the spectrum of the baseband signal in case of $n = 0$ and spectrum of the signal images in case of $n \neq 0$. An example of the original spectrum of the signal is depicted in Figure 3-2 (A). The spectrum of the respective digital signal sampled with $f_S > 2f_{max}$ is depicted in Figure 3-2 (B). Only the part of the spectra defined by equation (3-4) for n = -1, 0, 1 is included in the figure. When using $f_S = 2f_{max}$, the spectrum and its images are directly adjacent to each other, as depicted in Figure 3-2 (C). In this case, a realizable lowpass filter cannot recover the signal without influencing the frequencies close to f_{max} and f_{min}. When using $f_S < 2f_{max}$, the spectra overlap and the interval $(f_{max} - (2f_{max} - f_S), f_{max})$ is interfered by the image signal, as depicted in Figure 3-2 (D).

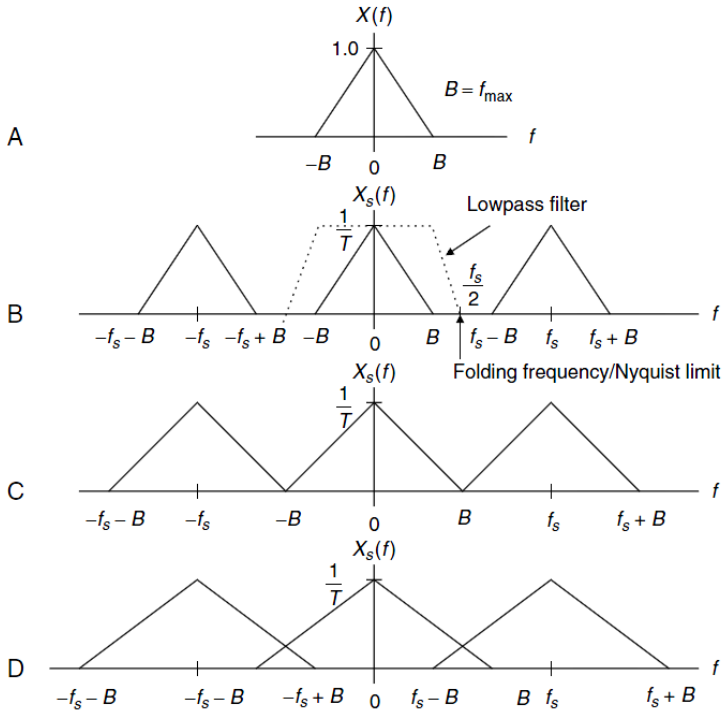

Figure 3-2 Spectrum of original signal (A) and sampled signal with more than sufficient sampling rate (B), sufficient sampling rate (C) and insufficient sampling rate (D)

3.1.2 Digital to Analog Conversion

The steps of construction of the analog signal from the digital signal is illustrated in Figure 3-3. The digital-to-analog converter converts the digital signal samples $y_n(t)$ to the analog sampled signal $y_S(t)$. The hold circuit produces the sample-and-hold voltage $y_H(t)$. The hold circuit depicted in Figure 3-3 (C) is realized according to the zero-order hold (ZOH) model. The ZOH model converts analog sampled signal $y_S(t)$ to continuous analog signal by holding each sample value for one sample interval. The hold function in ZOH model is a zero-order polynomial function, in other words the constant function. A far less common alternative to the ZOH is the first-order hold model, that connects the impulses of $y_S(t)$ with a linear function.

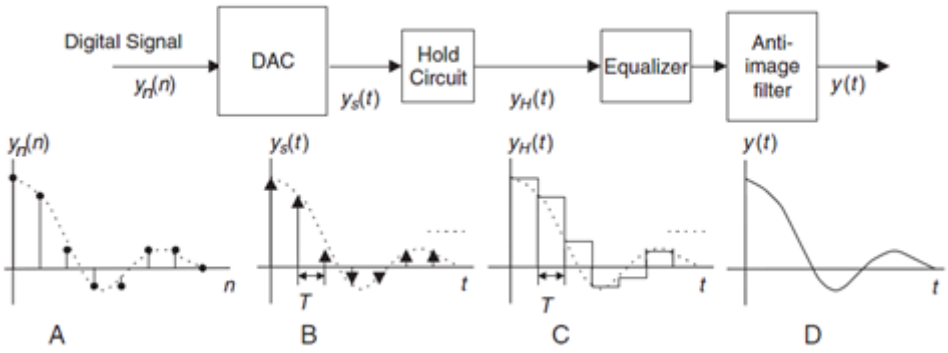

Figure 3-3 Steps of digital-to-analog conversion of the signal. A – digital signal, B - analog sampled signal, C –analog sample-and-hold voltage signal, D – generated analog signal [39].

The zero-order hold circuit converts the sequence of modulated Dirac impulses $y_s(t)$ to the signal

$$y_H(t) = \sum_{n=-\infty}^{\infty} y(n) \cdot rect\left(\frac{t - nT}{T} - \frac{1}{2}\right), \tag{3-5}$$

Resulting in an effective impulse response

$$h_h(t) = \frac{1}{T} \cdot rect\left(\frac{1}{T} - \frac{1}{2}\right). \tag{3-6}$$

The transfer function of the ZOH hold circuit is given by

$$H_H(s) = \frac{1 - e^{-sT}}{sT}, \tag{3-7}$$

where $s = j2\pi f$.

The frequency response of the DAC together with the hold circuit can be obtained as

$$H_H(f) = e^{-j\pi fT} \frac{\sin(\pi fT)}{\pi fT}. \tag{3-8}$$

In terms of magnitude of frequency response, we can calculate

$$|H_H(f)| = \left|\frac{\sin(\pi fT)}{\pi fT}.\right| \tag{3-9}$$

The magnitude frequency response shapes the sampled signal spectrum in the desired frequency band according to the *sin(x)/x* function. The positive effect is that the spectral images are attenuated due to the lowpass effect of the *sin(x)/x* function. An example of such a spectrum is depicted in Figure 3-4. This effect must be considered when designing a lowpass filter to remove the image frequencies from the generated analog signal. Detailed description of the features can be found in [40].

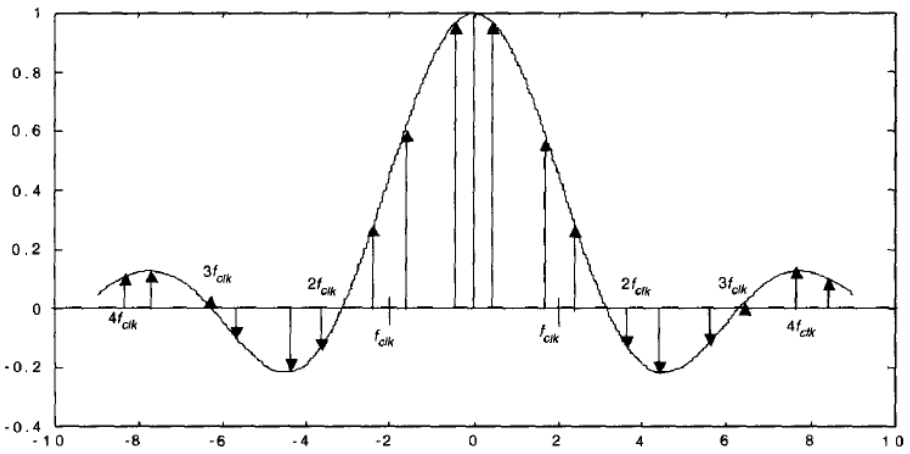

Figure 3-4 Example of spectrum of sampled signal [42]

The sample-and-hold effect can be compensated using an equalizer with an opposite magnitude response to the response of the hold circuit. The sampling rate can be increased and combined with interpolation. Baseband spectrum and its images are separated farther apart and lower-order filter can be used. This strategy is used in designs of PCIe DAC boards.

The digital signal $y_n(n)$ enters the DAC in form of a sequence of binary representation of each sample value. Each this binary represented sample consists of p bits and the number of voltage steps that can be represented by this binary number is 2^p. The finite resolution of the sample introduces the quantization error, described by the equation

$$e_q(n) = y_n(n) - y(n), \tag{3-10}$$

where $y_n(n)$ represents the digital signal samples with binary precision p bits and $y(n)$ represents the signal level expressed with unlimited precision.

The quantization error is bound by half of the voltage step size. With respect to maximum amplitude A_{max}, this relationship can be defined as

$$\left|e_q(n)\right| \leq \frac{A_{max}}{2^p 2}, \tag{3-11}$$

where the minimum amplitude step $A_{min} = A_{max} / 2^p$. For a signal of sufficient length, where the step size is much smaller than the dynamic range of the signal samples the quantization error appears in uniform distribution. With respect to the theory of probability, the power of quantization noise is related to the quantization step in a way described by equation

$$E(e_q^2(n)) = \frac{A_{min}^2}{12}, \tag{3-12}$$

where *E()* is the expectation function and $A_{min} = 1/2^p$. The closer explanation of the formula can be found in [41]. The quantization error appears in the form of the quantization noise. Its level is typically described with the means of the signal-to-noise ratio (SNR). The signal to quantization noise ratio can be evaluated as

$$SNR = \frac{E({y_n}^2)}{E(e_q^2)}. \tag{3-13}$$

Practical calculation of the SNR can be done using the root mean squared (RMS) value for expectation function evaluation as

$$SNR = \frac{\sum_{n=0}^{N-1} y_n^2(n)}{\sum_{n=0}^{N-1} e_q^2(n)}. \tag{3-14}$$

Another source of spurious signals added by DAC is the timing jitter. Depending on the quality of the clock input, this spurious energy is of non-linear nature and it is much lower in amplitude than the replicated images.

3.1.3 Digital Signal Synthesis

Digital to analog conversion is traditionally considered to be a part of a signal processing system, where analog signal is converted to digital, processed, and then converted to analog again. With the progress of performance of the digital processing hardware, application of digital signal in the field of signal generation became possible. The so called direct digital signal synthesis (DDS) have already replaced the traditional analog techniques in many applications with lower sample rates and bandwidth. The advantage of the analog techniques is the lower power consumption and less spurious frequencies. The older and current GPS satellites generate the signal still in analog.

The common analog signal generation techniques include direct analog synthesis (DAS) and phase-lock loop (PLL) method. DDS overcomes most of the problems associated with both DAS and PLL techniques. It is superior in terms of precision, ease of implementation and flexibility, which leads to its widespread use in digital communication systems [42]. In analog systems, the frequency is controlled with analog components, resulting in poor stability due to drift in the components, poor frequency resolution due to limitations in analog dials, and difficulty with digitally controlled tuning. Furthermore analog signal generators stray in frequency over time. Changes in temperature, humidity and other variables can affect the output of the analog oscillator.

A direct digital synthesis system consists of a frequency reference clock signal, numerically controlled oscillator (NCO) and digital-to-analog converter. The reference clock signal is generated by a single crystal oscillator. The theoretical maximum output frequency of a DDS system is given by Nyquist-Shannon theorem (3-3) to be $f_{clk}/2$. In practice, maximum frequency in the range between $f_{clk}/2.5$ and $f_{clk}/4.5$ is used due to limitations at the analog reconstruction filter [42]. When also the image

frequencies of the sampled signal spectrum at multiples of the sample rate are used, a signal with higher frequency can be generated. A general schema of an NCO is depicted in Figure 3-5.

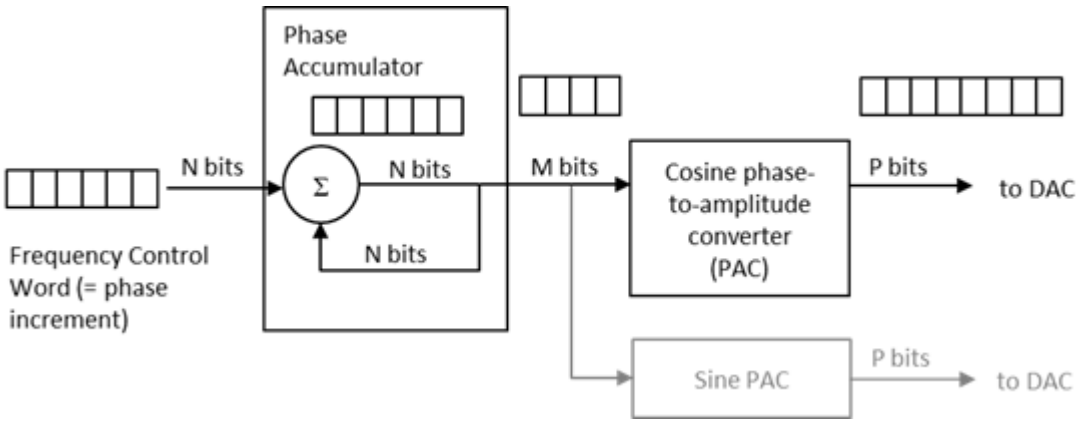

Figure 3-5 NCO schema

An NCO produces at its output a discrete-time quantized version of the desired output waveform. The period of the waveform is controlled by the frequency control word Δ_f, which holds the phase increment. A phase accumulator holds the actual phase and adds the value of the frequency control word at each clock epoch. The accumulated phase is then used as an input in the phase-to-amplitude converter (PAC). The value in the accumulator ACC at clock epoch n is given by

$$ACC(n) = \left(ACC(n-1) + \Delta_f\right) mod\ 2^N. \tag{3-15}$$

The modulo 2^N arithmetic of the N-bit accumulator simulates the periodicity of the waveform being generated. Considering the Δ_f and ACC to be integer numbers within interval [1, 2^N-1], the average frequency of accumulator overflow is given by

$$f_{overflow} = f_{clk} \frac{\Delta_f}{2^N}. \tag{3-16}$$

The frequency of the output signal f_{out} is identical to the average frequency of accumulator overflow. The minimum output frequency is given by $\Delta_f = 1$ and evaluates to $f_{min} = f_{clk} / 2^N$. The phase increment of the NCO in radians can be calculated as

$$\Delta_{f-rad} = \frac{\Delta_f 2\pi}{2^N}. \tag{3-17}$$

The signal is generated with two basic approaches. The first is called pulse output direct digital synthesis. In this approach, the PAC converter is either omitted, or it is realized by the most-significant-bit (MSB) function or the carry (i.e. overflow) bit function. As a result, the sawtooth, rectangular, or pulse waveforms are generated respectively. Other waveforms are then generated from these basic patterns.

The second approach is the usage of a lookup table. The PAC is implemented as a look-up table with 2^M waveform samples as items. The accumulated phase value is used as an index into the waveform lookup table. Resulting amplitude sample is represented by P bits. The waveform period is determined by the frequency control word and the clock frequency.

The phase truncation from N to M bits is needed, because the size of the look-up table can be the major design issue of the DDS system. Therefore, there is a need to limit the memory sources consumed by the NCO. When a sine waveform is generated by the NCO, the phase truncation causes the output signal sequence to be formed as follows

$$S_t(n) = \sin\left(\frac{2\pi}{2^M} integer\ part\left[\frac{\Delta_f n}{2^{N-M}}\right]\right). \quad \textbf{(3-18)}$$

If $M < N$ and $\Delta_f < 2^{N-M}$, the DDS system produces a frequency that is biased. Every so often, the accumulator output does not advance; the lookup table causes the DAC to deliver the same voltage as on the previous clock cycle. The phase is thus held back by $2\pi\imath / 2^M$ radians before continuing to increase.

3.1.3.1 Spurious Components due to Truncation of Phase

The truncation of the phase output word to M bits does not affect the frequency accuracy itself, but produces a time-varying periodic phase error, which is the primary source of spurious products. This phase error has a form of phase modulation with a periodic sawtooth waveform [42]. The location of the spurs caused by the phase truncation occurs at integral multiples of f_{spur} given by

$$f_{spur} = \frac{f_{clk}\Delta_f}{2^{N-M}} - kf_{clk}, k \in \mathbb{Z}. \quad \textbf{(3-19)}$$

The first harmonic spur is typically the strongest one. The more the phase gets truncated and M decreases, the closer spurs move to the ideal signal. The spurs closely adjacent to the target signal are harder to filter out.

Let broadband GNSS signal generation system introduced later in this work in section 5.5 serve as an example. The configuration of such a DSS system generates signal with bandwidth of ca. 427 MHz with f_{clk} = SR = 1.4 Gsps. For simplicity, only E5ab with intermediate frequency (IF) of 51.15 MHz and L1 with IF of 434.775 MHz are considered. The NCO is implemented using N = 32 bits for the accumulator and for the frequency control word. The PAC is realized using a special hardware circuit operating on floating point numbers with precision of 24 bits. The first harmonic spur caused by the phase truncation is then placed at 7.9×10^{15} Hz and at 8.5×10^{14} Hz for L1 and E5 respectively. First the spurs of the order $k > 10^6$ approach the target bandwidth, week enough not to harm the signal of interest.

Let the broadband GNSS signal generation be considered further on. When a PAC with lower resolution was used, the spurs would get closer to the signal. Let $M \in \{8, 7, 6, 5, 4, 3, 2\}$ and the order of the spurs $k \in \{-5, \ldots, 0, \ldots, 5\}$.

The spurs occurring close to the target bandwidth are depicted in Figure 3-6. The L1 and E5 GNSS IF signal bands are visualized as areas with proportional bandwidth; their respective spurs are given as stems. The spurs of the signals are labeled with b denoting the number of bits of PAC resolution M and with k denoting the order of the spur. The figure shows that first with M = 6, the spurs would occur near to and with M = 5 within the signal band. In this case they could not be filtered out effectively.

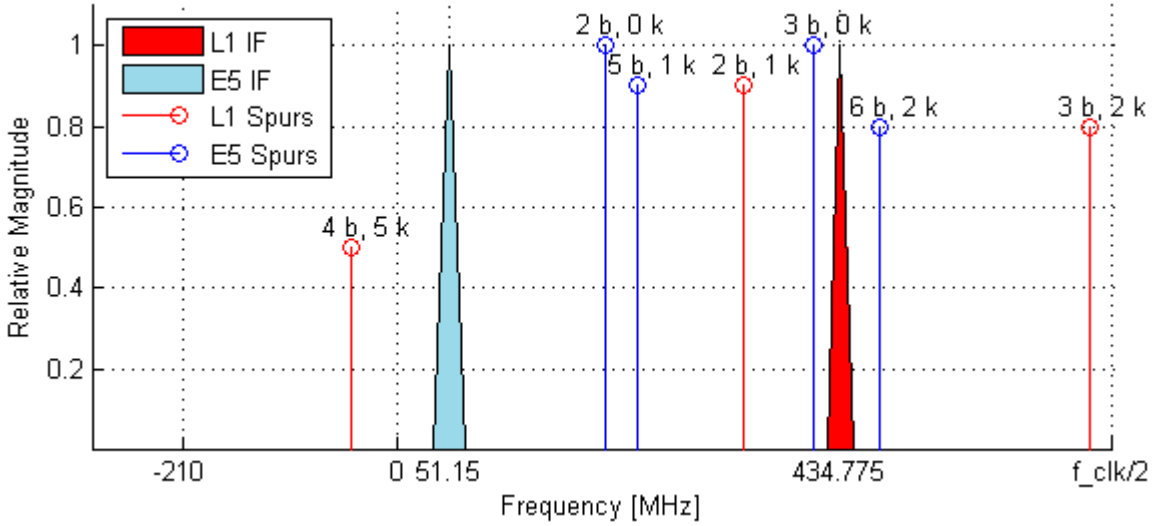

Figure 3-6 Example of spurious products caused by NCO phase truncation (f_{clk} = 1.4 Gsps, b - bits of PAC resolution, k - order of the spur)

3.1.3.2 Spurious Components due to Truncation of Amplitude

The next source of spurious product is the finite word length effects of the PAC output (amplitude) word [43]. The word length P of the PAC output is typically equivalent to the number of quantization levels for the DAC. The quantization to DAC format can also be done after mixing of multiple channels before input into DAC. This truncation introduces distortion and for periodic signals, this distortion consists of harmonic spurious frequency components [42]. The DAC resolution can be improved by oversampling, but it will lead to the need for a clock several times faster than the minimum clock frequency f_{clk}.

Considering the example of the broadband signal generation defined above and in detail in section 5.5, the amplitude is calculated with special hardware circuit with 21.19 bits precision. The DAC proposed in section 8.1 features resolution of 14 bits and it oversamples the digital signal four times to the target speed of 5.6 Gsps. The distortion caused by amplitude truncation by NCO and DAC is therefore low. More severe amplitude truncation is caused by addition of high number of GNSS signal channels to common broadband signal, which is analyzed in section 6.4.

3.1.3.3 Spurious Components Due to Periodic Jitter

The next source of the spurious components in the generated signal is the periodicity of values in the accumulator given by the size N of the accumulator and the value of the frequency control word Δ_f. If for the given Δ_f an integer value k exists so that $k\Delta_f = 2^N$, then each period of the waveform stored in the look-up table will be generated as the same sequence of sample values.

As an example, let $\Delta_f = 2$ and $N = 3$. Then the phase values in accumulator will be (0, 2, 4, 6), (0, 2, 4, 6) and so forth. After four clock cycles, the accumulator will reach the zero value again. Now let $\Delta_f = 3$. Then the phase values in the accumulator will be (0, 3, 6), (1, 4, 7), (2, 5), (0, 3, 6) and so forth. The zero value in accumulator will be reached after three periods of the waveform in the lookup table. This will create a spurious signal with frequency f_{spur}, which evaluates to the third of the target output frequency $f_{out} = 3/8 f_{clk}$.

The general location of the spurious components caused by the periodic jitter is given by

$$f_{spur} = k\frac{\gcd(\Delta_f, 2^N)}{2^N} f_{clk}, k \in \mathbb{Z}, \tag{3-20}$$

where gcd() denotes the greatest common divisor of two integer numbers.

Let the broadband GNSS signal generation defined in section 5.5 serve again as an example. The gcd($\Delta_{f\,L1}$, 2^N) = 8, and the gcd($\Delta_{f\,L1}$, 2^N) = 1. The frequency of the spurs is very low and the first dozens of the spurs are placed very close above 0 Hz. They can be easily removed by a highpass filter. The GNSS signals generated by a simulator nevertheless vary in frequency due to the Doppler Effect. The Doppler frequency shift ranges from ca. -10 kHz to 10 kHz for low dynamic scenarios. The frequency control word of the NCO contains a different value for each Doppler setting. Each frequency control word value causes different spurs due to periodic jitter.

In this NCO, only the Doppler frequency value f_D = -8082 Hz for L1 IF features so high gcd($\Delta_{f\,L1+\,fD}$, 2^N) = 8,388,608 that any of the first ten spurs is located above 1 MHz and therefore close to the lowest defined IF GNSS frequencies. The first ten spurs of this Doppler frequency are depicted in Figure 3-7. In case of the E5ab signal generation, none of the Doppler frequencies within the 10 kHz range cause any of the first ten spurs to occur close to the defined IF GNSS band.

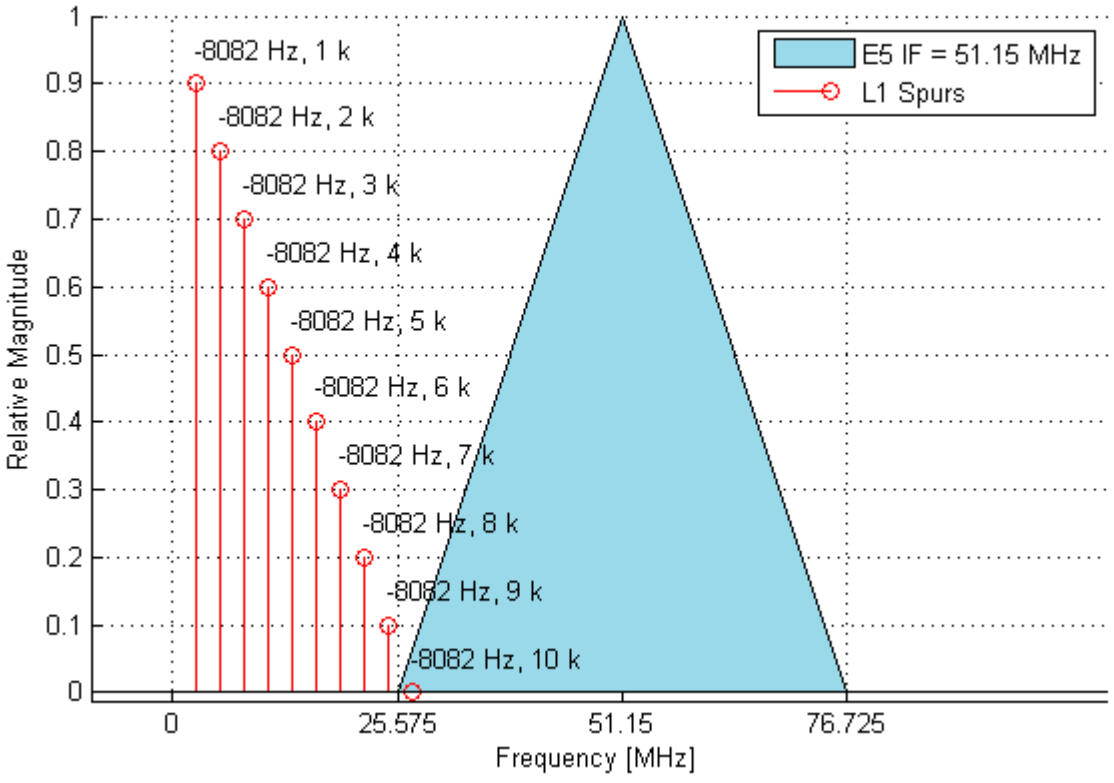

Figure 3-7 Example of spurious products caused by periodic jitter of NCO (f_{clk} = 1.4 Gsps, L1 spurs: f_D = -8082 Hz , k - order of the spur)

3.1.3.4 Performance of Direct Digital Synthesis Systems

The previous analysis of the phase truncation spurs and periodic jitter spurs showed, that the spectral characteristic of a DDS system are dependent on the values of the NCO variables. A change of the desired output frequency f_{out} and therefore of the frequency control word Δ_f has following effects. The amplitude of the desired output signal changes its magnitude with the *sin(x)/x* characteristic, as showed in Figure 3-4. The reason is the ZOH circuit in the DAC. The next effect is that the different frequency control word causes the spurious components to occur at different frequencies and with different amplitudes. These two effects combined, i.e. amplitude of desired signal changes and spurs change, cause change in the relative difference between the desired signal and spurs, which causes a change of the spurious-free dynamic range (SFDR) of the system.

The spectral purity and sideband noise are the major drawbacks of the DDS systems. The main sources of spurious signals are the phase truncation, amplitude truncation, periodic jitter spurs and the DAC non-linearities (ZOH circuit effect, quantization error and timing jitter). The description of these sources of spurious signals was given in previous sections. The stability of the output frequency is actually slightly better than the clock frequency since some integration of the timing jitter occurs over several clock pulses.

The other limitation of the DDS systems is the available bandwidth. With respect to the Nyquist theorem and practicability for filter application, the signal lowpass bandwidth is two and a half to four times lower than the clock frequency. The issue is on one hand the speed of the digital signal generation NCO depending on the speed of the underlying hardware. The bandwidth of the DDS system is also dependent on the performance of the DAC. There are only a few DACs on the market with speed exceeding 1 GHz. Earlier, the main difficulty was considered to be the accuracy of the high-speed DAC rather than the inherent non-linearities of the DAC [43]. In the digital part of the DAC and the whole DDS system, it is possible to predict the exact nature of the spurs and quantify the effects for deterministic signals. GNSS signals are varying in Doppler, in number of channels in sight and in their relative power. As a result, a high number of scenarios of occurrence of spurs must be considered. In the analog part of the DAC the accurate quantification and prediction is not feasible [42].

3.2 GNSS Signal Simulation

The computational tasks included in digital part of a general GNSS signal simulator can be grouped to three basic modules. The simulation definition part computes an epoch based description of the user scenario, the signal definition module computes the signal parameters for each channel and the signal generation module generates the digital signal described by the signal parameters. Detailed description of signal simulator architecture is given in Chapter 4. The description of tasks of the simulation definition part is given in this section.

The user input of a simulator comprises description of satellite positions in form of almanac and ephemerides, input of simulation start time and duration, receiver position, speed and trajectory together with optional settings of features of the propagation channel. The description of satellite orbits and user position and time need to be processed to satellite-to-receiver range and to pseudorange length.

3.2.1 Receiver Position Computation

The receiver position is input to a simulator typically in form of geodetic coordinates in the WGS-84 ellipsoid. The geodetic coordinates are given by longitude λ in grades, latitude ϕ in grades and height h in meters above ellipsoid. The relationship between Cartesian ECEF coordinates and the geodetic coordinates is visualized in Figure 3-8. The following transformation procedure is based on [44].

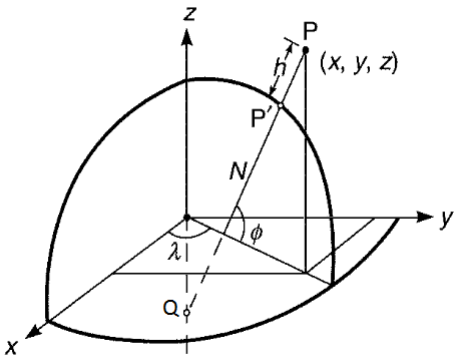

Figure 3-8 Relationship between Cartesian coordinates [*x, y, z*] and geodetic coordinates [λ, ϕ, *h*], [44]

The geodetic longitude is an angle between the x-axis and the projection of the meridian to the equatorial plane. The relationship between geodetic latitude and the Cartesian coordinate system is more complex. The distance from P to P', which is the length of the normal to the meridian ellipse, gives the height h. Prolonging the normal from P' to the crossing Q with the z-axis, we get the vector with length N. The angle between this vector and the equatorial plane gives the latitude ϕ. The distance N is important for definition of the transformation of the geodetic coordinates to Cartesian coordinates and it is calculated as

$$N = \frac{a}{\sqrt{1 - e^2 \sin^2 \phi}}. \tag{3-21}$$

The Cartesian coordinates [x, y, z] are computed from geodetic coordinates by

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} (N + h) \cos \phi \cos \lambda \\ (N + h) \cos \phi \sin \lambda \\ (N(1 - e^2) + h) \sin \phi \end{bmatrix}. \tag{3-22}$$

The inverse transformation from Cartesian coordinates to the geodetic coordinates follows an iterative scheme. First, the longitude is computed in a closed form easily by

$$\tan \lambda = \frac{y}{x}. \tag{3-23}$$

The geodetic latitude and the height over ellipsoid are computed iteratively in the following manner. First, the equation for the height is built. The equation (3-22) for x and y can be combined to the following relationship

$$\sqrt{x^2 + y^2} = (N + h)\cos\phi. \tag{3-24}$$

The equation can be also obtained from geometric description of the right triangle created from vector *QP* and *z*-axis. The height is then given by

$$h = \frac{\sqrt{x^2 + y^2}}{\cos\phi} - N. \tag{3-25}$$

Second, the equation for latitude is built from equation for coordinate z in (3-22) by division of each side of the equation by the respective left side and right side terms from (3-24) as

$$\frac{z}{\sqrt{x^2 + y^2}} = (1 - e^2 \frac{N}{N + h})\tan\phi. \tag{3-26}$$

The latitude is then computed from the equivalent expression as

$$\tan\phi = \frac{z}{\sqrt{x^2 + y^2}(1 - e^2 \frac{N}{N + h})}. \tag{3-27}$$

The iterative computation scheme starts with setting $h = 0$ and evaluation of (3-27). The next loop of the iteration evaluates the height h by (3-25) and the latitude ϕ by (3-27). This step is repeated, till the improvement of the values is lower than a given threshold.

For height h significantly lower than N the value quickly converges. For large h or ϕ close to $\pi/2$ other procedures are recommended [45].

3.2.2 Satellite Position Computation

The orbit description in form of almanac and ephemerides needs to be reprocessed to earth-centered and earth fixed (ECEF) coordinates. The almanac describes the satellite orbit as Keplerian orbit elements. The six Keplerian orbit elements a, e, ω, Ω, i, and m describe the orbit ellipse and position of the satellite at the time of ephemeris t_{oe}, as depicted in Figure 3-9. The axes X of the ECEF coordinate system are based in the center of Earth C. The axis X points towards the intersection between meridian 0° (Greenwich meridian) and the equator. The axis Z crosses the geographical north pole of the Earth (equivalent to the Earth spin axis). The axis Y is orthogonal to axes X and Z and forms a right-handed coordinate system.

The ellipse of the satellite orbit is defined by semimajor axis a, and eccentricity e. The satellite orbit forms an orbit plane, which crosses the equator plane at so called nodal line. The point K at the nodal line crossed by the satellite S on the way from the southern half space to the northern half space separated by equatorial plane is called ascending node. The angle between the axis X and the connection between ascending node K and the Earth center is called right ascension of the ascending node Ω.

The earth is place in one of the foci of the ellipse. The endpoint on the major axis closer to the Earth center C is called perigee and denoted P in the figure. The angle between K and the perigee P is called

argument of perigee ω. It increases counterclockwise perceived from the positive axis Z. The angle between the equatorial plane and the orbit plane is called inclination i.

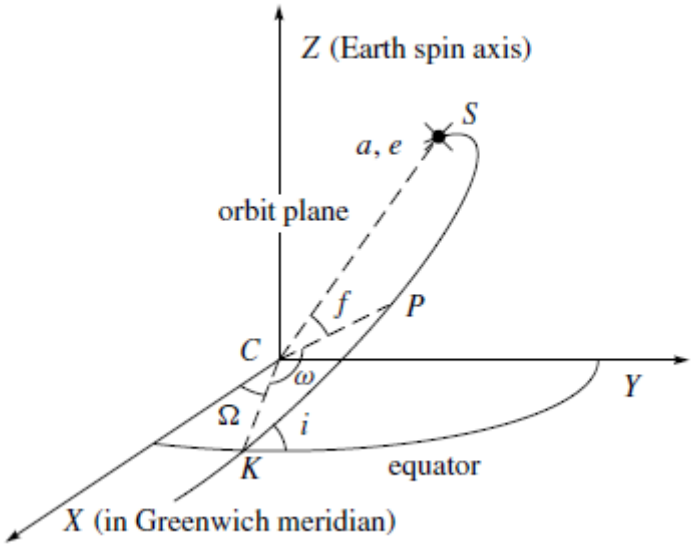

Figure 3-9 Kerplerian orbit elements of satellite *S* at the orbit around Earth center *C*

The angle between argument of perigee P and the satellite position is called true anomaly and denoted f in Figure 3-9. The position of the satellite in the orbit at the time of ephemeris t_{oe} is nevertheless given in the almanac by the mean anomaly m. The mean motion correction Δ_m is given in almanac to describe the satellite position. In the GPS data message, following ephemerides are transmitted [47]:

Parameter	Meaning, [units]
t_{oe}	Reference time of ephemeris as GPS time of week
$\sqrt{a}$	Square root of semimajor axis in [m]
e	Orbit eccentricity
i	Inclination angle (at t_{oe}) in [semi-circles]
Ω	Right ascension of the ascending node in [semi-circles]
ω	Argument of perigee (at t_{oe}) in [semi-circles]
m	Mean anomaly (at t_{oe}) in [semi-circles]
Δ_i	Rate of change of inclination angle in [semi-circles/s]
Δ_Ω	Rate of change of right ascension of the ascending node in [semi-circles/s]
Δ_m	Mean motion correction in [semi-circles/s]
$\delta\Delta_m$	Rate of mean motion difference from computed value [semi-circles/s^2]
C_{us}, C_{uc}	Amplitude of sine and cosine correction to argument of latitude in [rad]
C_{rc}, C_{rs}	Amplitude of sine and cosine correction to orbital radius in [rad]
C_{ic}, C_{is}	Amplitude of sine and cosine correction to inclination angle in [rad]

Table 3-1 GPS ephemeris parameters

The position of satellite *S* in ECEF coordinate system at time *t* is computed from the almanac data in a sequence of equation. First, the time epoch from time of ephemeris is set as $t_k = t - t_{oe}$. The mean anomaly at time t_k is computed as

$$m_k = m + \left(\sqrt{\frac{\mu}{a^3}} + \Delta_m + 0.5\delta\Delta_m t_k\right) t_k, \mu = 398600.5 \times 10^8 m^3/s^3. \quad \textbf{(3-28)}$$

The eccentric anomaly is then computed iteratively from the mean anomaly as follows

$$m_k = E_k + e \sin E_k. \quad \textbf{(3-29)}$$

The true anomaly is then evaluated from equations

$$\sin f_k = \frac{\sqrt{1-e^2}\sin E_k}{1 - e\cos E_k} \text{ and } \cos f_k = \frac{\cos E_k - e}{1 - e\cos E_k}. \quad \textbf{(3-30)}$$

Both equations are necessary, so as to compute the true anomaly in the correct quadrant. The argument of latitude is evaluated as $\phi_k = f_k + \omega$. The corrected argument of latitude is computed as

$$\mathrm{u}_k = \phi_k + C_{us}\sin(2\phi_k) + C_{uc}\cos(2\phi_k). \quad \textbf{(3-31)}$$

The corrected radius is computed as

$$\mathrm{r}_k = a(1 - e\cos E_k) + C_{rs}\sin(2\phi_k) + C_{rc}\cos(2\phi_k), \quad \textbf{(3-32)}$$

And the corrected inclination is given by

$$\mathrm{i}_k = i + \Delta_i t_k + C_{is}\sin(2\phi_k) + C_{ic}\cos(2\phi_k). \quad \textbf{(3-33)}$$

The corrected right ascension of the ascending node at t_k is computed as

$$\Omega_k = \Omega + (\Delta_\Omega - \Delta_{\Omega\mathrm{e}})t_k + \Delta_{\Omega\mathrm{e}} t_{oe}, \quad \textbf{(3-34)}$$

Where $\Delta_{\Omega e}$ denotes the rotation rate of the Earth. The value is set for GPS [47] to be $7.2921151467 \times 10^{-5}$ rad/s, which is equal to WGS-84 value used for navigation. The ECEF coordinates of the satellite *S* are then computed as

$$\begin{aligned} x_s &= r_k \cos u_k \cos\Omega_k - r_k \sin u_k \cos i_k \sin\Omega_k, \\ y_s &= r_k \cos u_k \sin\Omega_k + r_k \sin u_k \cos i_k \cos\Omega_k, \\ z_s &= r_k \sin u_k \sin i_k. \end{aligned} \quad \textbf{(3-35)}$$

3.2.3 Pseudorange Computation

The satellite position at a given time is computed from the input satellite orbit description. The true receiver time is set by the user. The transmission time t_{tr} for the signal received at a given time t_{rec} is

not known. In order to calculate the range and pseudorange between the satellite and the receiver position, the transmission time t_{tr} must be calculated. The true range is the distance between the satellite position at time t_{tr} and the receiver position at time t_{rec}. The calculation of transmission time t_{tr} can be solved by the iterative method.

First, the starting transmission time in the first iteration $t_{tr,0}$ can be set to be equal to the receiver time t_{rec}. Then the position of the satellite p_{sat} expressed in ECEF coordinates is evaluated for the receiver time from the orbit description and

$$p_{sat}(t_{tr,0}) = p_{sat}(t_{rec}) \tag{3-36}$$

The range between the satellite position p_{sat} and receiver position p_{rec} is calculated as

$$r_0 = \left|\vec{p}_{sat}(t_{tr,0}) - \vec{p}_{rec}(t_{rec})\right|, \tag{3-37}$$

Or equivalently as

$$r_0 = \sqrt{\begin{matrix}(x_{sat}(t_{tr,0}) - x_{rec}(t_{rec}))^2 + (y_{sat}(t_{tr,0}) - y_{rec}(t_{rec}))^2 \\ +(z_{sat}(t_{tr,0}) - z_{rec}(t_{rec}))^2.\end{matrix}} \tag{3-38}$$

The computed range is adjusted by the travel time error δr_{travel} in [m], so as to get the real travel time of the signal in meters as

$$\rho_0 = r_0 + \delta r_{travel}. \tag{3-39}$$

The travel time error is calculated as ionospheric and tropospheric error for given $t_{tr,0}$ and t_{rec} between satellite position $p_{sat}(t_{tr,0})$ and receiver position $p_{rec}(t_{rec})$ using the respective ionospheric and tropospheric models.

The next iteration of the computation starts with the approximated transmitting time calculated from the traveling time of the signal as

$$t_{tr,1} = t_{rec} - \frac{\rho_0}{c}, \tag{3-40}$$

where c denotes the speed of light in vacuum. The new transmitting time value is then applied to the equation (3-37) in the next iteration to compute the range as

$$r_1 = \left|\vec{p}_{sat}(t_{tr,1}) - \vec{p}_{rec}(t_{rec})\right|. \tag{3-41}$$

The value of the transmission time is refined in each iteration till the difference between the last and the current range becomes less than a defined threshold. The computation is finished when the following condition holds:

$$r_{diff,max} \geq |r_i - r_{i-1}|. \tag{3-42}$$

During the iterative computation of the transmission time, the satellite position at t_{tr} is either computed from the orbit description with the method described in previous section or interpolated between two analytically computed satellite position values.

With addition of receiver clock error δt_{rec}, the satellite clock error δt_{sat}, and the travel time error $\delta t_{travel} = \delta r_{travel}/c$, the pseudorange between the satellite and the receiver can be calculated as

$$\rho = r + (\delta t_{rec} - \delta t_{sat} + \delta t_{travel})c. \tag{3-43}$$

In this computation method the relativistic effect is neglected.

3.2.4 Azimuth and Elevation

The azimuth and elevation of each satellite serves for computation of atmospheric signal delays. For application of elevation mask and any more complex multipath modelling scheme, the azimuth and elevation of satellites relative to receiver position are also necessary. The elevation angle γ and azimuth α are defined as angles in local coordinate frame [*E, N, U*] of the receiver *P* as depicted in Figure 3-10.

Such a coordinate frame is called a local-level system or east-north-up (ENU) system. So as to get the expected azimuth relative to the north direction and elevation angle relative to the zenith direction, one axis points to the north, one upwards and the third is perpendicular to the two so as to form a right hand coordinate system. The transformation between the ECEF coordinates and the ENU coordinates is actually a transformation between two Cartesian coordinate systems. When two such systems have the same origin and are both right-handed, the transformation is just a series of three rotations around *x*-axis, *y*-axis and *z*-axis.

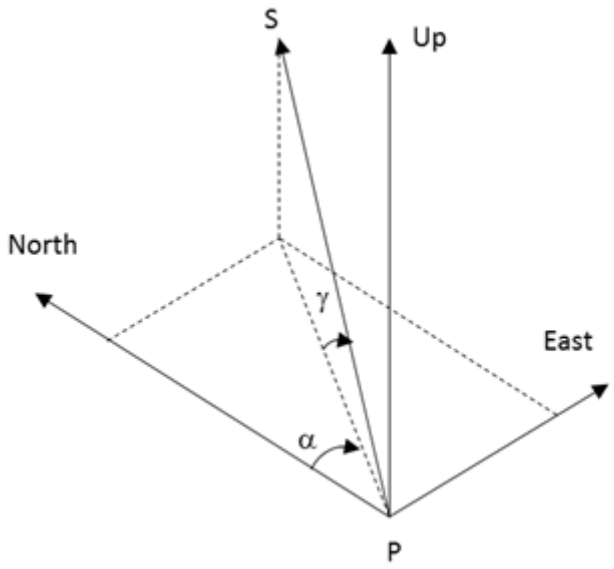

Figure 3-10 Azimuth (α) and elevation angle (γ) of a satellite *S* in the local coordinate system of receiver *P*

Let the positive rotation around axis *a* be defined as counterclockwise when looking toward the origin from the positive end of the axis *a*. The rotation of vector p_A around *x*-axis by angle θ is then defined by rotation matrix R_1 as follows

$$\vec{p}_B = \boldsymbol{R_1}(\theta)\vec{p}_A, \boldsymbol{R_1}(\theta) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & \sin\theta \\ 0 & -\sin\theta & \cos\theta \end{bmatrix}. \tag{3-44}$$

The rotation around *y*-axis by angle θ is then defined by rotation matrix R_2 as follows

$$\vec{p}_B = \boldsymbol{R_2}(\theta)\vec{p}_A, \boldsymbol{R_2}(\theta) = \begin{bmatrix} \cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta \end{bmatrix}. \tag{3-45}$$

The rotation around *z*-axis by angle θ is then defined by rotation matrix R_3 as follows

$$\vec{p}_B = \boldsymbol{R_3}(\theta)\vec{p}_A, \boldsymbol{R_3}(\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}. \tag{3-46}$$

In order to determine the transformation from ECEF to ENU, we have to find out the rotation angles that would bring the axes into coincidence. For this procedure, the transformation of the position of the receiver in Cartesian coordinates need to be transformed to geodetic coordinates [λ, ϕ, h] with procedure described in previous section When we start the procedure with the rotation about the *x*-axis, we will need to rotate by an angle (λ + 90°). The x-axis is now directed in the same way as the east axis in the ENU system. Then the next rotation is done around the z-axis by an angle (90° - ϕ). There are also alternative series of rotations with the same result. The ECEF and ENU coordinate systems do not have the same origin, therefore the rotation must be preceded by a subtraction of the vector connecting the origins of the systems from the point to be transformed.

In the ENU coordinate system the elevation angle γ and azimuth α described in Figure 3-10 are computed easily. Elevation angle is given by

$$\tan\gamma = \frac{U}{\sqrt{E^2 + N^2}}. \tag{3-47}$$

Azimuth is given by

$$\tan\alpha = \frac{E}{N}. \tag{3-48}$$

3.2.5 Clock Error

The key payload of a GNSS satellite is the precise atomic clock. It controls all onboard timing operations and the broadcast signal generation. Despite the effort to deploy a highly stable clock, correction data is needed to compute precise position solution. In GPS NAV message the clock correction fields can hold a deviation as large as 1 ms [46], which translates to a 300-km pseudorange error.

The satellite clock error δt_{sat} is the effective code phase offset of a satellite referenced to the phase center of the antenna with respect to GPS system time at the time of data transmission. The clock error is described in the data message of the GNSS signal service and defined in ICD [47]. In the GPS NAV message, three coefficients for the polynomial defining the clock offset are broadcasted. This estimated correction accounts for the deterministic clock error characteristics of bias, drift and aging together with the group delay bias and mean differential group delay. The coefficients do not account for the relativistic effects, which must be computed internally. The clock offset is given by the function

$$t = t_{sat} - \delta t_{sat}. \tag{3-49}$$

The parameter t stands for GPS system time, t_{sat} is effective code phase time at message transmission time and δt_{sat} is the code phase time offset. All three parameters are expressed in seconds.

The code phase time offset is computed using the coefficients and the age of the coefficients in the message as

$$\delta t_{sat} = a_{f0} + a_{f1}(t - t_{oc}) + a_{f2}(t - t_{oc})^2 + \delta t_{rel}, \tag{3-50}$$

Where a_{f0}, a_{f1} and a_{f2} are the coefficients of the second-order polynomial. Parameter t_{oc} is the clock data reference time in seconds. In the equation additionally the δt_{rel} is applied. It stands for the relativistic correction term expressed in seconds. The term is given by

$$\delta t_{rel} = Fe\sqrt{A}\sin E, \tag{3-51}$$

Where *e, A* and *E* stand for eccentricity, semimajor axis and eccentric anomaly of the orbit of the satellite, while *F* is a constant with value of -4.4428 × 10^{-10} computed from Earth's gravitational parameter μ and speed of light *c* as

$$F = \frac{-2\sqrt{\mu}}{c^2}. \tag{3-52}$$

The equations (3-49) and (3-50) are interdependent and cannot be evaluated separately. In the ICD, the recommendation is given to set *t* to the value of t_{sat}. Then the equation (3-50) can be solved and the clock error value can be applied to the pseudorange calculation given by (3-43).

In a simulator, different model can be taken for broadcasted and true simulated clock error. The NavX-NCS simulator by IFEN GmbH generates the clock error with second-order polynomial as defined in (3-50) and optionally adds residual satellite clock errors. The broadcasted clock error is dithered by three different random processes describing the stability of a frequency standard in the time in the form of the Allan deviation. The random processes are given by the parameters for the phase white noise, frequency white noise and frequency random walk. The procedure of derivation of the true clock error from the broadcast data have disadvantage of abrupt change of clock behavior in case of broadcast clock parameter update [48].

3.2.6 Multipath Signal

Multipath effect is a significant contributor to the measurement error in GNSS receiver. It occurs, when the signal incoming to receiver is mixed with reflected or diffracted replicas of the satellite signals. Since the reflected and diffracted signal path is always longer than the direct path, the reflections are delayed relative to the direct signal arrival. Multipath signal is thus a signal composed of a direct signal called line-of-sight (LOS signal) and a number of delayed reflections of the direct signal. Mathematically it can be described as

$$\begin{aligned} s(t) &= A_0(t)C(t-\tau_0(t))\exp(-j(f2\pi(t-\tau_0(t))+\varphi_0(t))) + \\ &\sum_{n=1}^{N} A_n(t)C(t-\tau_n(t))\exp(-j(f_n 2\pi(t-\tau_n(t))+\varphi_n(t))) \end{aligned} \tag{3-53}$$

In the equation, *N* is number of multipath reflections, *C(t)* stands for the spreading code with data bits, $A_0(t)$ is the amplitude attenuation of the direct path and $A_n(t)$ are the amplitude attenuations of the multipath returns. Symbol ϕ_0 stands for the carrier phase of the direct path and ϕ_n is the carrier phase of each multipath return. The propagation delay of the direct path is given by τ_0, the τ_n is the propagation delay of each multipath return, *f* is carrier frequency of the direct path and the f_n is the received frequency of each multipath return.

The parameters in (3-53) vary in time due to the motion of satellites, receiver and objects in the surrounding that act as reflectors. For sake of readability, this variation is not explicitly expressed in the equation. As well as the later reflections, the LOS signal can be attenuated by shadowing or even not visible, so that the first coming signal is already a reflection.

Improvements in the GNSS system and addition of augmentation solutions have reduced many sources of system errors and improved modelling of global propagation errors. These improvements left multipath and shadowing be in many cases the dominant contributor to the measurement error. When the delay of the reflections relative to the LOS signal is larger than twice the chip length, receiver can easily separate the LOS signal and reflection correlation peaks. As long as the LOS signal is tracked, the multipath has in this case little effect on the receiver measurement precision. When the multipath is caused by near-by objects or when reflecting at distant object with grazing angle of incidence, reflections with short delays can occur. Such reflections distort the correlation function between the composite signal and the replica and introduce errors in code and carrier measurements and results in range errors of up to a chip length equivalent.

Even more sever measurement error can occur, when the LOS signal is obstructed or shadowed. Shadowing is excess attenuation of the direct path causing LOS signal to be weaker than the multipath reflections. This situation can occur in both indoor and outdoor scenarios.

4 Simulation of Multipath GNSS Signal

Multipath is a significant propagation error influencing the precision of satellite to receiver range measurement. Algorithms for effective mitigation are still being developed and sensitivity of receivers to this error examined. The multipath signal generation is therefore an important simulator capability. All other simulated errors and effects can be expressed in terms of basic signal parameters – phase, frequency and amplitude of carrier and code. The multipath effect is an exception, as the multipath signal reflections are independent signal channels with their own phase, frequency and amplitude relative to the line-of-sight signal parameters.

Various models of the multipath exist with various level of simplification of the general multipath description given in section 3.2.6. The traditional approaches characterizing multipath focus on the security relevant outdoor scenarios, especially the airborne applications (e.g. [49]). Later, the land mobile applications spread and multipath has grown in relevance. In GNSS positioning, the extension of signal reception to indoor scenarios stressed the need for multipath mitigation solutions.

This development was caused by increase in sensitivity of GNSS receivers. In the mid-2000s, the sensitivity of commercial GPS chips improved by 25 dB or more. Starting with the SiRF Star IV chipset generation from CSR, signals as weak as -193 dBW could be tracked [50]. Contribution to this development was also made by the Assisted-GPS. This service makes it possible to download the ephemeris data in advance over the cell-phone channel instead of reading them from the GNSS message. The acquisition of the signal is then possible with lower C/N_0 and acquisition time is shortened [46]. Thanks to these solutions and numerous other technologies, the way for extension of focus of multipath mitigation from harsh outdoor environment, urban canyons and tree foliage, towards indoor environment was paved.

It needs to be mentioned that in a realistic indoor scenario not only effects caused by propagation in the indoor environment need to be taken into account, but the effects happening outdoors before the signal enters the building must be taken into account as well. In the outdoor environment the effects are in general multipath propagation, obscuration and diffraction by near and far objects. In indoor environment the power loss at wall transmission and multipath reflection on indoor objects are the key obstacles for precise position fix. Therefore a realistic simulation of GNSS in indoor environment should represent all these significant sources of signal impairments.

Numerous commercial simulators from middle to top-end solutions feature the generation of multipath effect. The overview is given in Table 2-2 in Chapter 2 GNSS Signal Simulators. The simplest approach offered e.g. by older Spirent simulators is the manual allocation of individual signal channels to individual multipath reflections, where the user needs to generate multipath parameters on his own. Such an attitude allowed users to describe any specific case of multipath, e.g. the typical airborne reflections from the wings and ground reflections during landing. It is nevertheless unpractical for the user in land mobile applications, such as a cell phone usage. The tests required to verify the positioning performance of the handset should include a wide set of different environments. The variability of the environment in time due to user's dynamics would need many short tests for each site.

Higher level solutions offer a statistical multipath model for outdoor environment. For Spirent devices the solution published in [51] applied the wideband Land Mobile Satellite model [52]. Furthermore,

the deployed Spirent simulator offered a simulation of signal impairments caused by environment using an obscuration mask. An obscuration mask describes a user defined environment in form of a mask of azimuth and elevation bins. The bins state where is placed which type of obscuration. Reflections are automatically generated and allocated to signal channels before the signal generation starts. For the whole simulation period, a single static setting of channels is applied.

Another solution or refinement to other multipath simulation methods arises for the user by the use of antenna gain pattern modeling. This feature is available in most GNSS simulators. The gain pattern attenuates all signals including echoes and is independent of user dynamics. It is suitable for simulation of the effect of head shading or effect of car roof over a receiver.

The support for multipath simulation in commercial simulators is focused on outdoor environment. Such support is less suitable for indoor environment, because there the multipath effects evince significant differences to outdoor environment. In buildings, the obstacles causing reflection are numerous, very variable and they are placed in low distance to the receiver. Both the signals penetrating the obstacles and the reflections reach the receiver. The result is a high and dynamic number of different reflections with dynamic behavior even for a static receiver caused only by the satellite motion. This can hardly be covered by the solutions for outdoor environment. A static set of reflections for whole simulation period would be a clear disadvantage. Also the limited number of channels in commercial hardware simulators limits the number of reflections more or less significantly. The environment mask applied for initialization of reflection rays does not account well for dynamics of indoor environment.

The work described in this chapter was a part of a larger project focused on Indoor positioning (called INDOOR in this text). In the framework of the project, theoretical models and measurement campaigns were performed for the characterization of the GNSS signal in indoor environment and development of positioning solutions [53].

The objective of this chapter is the combination and parametrization of the models and measurement results obtained within the project so as to implement a realistic high-speed indoor GNSS signal simulation in the institute's own DIF signal generator introduced in [26]. The simulated indoor signal is compared to a real scenario to evaluate its performance.

4.1 The Model Chain for Characterization of Indoor Environment

A satellite signal propagates from a satellite into a building through different environments. The environment where multipath occurs can be divided into three types. The first is the outdoor environment with far reflectors and obstacle shadowing. The second is the propagation through building walls that attenuates the signal significantly. The third is indoors, where the near obstacles cause numerous reflections.

A realistic characterization of the indoor GNSS signal for signal simulation should take these effects into account. For this purpose, a combination of multiple models to a model chain is proposed. It represents the three environment types as depicted in Figure 4-1

The outdoor environment is described with the modified narrowband Land Mobile Satellite (LMS) model [54], already implemented in the institute's own DIF signal generator. The transmission through walls is modeled using the results from [55] developed within the INDOOR project. The characteristics of reflections in indoor environment are given by satellite-to-indoor channel model

developed within the INDOOR project and introduced in [56]. It is a modified version of the Saleh-Valenzuela model proposed in [57].

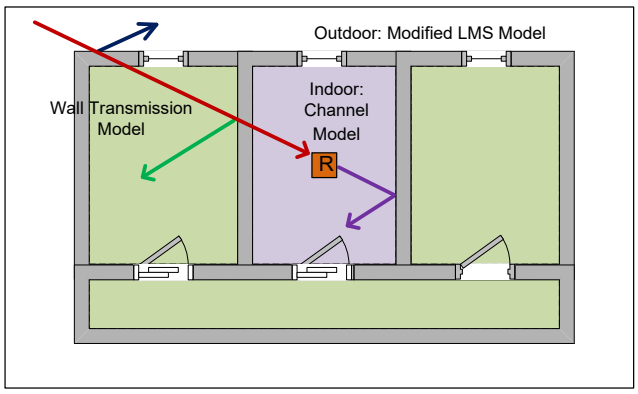

Figure 4-1 Model chain for characterization of the satellite-to-indoor channel

4.1.1 Modified Land Mobile Satellite Model

The description of the outdoor environment is realized by the narrowband Land Mobile Satellite (LMS) model developed at the German Aerospace Agency [54]. The model describes signal fading due to multipath for L-band frequencies. A narrowband version [54] and a wideband version [52] of this model exist, the narrowband version was chosen by the authors of the institute's own DIF generator. It is a two-state model classifying every epoch as a line-of-sight (LOS) state or a non-LOS state respective to each satellite. These two states are switching according to a Markov process. In LOS states, the fading is described by a Rician process, in non-LOS by a Rayleigh process.

The LMS model is combined with the computation of phase rotations of the fading signal by means of the Jakes function, introduced in [58]. The scheme of the concept is given in Figure 4-2. The resulting model, here called modified LMS model, had been proposed and deployed in the institute's own DIF generator by its authors.

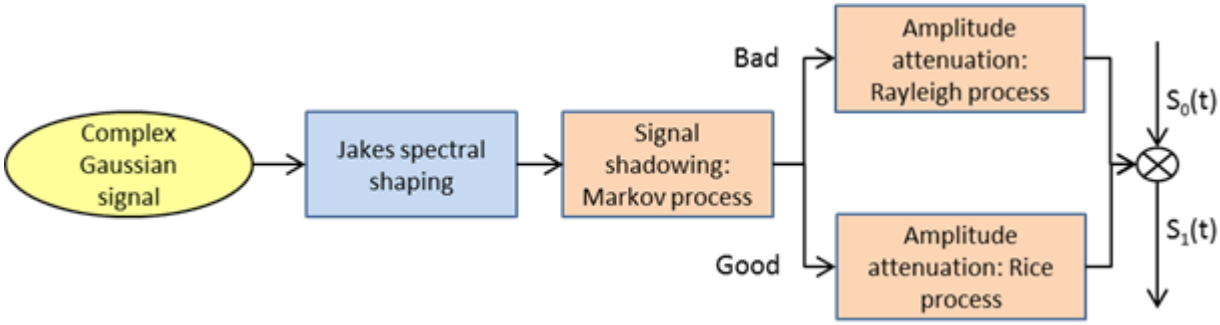

Figure 4-2 Modified LMS model for outdoor environment

4.1.2 Wall Transmission Model

To characterize the transmission of the signal from outdoor environment into indoor environment, the results of characterization and measurements of transmission of GNSS signal through walls presented

in [55] were adapted. The model computes simplified geometrical propagation through architecture, considering walls to have standard thickness and the materials to be homogeneous and isotropic. The model is based on the reflection and refraction features of an electromagnetic signal at an interface between two media. Simple description is given in Figure 4-3. The signal approaching the interface, s_{in}, is partially reflected as signal s_r, and partially transmitted and refracted as signal s_t. Both the reflected and the transmitted signals are influenced by factors describing the nature of the materials. These factors are permittivity ε, permeability μ, and propagation speed of signal in medium v. The important consequence of the effect is the attenuation of the transmitted and reflected signal in comparison to the original signal.

For description of the attenuation the reflection coefficient ρ and the transmission coefficient τ are used. The coefficients are defined by the ratio of the amplitudes of the reflected or transmitted waves and the incident wave. Their values depend on the angle of incidence and on the materials and surface of the media at the interface. An example of a material with permittivity $\varepsilon_r = 3$ is shown in Figure 4-4.

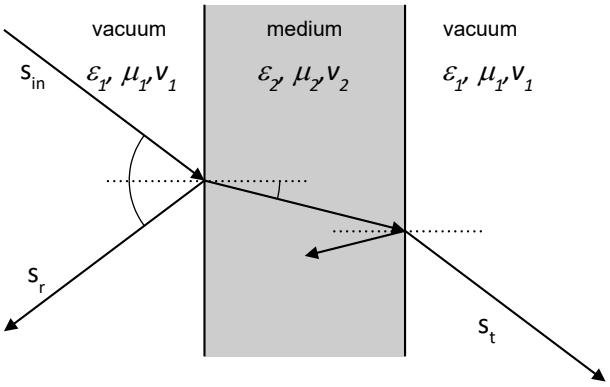

Figure 4-3 Reflection and refraction of a signal at an interface of two media

The polarization state of the incident wave also influences these coefficients. The influence of polarization state is significant especially for the reflection coefficient; the transmission coefficient is influenced only marginally [55].

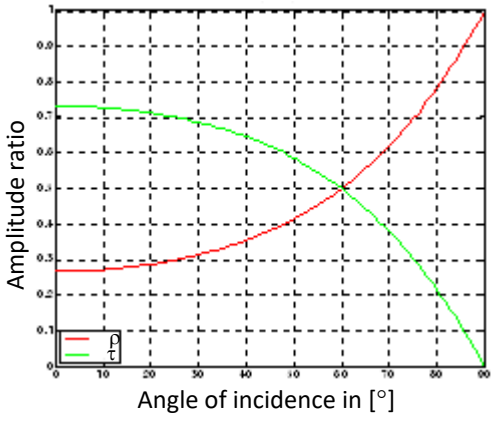

Figure 4-4 Reflection (red) and transmission coefficient (green) at an interface between vacuum and a material with $\varepsilon_r = 3$ for vertical polarization [55]

For the characterization of the transmission of the signal into indoor environment a simplification of the geometrical transmission is proposed. The role of this model, as depicted in Figure 4-1, is to describe the influence of wall transmission on signal propagating to the receiver on the direct line. The reflections on outer building walls are reflected back to outdoor environment (blue). Reflections at an inner wall (green) that might propagate to the receiver are possible, but typically weaker than the LOS signal. Although such reflections can come to the receiver through walls and other obstacles different from line of sight, the model chain simplifies the situation and regards them to be reflections of the LOS signal (red). These are covered by the indoor channel model. The reflections of the signal in the room with the receiver (violet) are also covered by the indoor channel model.

As a result only the LOS signal is taken into account by geometrical wall transmission modeling and no reflections are considered at this step of the model chain. The effect of polarization is neglected, because in case of transmission it is minor as it was mentioned before.

In implementation of the model chain, a simplified description of the test site building needs to be constructed. The values of the transmission coefficient and reflection coefficient measured in [55] are used. The building walls are regarded to have the same parameters as in the measurement and the values are used without modification. For model of the test site building used later in this work the values listed in Table 4-1 were used.

The building is modelled as a structure of rectangular rooms with one outdoor wall made from the defined materials. An example of such a model building with its ground plan and side profile in depicted in Figure 4-5. It is a simplification of an office building with brick walls, glass windows and concrete ceilings. The roof is expected to be flat.

L1 signal at 1575.42 MHz	Transmission coefficient τ	Attenuation inside medium
Dry wall	0.5788	0.995
Glass	0.5789	0.9867
Lumber	0.7554	0.7621
Bricks	0.6547	0.4624
Concrete	0.5964	0.1849

Table 4-1 Transmission and attenuation factors for L1, incidence angle 0°

For each satellite elevation and azimuth pair, the number of crossed walls and respective attenuation together with loss through partial reflection are computed as follows:

$$\mathrm{A} = \prod_{w \in W} \tau_0(I_w)\tau_{m(w)}A_{m(w)} \tag{4-1}$$

The value W stands for all crossed walls in term of the material, $m(w)$ is material of the wall, τ_0 is normalized attenuation function as depicted in Figure 4-4 based on the incidence angle at wall I_w, $\tau_{m(v)}$ is the transmitted part of the signal and $A_{m(w)}$ is the attenuation of the wall material.

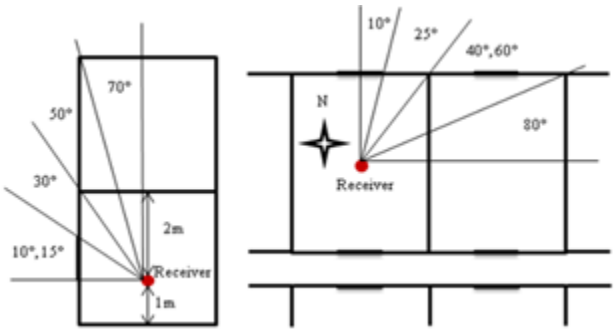

Figure 4-5 Example of a building model – side profile with elevation angles and ground plan with azimuth

4.1.3 Indoor Channel Model

The last element of the chain is the indoor channel model based on Saleh-Valenzuela model for indoor GNSS signals published in [56]. This model characterizes statistically the indoor part of the satellite-to-indoor channel for satellite navigation purposes. The main theoretical reference for the model is the work of Saleh and Valenzuela [55]. The modified indoor channel model adapted the original schema to the satellite-to-indoor channel according to an analysis of field measurements.

The model describes the coming signal as a set of clusters (i.e. packages) of signal rays. The clusters differ in arrival time and number of comprised rays. The rays in the clusters have different amplitudes, time spacing and span over different time periods. Indeed, the subsequent clusters are attenuated in amplitude in contrast to the first one, and also the amplitudes of rays within single cluster decay with time at a different rate. The model proposes that both of these decaying patterns, namely that associated to a particular cluster and that within the cluster, follow an exponential function of time. These exponential decaying patterns are controlled by two time constants: Λ for the cluster arrival decay time and Λ for the ray arrival decay time. The idea of reflections organized in clusters is visualized in Figure 4-6.

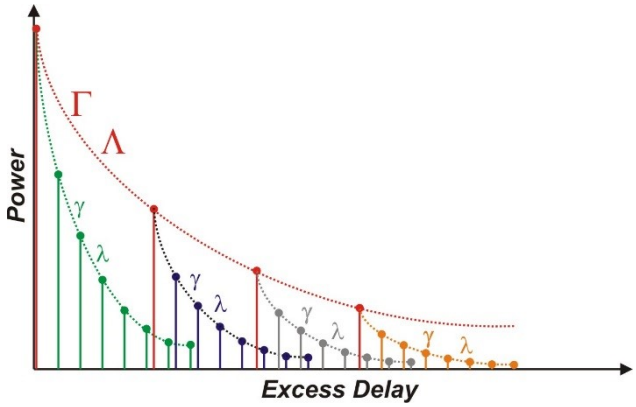

Figure 4-6 Indoor model - power decay profile [56]

The amplitude of each cluster and its rays decays exponentially by the order of exp(-1/Γ) and exp(-1/γ). The phase associated with each arrival is expected to be a statistically independent and uniformly distributed within [0, 2π).

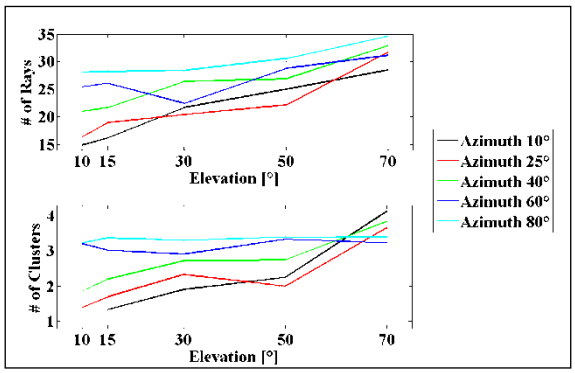

Figure 4-7 Mean number of clusters and rays [56]

As examined in [56] the parameters of the model Λ, λ, Γ, γ and number of in each cluster show dependency on azimuth and elevation. The mean values for number of clusters and rays are given in Figure 4-7.

The indoor channel model does describe the number and character of the reflections, but it does not comment on time behaviour of the reflections. The static allocation of chosen rays to limited set of channel for the whole simulation period does not account for the variable character of multipath well, as it was mentioned in [51]. In indoor environment the variability is even stronger and influences majority of tracked signals.

Therefore in this work a computation of a set of new reflections on a regular basis in order of seconds with polynomial interpolation between the successive values is proposed. The objective of this approach is to ensure variability of the reflections in time with smooth course over the simulation period so as to approach the features of real multipath effect.

The issue of this approach is the way, how the rays are ordered for interpolation. Examples of two following epochs of a simulation are depicted in Figure 4-8. In this work the rays are sorted in clusters and then the cluster are sorted according to time delay. This approach is evaluated in the section 4.5 Comparison of Simulation and Field Test.

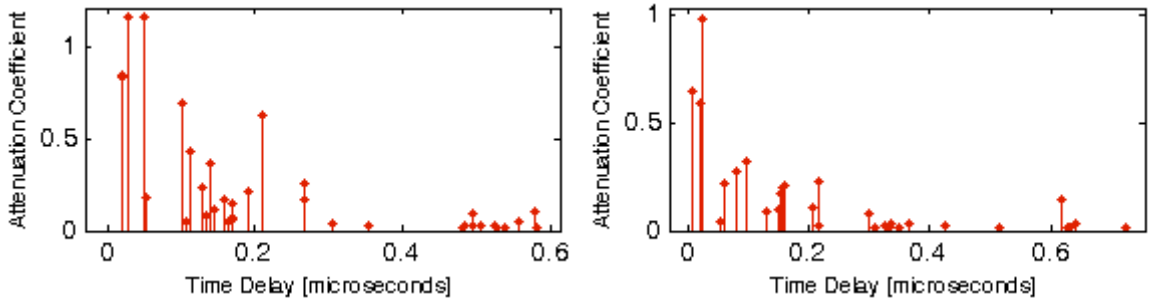

Figure 4-8 Epoch 1 of the simulation, 34 reflections - left, epoch 2 of the simulation, 29 reflections

4.2 Implementation in DIF Signal Simulator

The GNSS DIF signal simulator used for multipath generation was developed at the Institute ISTA of Universität der Bundeswehr München [26]. The DIF simulator is a software tool running optimized for operation on an Intel CPU running with a speed below real time. Advantages for usage as a platform for implementation of the multipath module are the high volume CPU memory and general

flexibility of CPU programming. It can therefore generate unlimited number of channels and is suitable for alternative signal generation algorithms.

The computation demand of a generation of multipath signal with many reflections is generally high and number of channel of simulators is always limited. For the multipath model chain proposed in this work a special multipath signal generation algorithm was developed. The idea of this algorithm is to exploit the nature of a CPU platform and similarity of multipath reflections with the original LOS signal. On Intel Core 2 Quad CPU, it improves the speed by one third in comparison to the independent channel generation. The algorithm and its performance is described in [58].

This algorithm generates multipath rays from the original signal by time shift, phase shift and attenuation. It introduces some simplification with respect to equation (3-51), as no Doppler frequency difference between the original signal and a reflection is admitted. The reduction to no Doppler difference is in case of indoor environment acceptable, as the reflection rays are generated at near objects, which are static, and the user speed is typically very low.

Another simplification is given by the fact, that the time shift is rounded to whole number of samples and therefore code and carrier skew in multipath channels is introduced and the possible code phase of the channels is limited to number of samples per chip. With low sample rate and high number of reflections, it is probable that several multipath channels fall into the same code phase bin and receiver can compute higher correlation value for this code phase than for LOS signal with higher power than any single reflection.

4.3 Field Test Verification Scenario

A field test was performed to verify the applicability of the proposed model chain. The test was located at the campus of Universität der Bundeswehr München. The south wing of the building 42 served as the test site. The ground plan is depicted in Figure 4-9.

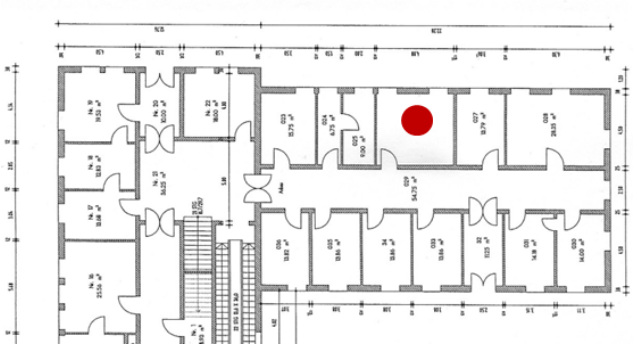

Figure 4-9 Test site at the campus of Universität der Bundeswehr München

The building is a simple office building with only one floor made of bricks walls, concrete ceilings and gable end roof. The measurement was done on 11 January 2011 with the duration of 20 minutes. The plot of satellite position is depicted in Figure 4-10. Two identical receivers were used for the measurement. The chipset of both receivers was the low-cost high-sensitivity SiRF Star III receiver used often in cell phones and other consumer handheld devices. It was integrated in the Garmin 20x navigation device. One of the receivers was used as the primary test receiver at the test site. The other one was located at a nearby outdoor location with a clear view of the sky to facilitate comparable measurements.

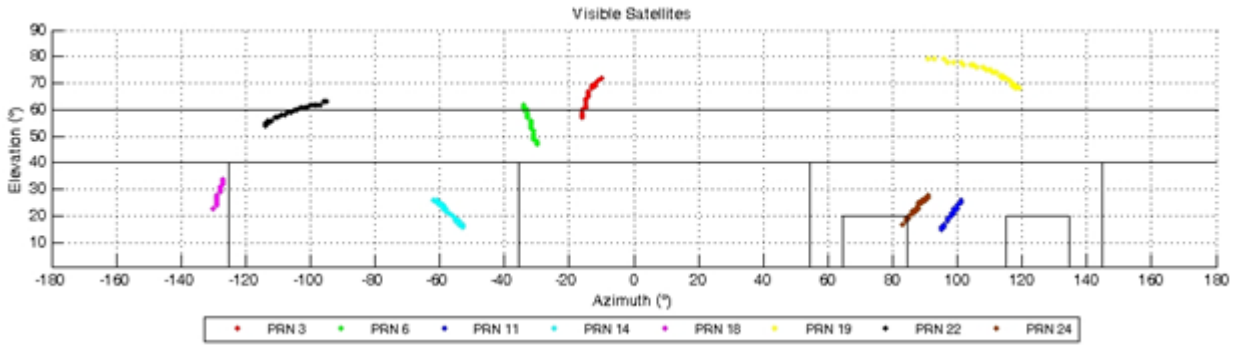

Figure 4-10 Circular view of the test site with satellite positions during the test

The signal attenuation at the test location was measured by differencing the test receiver carrier-to-noise density ratio data with that from the reference station receiver of similar type. This operation is shown in the following equation:

$$F = \frac{C_{ref}}{N_{0ref}} - \frac{C_{rover}}{N_{0rover}} \; [dB] \tag{4-2}$$

Where F is fading in dB, $C/N_{0(ref)}$ is carrier to noise density ratio for the reference receiver in dB and $C/N_{0(rover)}$ is carrier to noise density ratio for the test location receiver expressed in dB. This provides a good measure of the amount of signal fading at the test location.

From the eight satellites in view only six that were observed by the receiver for a longer period are included in the fading comparison. The fading values are depicted in Figure 4-11. The graphs display the well-known interference behavior associated with multipath and slowly changing mutual carrier phases [60]. Only in case of the PRN 19 the effect is less obvious. The reason for this is probably the high elevation of the satellite and the fact that the building has only one floor and therefore there is just little possibility for high elevation signals to reflect.

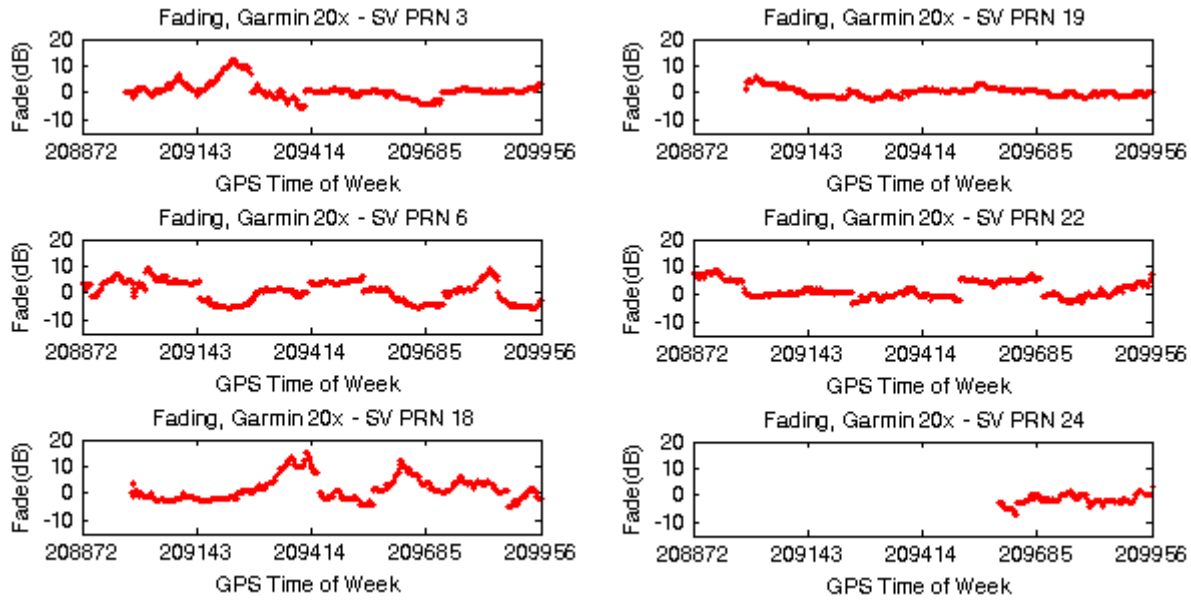

Figure 4-11 Signal fading measured at the test site

For the evaluation of the multipath effect on pseudorange measurements and resulting position precision the comparison of measured data and known position obtained from geodetical measurement was done. Figure 4-12 shows horizontal and vertical error measured during the campaign. The first minutes were necessary to warm up and then the position error remains quite smooth. The pseudorange measurement and comparison of residuals was not possible, as the used receiver is limited to NMEA output where pseudorange measurements are not included.

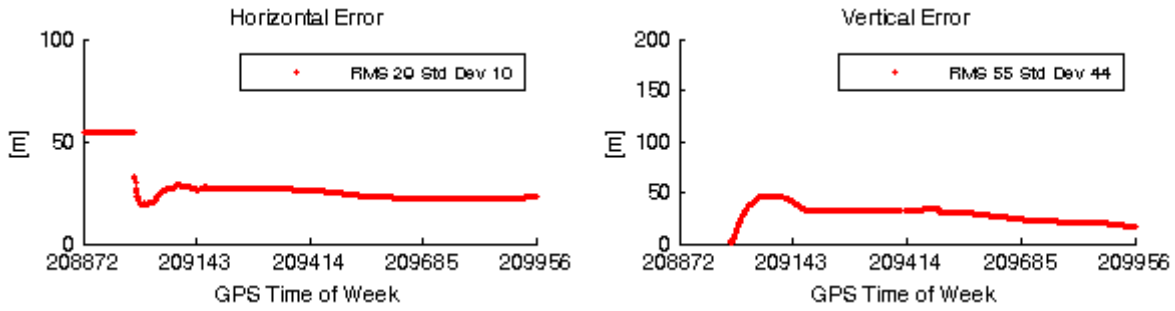

Figure 4-12 Horizontal and vertical error measured at the test site

4.4 Simulated Verification Scenario

The DIF simulator was set up to simulate the same period as the field test. The GPS constellation was described by a SP3 orbit file for the day of the field test. The epoch length for multipath description was set to 1 second. Ionospheric delay, tropospheric delay, noise and clock error were simulated to keep consistency with the field test.

The environment for the outdoor modified LMS model was set to suburban. The model of the test site building depicted in Figure 4-13 is a simplification of the architecture of the building wing. All the observed satellites were simulated.

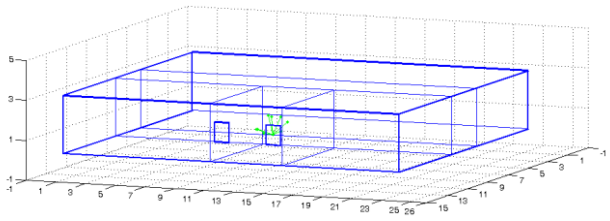

Figure 4-13 Model of the test site building at the campus of Universität der Bundeswehr München

The front end and antenna description of the DIF simulator was set to describe a general patch antenna and front end of a low-cost chipset. The simulated data were received by a software receiver accepting IF sample file input.

4.5 Comparison of Simulation and Field Test

The receiver performance at the test site and the receiver reaction to the simulated signal were compared. The comparison of the fading F as defined in (4-2) is presented in Figure 4-14 for the six SVs in sight. The RMS and the standard deviation (1 σ) of the fading values are listed in each graph.

The RMS of the fading values shows good correspondence in case of PRN 18 (simulated scenario RMS = 4 dB, field test RMS = 5 dB). In case of the other PRNs, the simulation modeled deeper fading than what occurred during the test. The PRNs 3, 6, 19 and 22 all penetrated to the receiver through the ceiling. This shows that the assumption for the concrete roof expected higher attenuation than the value measured at the test site. The simulation of propagation of the signal from PRN 18 through two walls estimated the situation better. PRN 24 had a low elevation between 18 and 28 degrees. In the field test, it was tracked first when it crossed 25 degrees, roughly at the point where it crossed only one wall and once the roof. The simulation receiver was able to track the signal penetrating three walls, then for two walls and ceiling the signal was too weak and again for wall and ceiling the tracking was successful. In the field test, probably the tree shadowing behind the building prevented the receiver from tracking the weak signal. Then again the simulation of much stronger attenuation in the roof is probably the reason for mismatching RMS and later the start of tracking at a higher elevation.

The standard deviation shows for all PRNs reasonable correspondence between simulated and field data. The course of the fading shows for the simulation the sinusoidal pattern typical for multipath, but with much higher frequency. It shows that the interval for multipath epochs of one second is too short for the static scenario and it would need an adjustment.

The horizontal and vertical error values are compared in Figure 4-15. The position solution from the simulated data was computed without smoothing. Therefore, the variance cannot be reliably compared. For the horizontal error, the RMS values show good correspondence. In case of vertical error, the difference is higher. The addition of the PRN 24 to position computation improved the precision thanks to a better geometry in case of field test as well as in case of simulation, where the inclusion after loss came about five minutes later.

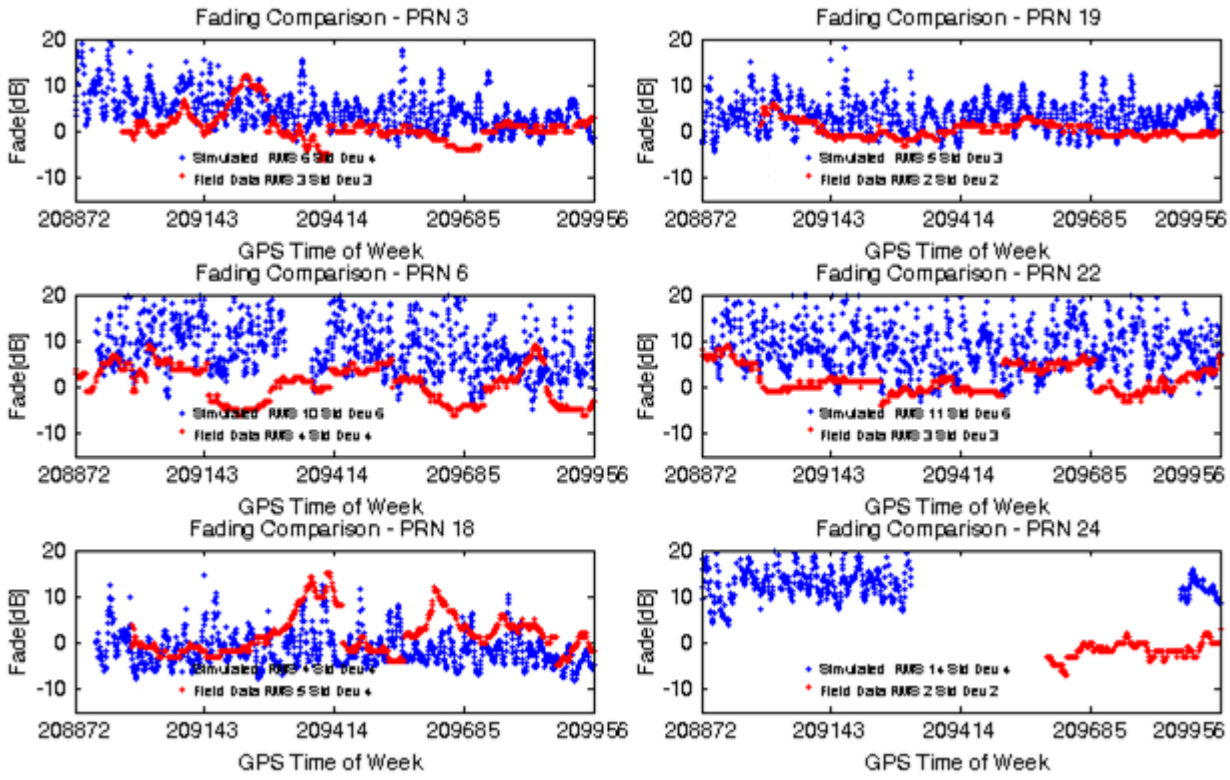

Figure 4-14 Fading comparison between simulation and field test

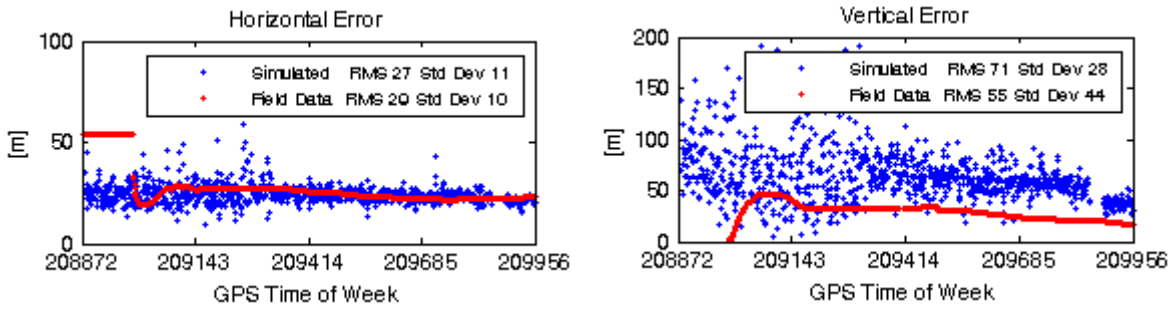

Figure 4-15 Horizontal and vertical error - comparison between simulation and field test

The comparison of the field test and simulation showed that the model chain provides fading and position error effects similar in magnitude to the field data. The deeper attenuation in case of satellites with high elevation shows the dependence of the model chain on a fitting building model. The fading data exhibited a good correspondence also in variation. The sinusoidal effect typical for multipath was reproduced. A comparison with [51] shows the strength of this attitude for simulation of signal in indoor environment where almost all satellites are influenced by multipath. Nevertheless the realistic frequency is an open topic. The evaluation of the positioning data was limited by the features of the used receiver. A test campaign with raw data output including pseudoranges followed by position computation with the means of the same algorithm for field and simulated data could deliver a closer look to the delay characteristics of the model chain.

5 GNSS Signal Simulator Architecture

5.1 Evolution of GNSS Signal Simulator Design

Fifteen years ago and earlier, GNSS simulators were built based on prevalently analog architecture [2]. Each signal channel was generated separately by modulating an analog IF carrier. The modulation data, i.e. PRN codes, navigation message and Doppler values were computed in the digital part of the simulator and given over to the analog part. This principle is depicted in Figure 5-1. For contemporary GNSS simulators, such architecture is not in use any more.

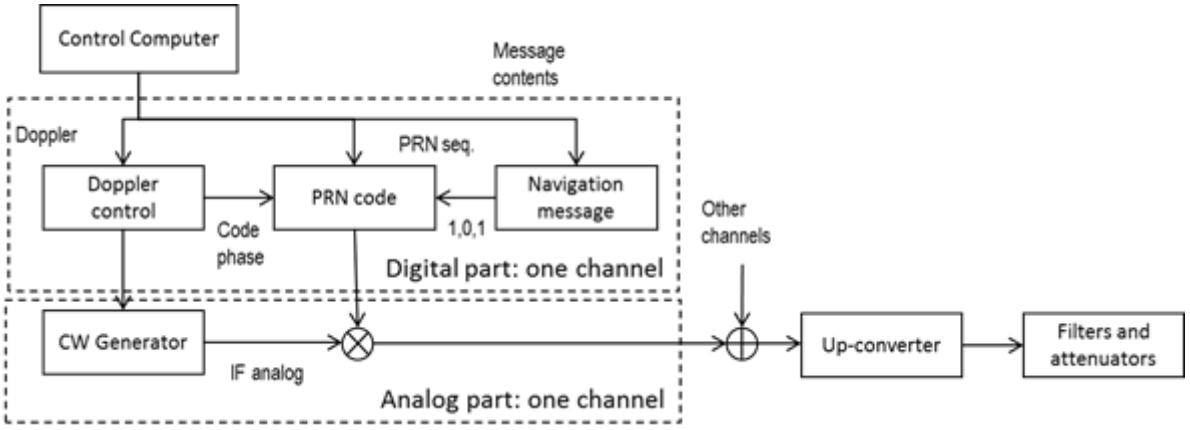

Figure 5-1 Architecture of an analog signal simulator

Nowadays, digital simulator architecture as sketched in Figure 5-2 is being used. The modulation data are generated in digital form similar to the earlier analog architecture. The difference is the deployment of the generation of the IF signal carrier and its modulation in digital form. Channels of the signal service are also digitally mixed and bandpass filtered. Thereafter, the digital signal is converted to analog form.

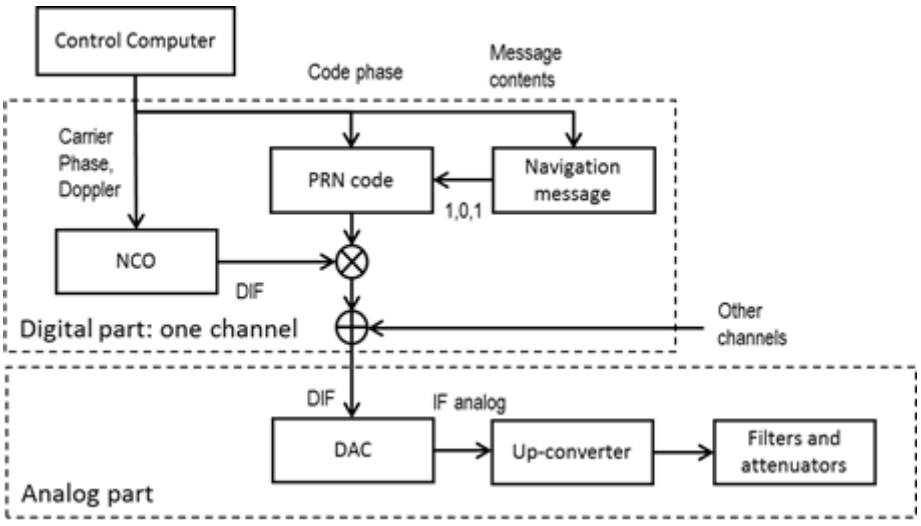

Figure 5-2 Architecture of a digital single-frequency signal simulator

Advantages of this approach in contrast to the analog architecture are high quality of signal, low performance loss due to aging, high predictability, high precision of frequency, high frequency resolution and no inter-channel or inter-frequency biases [2].

Disadvantages are the relatively small bandwidth, limited available relative signal power range among channels and spurious frequencies problem. The spurious free dynamic range (SFDR) has to conform to the SFDR published in the interface control document (ICD) of a GNSS system. For GPS L1, L2 and L5 signals up to generation III, the in-band spurious transmissions are defined to be at or below -40 dBc [47], [61] and [62]. For Galileo E1OS, this value not given in the current version of the ICD [63], it will be probably published in a later stage of the system deployment.

5.2 Digital Part of GNSS Signal Simulator

The functional structure of the digital part of a GNSS signal simulator consists of three basic modules module as depicted in Figure 5-3. The simulation definition part computes an epoch-based description of the user scenario; the signal definition module computes the signal parameters for each channel and the signal generation module generates the final digital signal. For comparison, an abstract simulator model with focus on simulator specification can be found in [6].

The first module takes user input in form of almanac, ephemerides, simulation start time and duration, receiver position, speed and trajectory together with additional settings of propagation channel. This module computes raw description for each epoch of the received signal as a set of pseudoranges, signal power, additional attenuation, delays and clock errors. Epoch length is set according to user dynamics to be simulated starting at the order of seconds and decreasing for flight or missile scenarios with high acceleration and jitter dynamics to a millisecond.

The second module calculates precise signal definition static over a short microepoch of signal. The length of the microepoch can be in order of milliseconds and must be even shorter for highly dynamic environments. The output of this module is a description of every signal channel with carrier phase and frequency, code phase and frequency, amplitude and PRN sequences. Description of all impairments and effects computed in simulation definition is finally expressed in terms of basic signal parameters – phase, frequency and amplitude of carrier and code. Multipath effect is an exception. The multipath signal reflections are generated as independent signal channels with their own phase, frequency and amplitude derived from the line-of-sight signal channel.

The third module performs signal generation. The digital carrier is generated at the given intermediate frequency. The baseband signal is computed and modulated onto the carrier. The individual channels are added together and further adjusted – filtered, upconverted, mixed with noise or attenuated.

The simulation definition module has the lowest computational load as it outputs data only for each epoch; signal definition module has higher load as microepochs are much shorter. The computational load of both modules varies according to implemented error models. The signal samples generation module carries out the highest workload due to MHz-order speed for PRN generation and MHz-to-GHz-order speed for final signal generation. To reach real-time performance, the key bottleneck is the speed of the signal generation module.

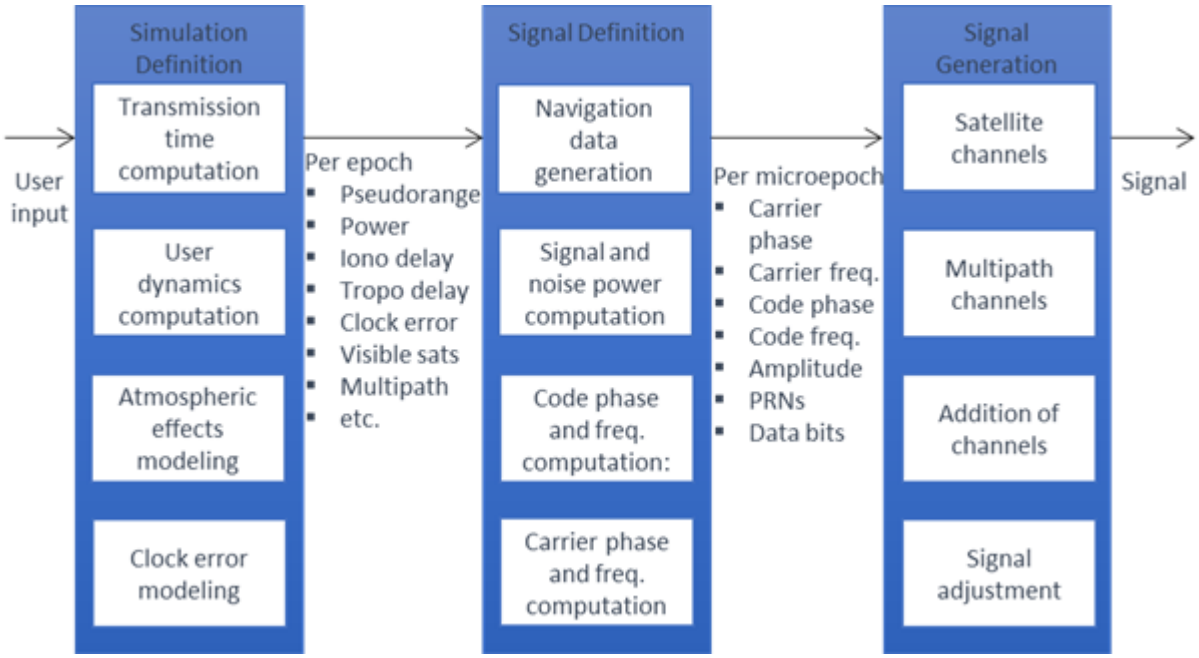

Figure 5-3 Functional structure of a digital part of a signal simulator

The features and limits of the performance are given by the chosen platform for deployment of the digital signal generation module. Although manufacturers do not publish the architecture and hardware their products are based on, it is well known, that field programmable gate arrays (FPGA) and digital signal processors (DSP) are used for digital signal generation. In experimental and non-real-time simulators, CPUs are used. The application-specific integrated circuits (ASIC) are the traditional choice in real-time software radio.

A dedicated ASIC circuitry would offer very low price per piece, low power consumption and high speed, but with the low number of simulators being sold, the costs for an ASIC design are too high. Therefore, an off-the-shelf processing engine are used.

Closest to a dedicated ASIC is a field-programmable gate array (FPGA). It is an integrated circuit designed for configuration by a customer or a manufacturer after production. Because of this feature, it is called field-programmable. The FPGA configuration is carried out with a hardware description language. An FPGA consists of logic cells connected to an array of programmable logic blocks. A system of reconfigurable interconnects allows the blocks to be connected according to a user-defined design. A typical logical cell consists of two 3-input look-up tables, full adder, flip-flop and interconnects to other cells. After filling of the lookup tables, setting operation mode and configuration of interconnects, any logical and arithmetical function can be implemented. This feature of FPGA can be simply proved by the fact that it is possible to configure an FPGA to act as a general microprocessor. An example of such a microprocessor software for FPGA is MicroBlaze from Xilinx [64]. Traditionally, FPGAs have been reserved for low-volume applications. The reason was that the higher price that companies pay in hardware costs per unit for a FPGA is more affordable than the development resources spent on design and production of a dedicated ASIC. Recently, the cost and performance dynamics of FPGAs developed an affordable low-cost segment. Development of ASICs faces growing costs of design. This situation have broadened the range of products, where applications of FPGA are viable.

The other engine dedicated for digital signal processing tasks and applicable in a simulator architecture is the digital signal processor (DSP). It is a specialized microprocessor with an architecture optimized for the digital signal processing. It succeeds over a general-purpose microprocessor for digital signal operation, because it generally tends to provide a lower-cost solution with better performance, lower latency, and no requirements for specialized cooling or large batteries. It is based on Harvard architecture or Modified von-Neumann architecture with separate program and data memories [42]. The architecture excludes memory management unit and virtual memory and memory protection, works with fixed-point arithmetic and use single-instruction multiple-data (SIMD) operations, defined according to Flynn's taxonomy [65], and multiply-accumulate instructions extensively. DSP is even more flexible than a FPGA, as it can be reprogrammed even with high-level programming language like C. The flexibility however comes in some cases at the cost of efficiency. Applications with several simple computations that could be performed in parallel may need to be broken into sets of sequential computations based on number of multiplier and accumulators on the DSP. Even though there is a set of parallel units like multiply-and-accumulate for the SIMD operation, this number is fixed and may not be optimal for the particular task to be performed.

Despite of the general architecture without focus on digital signal processing, general microprocessors (CPUs) are used as a signal generating engine for experimental DIF simulators [25], [45], [4], [26]. Also manufacturers of commercial non-real-time DIF simulators with replay functionality use CPUs, e.g. NavSys [19]. Deployment of CPUs for a real-time simulator is nevertheless limited by non-real-time operating systems and low performance for digital signal processing in contrast to their overall high computing performance, high price and high power consumption.

The development on the field of CPUs tends to add more and more previously DSP specific features to saturate the growing need for processing of multimedia content by a general computer [66]. Therefore, the focus of a CPU moves from traditional data manipulation to inclusion of focus on mathematical calculation specific to digital signal processing.

5.3 Multi-frequency GNSS Signal Simulator Architecture

The choice of an appropriate architecture is especially important when high performance is needed. As described in section 2.1.3, this is the case for the multi-frequency multi-service simulators that strive to be able to simulate the full scale of current and planned GNSS signal services simultaneously with realistic multipath. The overview of performance of recent market products is given in section 2.3 in Table 2-1 and Table 2-2. The scope of the current and planned GNSS signals and the indoor applications with heavy multipath pose nevertheless a considerable challenge on the architecture performance.

The current and planned GNSS signal services are transmitted in the reserved frequency bands placed in the frequencies designed for radio navigation satellite systems (RNSS) - in frequency range 960 – 1,300 MHz - and aeronautical radio navigation systems (ARNS) - in frequency range 1,559 – 1,610 MHz. An overview of the the GNSS frequency bands is given in Figure 5-4. The span of these frequency bands limits the deployment of common front ends in design of multi-frequency receivers and simulators. The synchronization of multiple front ends poses nevertheless a design challenge.

In the architecture of a conventional multi-frequency GNSS simulator, the frequency bands are generated separately. After generation of digital signal samples, the conversion of separate frequency

bands to analog signals follows. Thereafter, the final analog signals with different frequencies are mixed together. The concept is depicted in Figure 5-5.

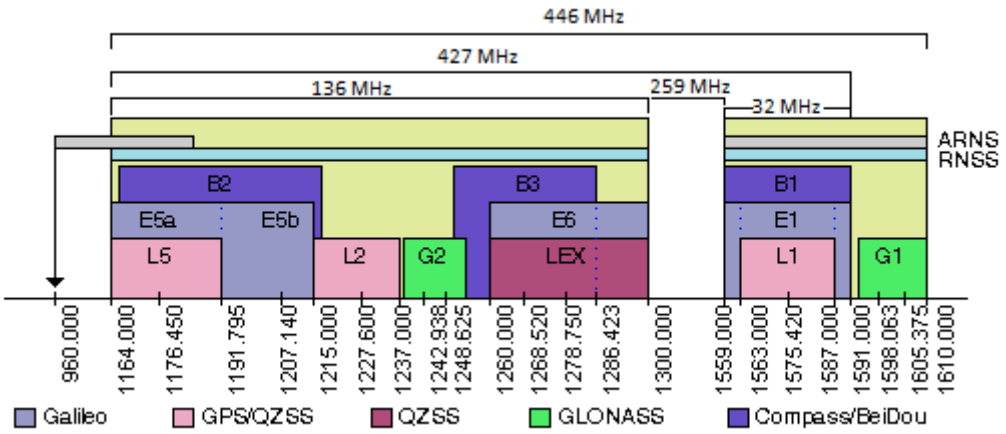

Figure 5-4 GNSS frequency plan for GPS, Galileo, QZSS, GLONASS and Compass/Beidou

The signal bands generated together in digital form comprise more services wherever the frequencies are close enough. The target is to reduce the number of separated signal generation and conversion paths. E1 and L1 are always generated together. E5a is combined with L5. In broadband designs, E5 is generated in full bandwidth and combined with L2. Moreover, the proximity of L1 with G1 and B3 with E6 makes their common generation possible as well.

The high costs of multifrequency multiservice simulators are partly caused by the costs of multiple high-performance digital signal generation engines (FPGAs or DSPs). Another factor is the costs of the hardware for synchronized upconversion and mixing of the separate signal bands. Therefore, the number of simultaneously generated carriers is an important feature of the simulator that has a great influence on the price of the simulator product, because of the costly hardware for the synchronized analog upconversion and mixing.

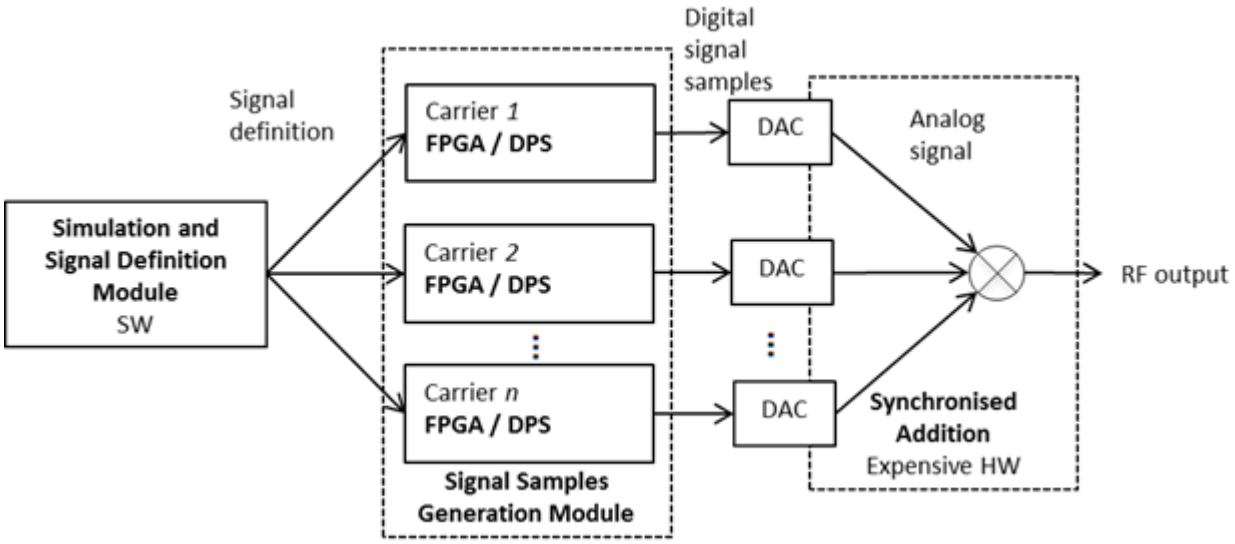

Figure 5-5 Architecture of a conventional multi-frequency GNSS signal simulator

5.4 GPU-Based Broadband GNSS Signal Simulator Architecture

This thesis introduces a concept of a broadband GNSS signal simulator that generates the full GNSS bandwidth of 446 MHz as one digital and after conversion analog signal band. It uses only one single digital-to-analog converter and one upconverter. The synchronized analog upconversion of separate bands and synchronized mixing can be omitted. The signal generation module must generate the digital signal with high sample rate. With Shannon-Nyquist theorem in mind, the sample rate must exceed 892 Msps.

It is not possible to reach such a high signal generation speed for multiple GNSS services and sufficient number of channels with current FPGAs, DSPs or CPUs. Also the outlook for the near future shows, that the development towards faster processor technology is about to reach its limits below 4 GHz of clock speed. Further development progresses in the direction of parallel multiprocessor technology.

A special processor segment with the focus on massively parallel processing are the graphic processing units (GPU). These parallel processors with hundreds of cores are very cheap thanks to mass-market production for video and gaming industry. A high-end graphic card for gaming with the newest GPU technology, the GTX Titan X, costs about 1,000 EUR. It can be connected in assemblies of two or four units in one computer. Besides the usage for computer graphic generation and processing, a GPU can also be used for general computer processing tasks.

This thesis proposes the usage of the computing power of a GPU for the Digital Signal Generation module of the broadband GNSS signal simulator. The concept of the architecture is depicted in Figure 5-6. The generation of the single broadband signal comprising multiple GNSS carriers is executed on a pair of GPUs with high sample rate. The high-bandwidth signal is then converted to analog using a single broadband DAC. The analog signal is then upconverted to the GNSS frequency band without any loss of synchronization among the individual signal bands. The objective is the development of algorithms and evaluation of the concept of this alternative broadband architecture of a real-time GNSS signal simulator.

In contrast to state-of-the-art FPGA and DSP based architectures, the nature of the GPU-based computation is massive parallel single-instruction, multiple data (SIMD) processing with limited shared memory resources and limited pipelining.

The Simulation Definition module and the Signal Definition module are realized using a CPU. The conventional architecture also computes the Simulation Definition by means of software on a control computer. This was the case for the old analog architecture (Figure 5-1), newer digital architecture (Figure 5-2) and remains like this also in case of contemporary multi-frequency multiservice simulator architecture (Figure 5-5). The low-speed parts of Signal Definition module are probably also computed in this way in most conventional architectures. The high-speed parts are probably implemented on the used signal generation engine. Nevertheless, the delimitation of the low-speed and high-speed tasks and other details about the architecture are a part of manufacturer's unpublished expertise.

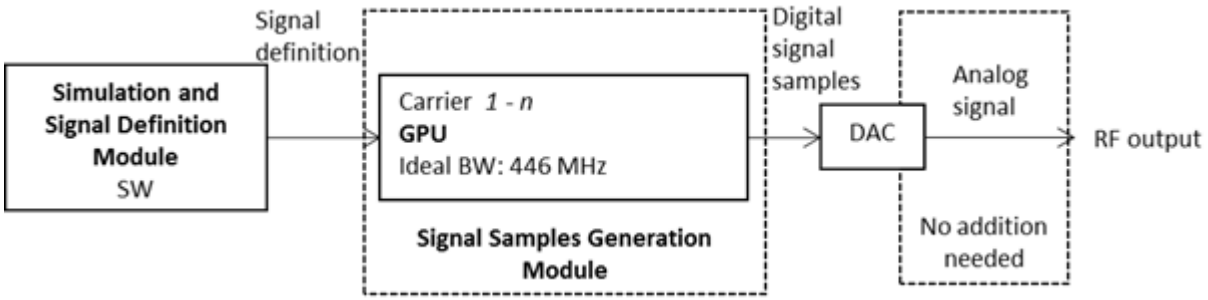

Figure 5-6 Broadband GPU-based GNSS signal simulator architecture

The concept of broadband GPU-based signal generation makes it possible to omit the costly analog parts for synchronized separate up-conversion and mixing of the separated analog signals. Moreover, such a digital signal generation architecture for the broadband GNSS signal offers higher scalability than the separated digital and analog engines for dedicated frequency bands. The signal generating computing power can be freely allocated to satellite signal channels and multipath signal channels of any GNSS service of choice. Another important degree of freedom in contrast to the conventional architecture is the fully free assignation of the central frequencies and frequency bands in the one common broadband signal.

For the realization of this architecture concept, the low-cost mass-market paradigm of GPU computing was paired with a low-cost computer system backbone. A gaming class PC-system with a gaming class motherboard and other components is used. The objective is to prove the low-cost aspirations of the whole simulator system.

The proposed architecture implies also several drawbacks. The current availability of high-quality broadband DAC circuitry is limited to handful of products. The DAC is needed in a bundle with a platform connected with high-speed interface to the computer system.

Another issue is the availability and costs of a high-quality broadband upconverter. The quality of these components must be taken into account and even compensated for in digital signal generation. The objectives of the development work in this thesis arise from the features of the architecture. They are handled in following chapter as follows:

Chapter 5 Precision Analysis of Digital GNSS Signal,

- Analysis of the quality of the generated digital signal and discussion of tradeoff between performance and quality.

Chapter 6 Digital Signal Generation

- Development of massive parallel algorithms for GNSS signal generation on a GPU
- Development of parallelized streaming concept of generated signal over gaming level PC to a DAC converter
- Achievement of real-time throughput of the signal with bandwidth of whole GNSS frequency span through whole system
- Optimization of the performance so as to achieve the real-time performance for as high number of GNSS signals and channels as possible

Chapter 7 Digital-to-Analog Conversion and Up-Conversion

- Research of market-ready solutions for broadband DAC conversion for gaming level PC-system
- Deployment of chosen DAC board and optimization of the performance
- Research of market-ready broadband up-conversion solutions

5.5 Test System

For the development, implementation and verification a test system was built up. The features of the chosen PC system, and its components inclusive GPU influenced the final version of the algorithms and solutions proposed in this work. The hardware and software parameters of the platform determine the optimal implementation strategy and determine the performance and precision of the signal generation. As an example, data types, library functions, memory management and data transfer procedures can serve.

The Test System is a high-end gaming PC depicted together with all components in Figure 5-7 on the left side. It is based on a top gaming-level mainboard ASUS Rampage IV. The mainboard accommodates two PCIe x16 v.3.0 slots, two PCIe x16 v. 3.0 slots with x8 performance and two PCIe x1 v. 3.0 slots. When all four x16 slots are utilized, the second PCIe x16 slots degrades its performance to PCIe x8.

The PC-system features Intel Core i7-4930K 3.4 GHz processor with six cores built using 22-nm technology with 1.86×10^9 transistors. It comprises 256 KB L2 cache per core and 12 MB L3 cache. In the PC-system, the memory Corsair Dominator Platinum Series DDR3-2133 CL9 with 32 GB size and peak transfer rate of 17,066 MB/s is placed. The system runs Windows 7 with Microsoft Visual C++ 2012. Two graphic cards Nvidia GeForce GTX TITAN Black are included in the system. Detailed features of the GPU are given in the next section. The summary of the key components of the Test System is given in Table 5-1.

Component	Product
Mainboard	ASUS Rampage IV Black Edition, Intel X79 Mainboard, RoG - Socket 2011
Memory	Corsair Dominator Platinum Series DDR3-2133, CL9 – 32 GB
Processor	Intel Core i7-4930K 3.4 GHz (Ivy Bridge E) Socket 2011
GPU	2 × GeForce GTX Titan Black, 6,144 MB DDR5

Table 5-1 Test system components

Beside the standard PC components, a DAC system compatible with one of the mainboard output interfaces is needed. This interface must offer a very high data transfer speed close to the memory transfer speed of 17,066 Msps. The generated digital signal samples must be transferred through this interface with a speed exceeding the given sample rate to reach real-time operation. The minimum sample rate to generate the whole GNSS bandwidth was defined as 892 Msps in previous section. In practice, the sample rate must be higher, ranging from the maximum represented frequency multiplied by 2.5 and 4.5 as explained in section 3.1.3. Therefore, throughput of at least 1.115 Gsps is needed. With common resolution of 2 bytes per sample, this results in need for throughput exceeding 2.230 GB/s. With respect to need for high quality signal and features of available DACs, the sample rate of 1.400 Gsps and respective data throughput of 2,800 MB/s was chosen for implementation.

The PCIe v. 3.0 x16 interface with throughput of 15,754 MB/s offers enough capacity and therefore it is a clear candidate for the digital signal samples transfer connection. The alternatives of USB v. 3.0 (625 MB/s) or SATA v. 3.0 (675 MB/s) lie far below this performance.

Availability of a broadband digital-to-analog converter for the PCIe interface was examined. For the verification of individual signal service bands in low sample rate modus, the narrow band PCIe DAC converter ICS1572A from General Electrics was deployed. For broadband signal generation, a high-performance PCIe DAC converter FMC230 PC820 from the company 4DSP was purchased. The details are given in Chapter 7.

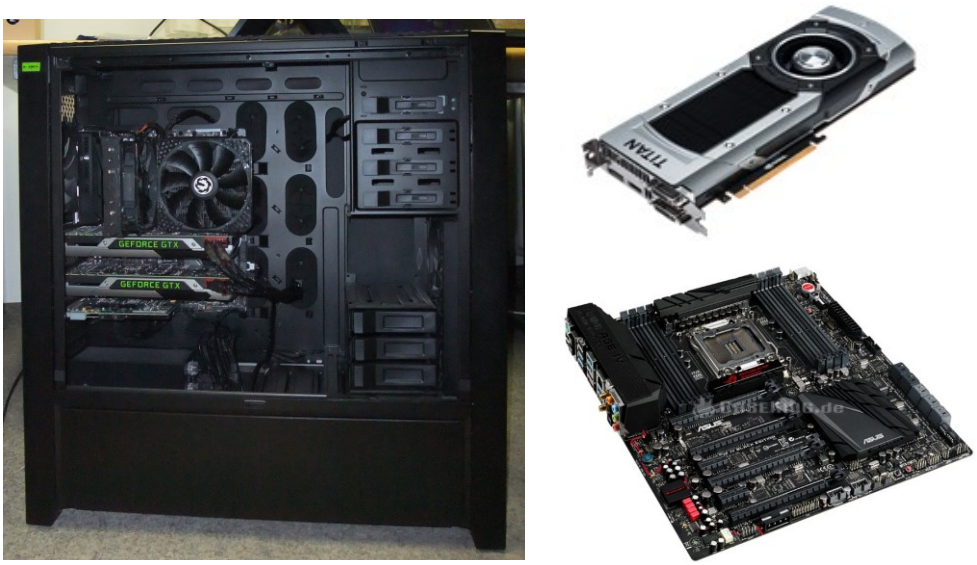

Figure 5-7 Test System with 2x GPU - left, GeForce GTX TITAN Black - top right, mainboard ASUS Rampage IV Black Edition – bottom right

For evaluation of real-time ability, data throughput and signal generation performance, the particular configuration settings of the GNSS signal simulator must be defined. Two configurations are considered in this thesis:

- Single Service configuration: 22.856 Msps sample rate (SR), 5.115 MHz IF, GPS L1 C/A, 12 satellite channels
- Broadband configuration: 1,400 Msps SR, 427 MHz BW, civil signals for GPS and Galileo: L1 C/A for GPS and E1 OS and E5ab for Galileo, 12 satellite channels per service

The Single Service configuration represents the minimum signal simulator configuration. The simplest conventional simulators described in Chapter 2 GNSS Signal Simulators feature similar abilities. The Broadband configuration represents the maximum performance in terms of bandwidth and sample rate. The objective is to include generation of as many GNSS services with all satellite channels in view and multipath channels as possible.

5.6 GPU Architecture and CUDA Interface

The GPU GeForce GTX Titan Black by Nvidia depicted in Figure 5-7 top right was chosen for deployment. The GPU chip was manufactured with 28 nm process and thus offering good power

efficiency. The internal architecture of the GPU is depicted in Figure 4-8. The features relevant for this work are listed as follows:

- Architecture Kepler GK110 with CUDA computing capability (CC) 3.5
- 6,144 MB DDR5 GPU memory with 336 GB/s (called device memory in Nvidia documentation) – shared among all multiprocessors
- 1,536 KB of L2 cache
- PCIe v. 3.0 x16 interface to host computer
- 15 multiprocessors (Kepler GK 110 multiprocessor is called SMX)

The internal structure of the multiprocessor SMX is depicted in Figure 5-9. The features relevant for this work are listed as follows:

- 192 single precision processing cores performing integer and floating point operations
- 8 double precision units on GeForce graphic cards [69], 64 on Tesla and Quadro (Figure 5-9)
- 32 special function units (SFU) – goniometric, logarithmic and inverse functions performed in a single clock cycle
- 32 load/store units
- 48 KB of shared memory – shared across all cores of one SMX
- 16 KB of L1 cache
- 65,536 of 4B-registers (with 1,024 in one block of threads: max. 64 registers per thread)
- GPU clock speed of 889 MHz

Figure 5-8 Architecture of Nvidia GPU GeForce GTX Titan Black with 15 SMXs, shared L2 cache and six memory controllers [67]

Full summary of the features of the GPU can be found in the documentation of the manufacturer. Technical details about the architecture are given in the whitepaper Kepler GK110 [67].

The registers are the fastest memory of the GPU. Each register is assigned to a single thread only. All threads operating on one multiprocessor share access to the so-called shared memory. A warp of threads can fetch data in just one clock cycle. The size of 48 KB is nevertheless very limited. All multiprocessors share the main GPU memory with sufficient capacity of 6 GB, but the access speed is much lower than in case of the shared memory. Around 400 - 800 clock cycles are needed to start the data transfer from the GPU memory. Additionally, the SMX features the texture memory with 48 KB, which is read-only for the duration of one kernel run. Access speed is similar to shared memory.

Figure 5-9 Architecture of a multiprocessor SMX from Kepler GK110 C.C. 3.5 series with 192 single-precision cores, 64 double-precision units (DP Unit), 32 special function units (SFU), and 32 load/store units (LD/ST) [67]

For the programming of the GPU a special library with multithreading access pattern is needed. There are general programming interfaces for parallel programming, for example OpenCL [68]. A program based on such a general interface can exploit just a fraction of the performance that could be reached by an architecture specific library. For the analogic task of GNSS signal acquisition and tracking, this observation was published in [32]. The GPU manufacturer Nvidia developed for the general purpose programming of GPUs the Computing Unified Device Architecture (CUDA) interface. In this work,

the CUDA C extension of C/C++ was used for programming of the GPU. The first algorithms were tested using CUDA version 2.1. The latest were benchmarked with CUDA version 6.0 described in detail in [69].

CUDA introduces a special threading concept for easy parallel programming of Nvidia GPUs with hundreds of cores. CUDA C extends standard C by allowing the programmer to define C functions, called kernels. These, when called, are executed *N* times in parallel by *N* different CUDA threads, as opposed to only once in regular C functions. The CUDA threading concept outlined in Figure 5-10 suggests deployment of many threads, in the order of thousands. The threads are organized in multiple blocks of threads in a so-called grid. There should be several hundreds of threads placed in one block. The threads are sorted in groups of 32 threads, so called warps. Each of the blocks of threads is executed by one multiprocessor of the GPU. Threads in each warp are run in parallel on the multiprocessor cores in SIMD pattern. For multiple threads, a single instruction is scheduled to multiple cores of the multiprocessor simultaneously.

Each SMX uses four instruction schedulers (so called warp scheduler, see Figure 5-9). The instruction scheduler issues one instruction for whole warp of threads (ea. 32) each two clock cycles. In the first clock cycle, the instruction is executed in parallel for the first half of the warp (ea. 16).

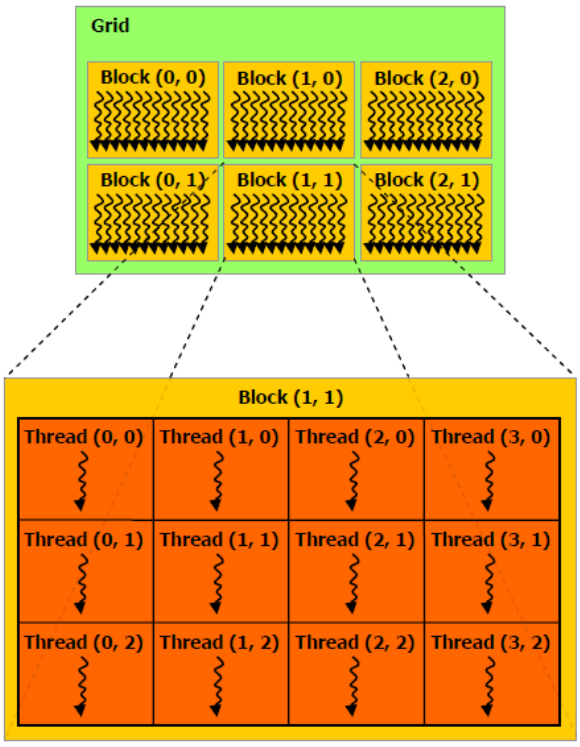

Figure 5-10 CUDA threading concept [69]

In the second cycle, the second half of the warp is executed. The instruction can be scheduled either for 16 processing cores or for 16 load/store units (LD/ST in Figure 5-9) or for 8 special function units (SFU in Figure 5-9). With four schedulers, always four groups of processor units are occupied at a time (e.g. 16 single precision sores, 8 SFUs, 16 double units and 16 single precision cores). One multiprocessor can hold up to 1,024 threads at once, computing always the four warps in parallel that are ready for computation. It is convenient to use high number of threads to cover the waiting time for loading data from GPU main memory. Nevertheless, the limiting factor for the number of threads

residing on a multiprocessor is the number of registers. Each multiprocessor features 64 × 1,024 of 4 B-registers, thus with maximal number of threads per block being 1,024, each thread can use only up to 64 registers without employment of the much slower GPU memory.

Each multiprocessor includes 48 KB of fast shared memory accessible from all residing threads. The shared memory is organized to 32 banks, which can be accessed by 32 threads in parallel. The pattern how the threads access the shared memory is very important for high throughput of the parallelized program. It is an essential criterion of a good algorithm for GPU to introduce an effective shared memory access concept that uses the parallel access to the 32 banks as much as possible and that avoids bank conflicts.

The GPU memory and L2 cache is shared by all multiprocessors and it is accessed through multiprocessors' own L1 caches (16 KB). The access to the main memory is slow; it takes 400-800 clock cycles. To speed up the access, the GPU memory accesses should be coalesced to chunks of 32 following 4B-numbers. To use up the computing power of the cores of the GPU, the memory access must be carefully designed with architecture and memory hierarchy in mind.

6 Precision Analysis of Digital GNSS Signal

The quality of the output signal of a GNSS signal simulator is a key performance parameter. Design of the simulator has to fulfill the defined requirements for signal quality and real-time performance for required ranges of settings. A common basis for definition of the quality of the signal produced by a simulator was proposed by Tetewski in [6].

The relationship between the parameters of the design of both digital and analog parts of the signal simulator and the quality of the output signal is not straightforward. It is an important task in development stage of a GNSS signal simulator to analyze this relationship so as not only to reach the desired output quality, but outweigh the performance and costs of the components as well.

This chapter focuses on the quality of the generated digital signal. The digital signal quality results from numerical precision of key variables of the signal generation process. This numerical precision can be expressed as the number of bits and the numerical format of the variable. The speed of digital signal generation is influenced by the numerical precision influences to a great extent. The task of specification of numerical precision is demanding as the influence on speed and precision oppose each other. This effect is easily noticeable in case of the representation of a signal sample: high bits-per-sample resolution lowers the sample-per-second throughput through a given bandwidth of a digital interface and vice versa.

In this chapter, the key variables of the signal generation process are described in detail and their influence on the signal quality and generation speed is evaluated. For each variable, the parameters influencing its range of values are listed and the equation for the numerical precision of the parameter is given. The application on the Test System simulator architecture is presented and the numerical precision for its settings is listed together with analysis and measurements of the influence on the quality of the generated signal.

6.1 Test System Configuration

The key variables of the signal generation are to some extent conform to the used hardware. The reason is that the scope of available data types, core functions and their speed and precision are dependent on the hardware parameters of the platform. The Test System as defined in section 4.5 is considered for this analysis. The Single Service configuration and Broadband configuration are applicable as defined. Additionally, L1 Services configuration is defined as follows:

- L1 Services configuration: Single frequency L1-based GNSS simulator with services GPS L1 C/A, P (Y), L1C and Galileo E1 OS and 12 satellite channels per service

The native and broadly supported data type of the GPU in the Test System is single-precision (32 bits) floating-point number (abbreviated as float further on). The float is also the native data type on the older GPUs by Nvidia and GPUs by Nvidia's competitor AMD. GPUs are designed for fast processing of raster graphic. The human perception sensitivity for information about each visual point - color and brightness - is mostly exponential with limited resolution, similar to a shorter exponentially represented data type. On the other hand, the human sight features very high sharpness and needs high refresh rate of visual input to perceive a movement as continuous. As a result, high number of points and high data throughput are needed, which makes shorter data types advantageous.

The float-based arithmetic outperforms on GPU integer based operations. In contrast to it, CPUs are optimized for integer operations and DPS for fixed point arithmetic. The conventional implementation for digital signal generation is fixed point integer based arithmetic. Therefore, two implementations are considered and compared in the following sections – the float-based and the fixed-point 32-bit integer based solutions (shortcut as "fixed point" in this text).

The CPU and GPU arithmetic is restricted to a handful of data types that can be effectively input in execution of all instructions. The other data types need to be converted to designed data types for specific instructions. The chosen GPU architecture can operate on signed and unsigned integer with 83% performance compared to float and on double (64 bit) floating point numbers and long (64 bit) integers with 33% performance. Additionally, the 64-bit data types consume twice as much of the very limited shared memory. Operations on longer data types need to be implemented by multiple operations using 64-bit variables. The 8-bit and 24-bit variables can be addressed, but need to be converted to 32 bits before any operation can be executed. The overview of performance of individual instructions on supported data types is given in [69].

Three signal simulator variables play key role in the performance. With respect to section 4.5, these are time from the beginning of the simulation in the Signal Definition module, the variable holding each signal sample in the Digital Signal Generation module and the variables holding NCO parameters in Signal Samples Generation module.

6.2 Time from the Beginning of the Simulation

Each epoch of the signal is defined by its precise time from a demarcated starting point. This starting point might be e.g. the beginning of the simulation or the start of the actual GPS week. Professional GPS receivers use typically the day of week and additional variable for week number and day of week, or split the time into an integer part and a decimal part [45].

The time epoch variable in the simulator follows similar patterns. It is used to compute the entire signal parameters and signal level for each epoch and its precision influences the precision of all dependent computations. In order to simulate the point position with precision in order of the millimeter we have to know time with the precision of 0.01 nanosecond. The precision of the variables in terms of number of bits must therefore be carefully chosen.

The precision of positioning in terms of meters translates into signal precision in terms of signal phase. The relationship between both depends on the carrier frequency used for the signal generation. In the digital part of a simulator, the maximal carrier frequency is the designed intermediate frequency (IF) plus the maximal Doppler to be simulated. The time epoch expressed as carrier phase comprises the number of full cycles and decimal part. The number of bits N_t needed for representation of time epoch expressed in cycles is then computed as

$$N_t = \lceil log_2(tf) \rceil + N_\varphi \tag{6-1}$$

where t is time from the beginning of the simulation, f is carrier frequency with maximal Doppler and N_φ are designed bits for carrier phase resolution.

The simulation duration in a simulator system can be just several seconds for tests of tracking and acquisition. For tests of time-to-first-fix, several minutes are needed. Duration of several hours might

be necessary for positioning tests and a week serves as a reasonable maximum for very long tests, e.g. for signal monitoring scenarios.

The intermediate frequency, in this text referred to as IF, is specific to every simulation system. The IF of 4.092 MHz is the very minimum, sufficient for single service L1 C/A scenarios. Maximum would be direct application of RF frequency with full bandwidth, if ideal DACs and filters were available. Figure 6-1 illustrates the relationship between simulation duration and variable resolution for these carrier frequencies.

As the precision limit for carrier phase, the thermal noise of a phase lock loop (PLL) of target receiver design was considered for the figure. For the Test System specification the maximal signal-to-noise ratio (C/N_0) reaches 68 dB/Hz in the test receiver, which according to [46] results in 0.0002 cycle for given settings. The digital representation with 13 bit offers even 0.000122 cycle precision.

From the figure it is clear, that to represent 1 hour, 47 bits are total minimum, and for 1 week at least 55 bits are necessary. In case of CPU and GPU based systems, the 32-bit formats (float, fixed point unsigned integer) are therefore insufficient for most of simulator scenarios. On the other hand, the 64-bit format (unsigned integer) would be enough even for one week of samples with RF frequency.

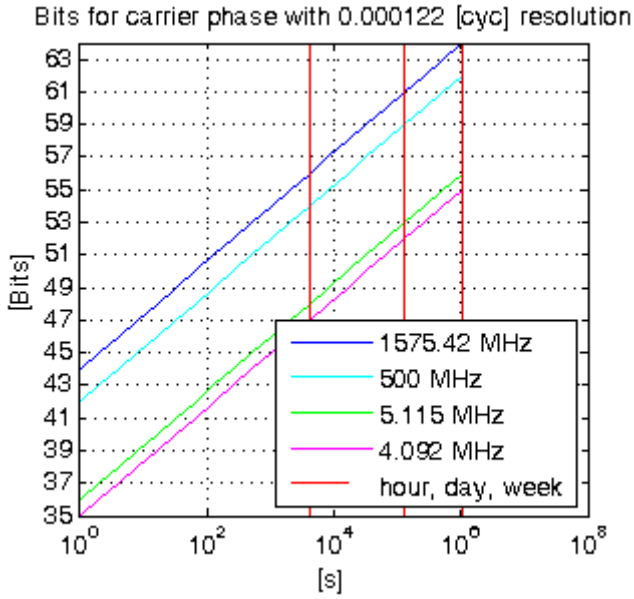

Figure 6-1 Resolution of time epoch with respect to simulation duration

The Test System simulator considered in this thesis defines the time epoch as time from the beginning of the simulation and limits the possible simulation duration. The chosen resolution is 54 bits (the number of resolution carrying bits from the 64 total) with double floating-point data type. For the broadband scenario with the carrier phase resolution of 13 bits, the chosen double data type limits the simulation duration to 36 minutes only. Unsigned long integer (with 64 bits) is for longer scenarios necessary; up to 625 hours could be represented.

6.3 Signal Samples

The next key variable of the signal generation process is the variable holding the generated digital signal sample. A GNSS receiver uses usually a low number of bits (2 - 4) to represent the received

signal buried in noise. In a GNSS simulator, the required number of bits has to account for the amplitude resolution of individual signal channels (one channel is used per satellite and frequency band), relative signal power of the channels and the total number of channels represented in the digital signal.

The power levels of the simulated signal samples can vary largely. Especially for testing of highly sensitive receivers, the ideal simulator should generate signal channels with weakest power that can be tracked together with strong LOS signal channels or even with pseudolite signal channels.

In a signal simulator, a signal channel is generated with digital signal synthesis technique using a numerically controlled oscillator (NCO) technique. The digital signal synthesis is described in Chapter 7. Considering the schema of an NCO given in Figure 3-5, the output signal sample of individual signal channels has amplitude resolution given by the phase-to-amplitude converter output resolution with *P* bits. The number of satellite channels added to one sample stream influences the number of bits needed for each sample in a straightforward manner. Each duplication of the number of satellite channels adds one bit needed to hold the full resolution. The bit resolution for representation of a certain number of signal channels can be calculated as

$$2^{Y-1} < x \leq 2^{Y}, \tag{6-2}$$

where x is the number of channels and Y is the resolution in bits. The next factor influencing the required number of bits is the relative amplitude among the signal channels. If the noise is included in the sample stream, as it is the case in DIF simulator system, the ratio between signal and noise amplitude needs to be considered as well. With power difference between two channels being z dB, D bits are needed in the sample variable to account for the relative signal power. Number of bits D is given by

$$2^{D-1} < 10^{\frac{z}{20}} \leq 2^{D}. \tag{6-3}$$

The number of bits N_S required to hold the full resolution of the final sample can then be computed as

$$N_S = Y + D + P, \tag{6-4}$$

where Y is number of bits to represent the number of channels, D is number of bits to represent the relative channel power and P is number of bits to hold the amplitude of each channel even at lowest possible signal power level.

In case of GNSS signals, the PAC of an NCO is an implementation of sine or cosine function. Typically, the phase-to-amplitude converter is implemented as a lookup table. The Nvidia GPUs nevertheless include multiple special function units (SFUs), which compute goniometric functions of float numbers in a single processor clock cycle. CUDA accesses these units with *__cosf* and *__sinf* functions, which deliver output resolution of 21.19 bits [70]. The number of SFUs is only one sixth of the number of float processing cores placed on a GPU, therefore also an implementation using look up table could be considered. A lookup table needs $2^M \times P$ bits of memory space and very fast access. The fast memory of the GPU consists of registers (64 per thread) and shared memory with only 48 KB needed for generated samples and PRN sequences. Therefore, the usage of SFU is the better solution offering additionally very high precision in contrast to lookup tables with typically 256 items.

In contrast to amplitude resolution in DDS systems with lookup table, the PAC resolution of 21.19 is very high. Nevertheless, limited number of bits used to represent the amplitude of the synthesized signal is usually not the major source of the errors in a DDS system [42]. This resolution is therefore reduced to 13 bits in this work. Table 6-1 enumerates the typical number of channels for the categories of Test System configurations and gives bit resolution needed for a sample variable.

Test System configuration	**# Channels**	**Bits (*Y*)**
Single Service: L1 C/A	12	4
L1 Services: Single frequency, multiple services: L1 C/A, P(Y), L1C, E1OS	72 (12, 12, 24, 24)	7
Broadband: civil GPS and Galileo on L1, L2 and L5	168 (12 × (5 + 2 + 6))	8

Table 6-1 Bit resolution accounting for the number of signal channels

Maximum received power of a satellite signal on the Earth surface is defined by the ICD of the respective GNSS service. Minimum power is given by the sensitivity of the "best" receiver to be tested by the simulator. For DIF signal generation, the noise needs to be included in the samples. The noise power is the strongest component to be considered in this case. For analog signal generation, the noise can easily be added later. Hence, it does not need to be considered in digital signal generation. Table 6-2 enumerates three DIF and three RF scenarios with the minimum and maximum power of the signal. For DIF scenarios, the noise floor for bandwidth of 2 MHz is considered as maximum.

Scenario	P_{min} **[dBW]**	P_{max} **[dBW]**	**dB diff**	**Bits (*D*)**
DIF, max.: extremely high sensitivity	-205	-140 (noise)	65	11
DIF, commercial chip sensitivity	-190	-140	50	9
DIF, ICD - maximum	-160	-140	20	4
RF, extremely high sensitivity	-205	-150 (ICD)	55	9
RF, commercial chip sensitivity	-190	-150	40	7
RF, ICD - maximum	-160	-150	10	2

Table 6-2 Bit resolution accounting for the relative power of signal channels

For final resolution of the sample variable, the designed number of bits accounting for the power difference, number of channels and carrier phase resolution need to be added as given in (6-4). An ideal DIF simulator would generate noise together with the weakest signal that can be tracked. It would be able to generate all signal channel in view from all existing and planned services together with heavy multipath. The sample amplitude would be used without reduction even for the weakest signal. Such an ideal GPU-based DIF simulator would therefore allocate 11 bits for power difference, 9 bits for channel number and 22 for PAC output resolution. This would result in a variable with 42 bits that would fit a double-precision integer or double-precision floating-point data type.

To use the GPU native data type float, the resolution must be reduced. When RF scenario is considered, the 24 bits of mantisa can be split among relative power, number of channels and carrier resolution as 5 + 8 + 11 = 24. For Single Service scenario in DIF with 12 channels, the float data type is sufficient. The bits are allocated as 7 + 4 + 12 = 23, so that the noise, 12 channels and reasonable carrier resolution can be represented. In the Test System simulator, the signal sample can be

configured as float or double data type. Test measurements for Single Service scenario with float and double samples are presented in the final section of this chapter.

On the Test System, the signal generation with double samples in much slower than with float. The double data type based operations have much lower throughput than float based. The shared memory can hold just half of the number of samples. The performance of generation of samples with double precision was measured to be just about 30% of the performance of the solution with float samples.

6.4 NCO Variables

In the Signal Definition module of the simulator as defined in section 4.1, for each microepoch of the signal to be generated, the starting code phase, carrier phase and frequency are calculated. In the Signal Generation module, these values are used to initiate the NCOs in the signal synthesis process. In this work, one NCO is applied for carrier wave and one for PRN code generation of each signal channel. Numerical precision of the NCO variables influences the precision of the generated samples and has strong impact on the overall speed of the signal generation. Careful choice of numerical precision is therefore necessary. The design of format and precision of NCO variables is influenced by the magnitude of and relationship between IF, SR, f_c, PRN code length and the minimum resolution of the Doppler frequency.

Notation of NCO parameters as defined in Figure 3-5 is used in this chapter, so that N denotes number of bits designed for phase increment and accumulator, M denotes bit resolution of the PAC input and P denotes bit resolution of the PAC output. Furthermore, C denotes the code phase and φ denotes the carrier phase. Code phase and carrier phase increment of two successively generated samples are referred to as ΔC and $\Delta\varphi$ respectively.

In a general NCO with fixed point variables, the relationship between the number of bits for frequency control word and resolution (i.e. minimal step) of output frequency depends on sample rate. The resolution of the output frequency Δf is then given by

$$\Delta f = \frac{f_{SR}}{2^N}, \tag{6-5}$$

where N is number of bits of the frequency control word, Δ_f is resolution of output frequency and f_{SR} is the sample rate. The desired frequency resolution for carrier NCO is equal to the desired Doppler frequency resolution of the simulator. It is designed according to desired dynamics range of the simulator capabilities. The frequency resolution is generally expected to reach at least the sub-Hz level.

Two NCO designs are considered and compared in this work – the fixed-point 32-bit integer based solutions (shortcut as “fixed point” in this text) and the float-based solution. The fixed-point NCO is a direct implementation of the conventional NCO model. The carrier NCO uses carrier phase increment and accumulator in unsigned integer format with 32 significant bits. All 32 bits serve for storage of decimal part of the fixed-point number. The code NCO uses code phase increment and accumulator in unsigned long integer with 64 significant bits. 16 bits hold the integer number of chips and 48 bits hold the code phase.

The signal generation algorithm using the NCOs was designed for operation of the GPU. The detailed description of the algorithm for the specific GNSS signal services is given in Chapter 7 Digital Signal

Generation. The following short summary depicts the features relevant for NCO precision analysis. The GPU signal generation algorithm employs dozens of threads. Conventional digital signal synthesis uses a single NCO to compute successive samples of a signal waveform in serial. The number of threads employed in the GPU based algorithm is $x = n \times 32$ where $n \in [1, 32]$. Factor n is adjusted according to the overall load of the computation. Each thread features its own NCO. The carrier phase increment of the NCO in one thread is given by

$$\Delta\varphi_i = x\Delta\varphi, \tag{6-6}$$

Where x is number of threads applied for the signal channel, $\Delta\varphi_i$ is the carrier phase increment of thread i and $\Delta\varphi$ is the carrier phase increment of NCO when operating in serial.

The PRN spreading codes are stored in the shared memory of the GPU multiprocessor. The size of the memory is limited to 48 KB. Computed samples occupy 40 KB. In the remaining 8 KB, only short parts of the PRN sequences of up to 12 satellites can be stored. The size of these parts is set to 64 chips. After generation of all respective samples, the new parts of PRN codes are loaded from the GPU memory.

The carrier NCO uses the full length of the 32-bit variable for the decimal part of the phase expressed in cycles. The automatic overflow is exploited as modulo 1 operation to keep the phase in cycles < 1. The code phase and message bit phase are hold by other variables; therefore the information about full number of cycles can be discarded.

This setting of the carrier NCO results in resolution of the carrier phase $\Delta f = 0.0053$ Hz for the Single Service configuration with SR = 22.856 Msps. For the Broadband configuration with SR = 1 Gsps, the resolution of the carrier phase $\Delta f = 0.23$ Hz.

The fixed-point code NCO is implemented as 64-bit NCO with 16 integer bits and 48 fractional bits. A good quality simulator should keep full control of the ratio between code and carrier frequency. The frequency set in the carrier NCO should be therefore precisely represented in the code NCO. The ratio between code and carrier frequency is varies among GNSS services. The value of 1,540 for L1 C/A signal with high carrier rate and low code rate is considered in this work as upper limit. This results in the need for additional 11 bits for the code NCO resolution and sums up to number of fractional bits > 43.

This setting of the code NCO results in resolution of the code phase of 8.12×10^{-8} chips/s for the Single Service configuration. Assuming chip length of 293 m (L1 C/A), it corresponds to range-rate resolution of 2.3×10^{-5} m/s. For the Broadband configuration with SR = 1 Gsps, the resolution of the code phase is 3.55×10^{-6} chips/s resulting in 0.001 m/s range-rate resolution for L1 C/A service. The summary of the design of the NCO variables for algorithm with 64 threads is given in Table 6-3.

The maximum value of each NCO variable is listed in third column of the table. In a serial algorithm, the maximum value of $\Delta\varphi$ would be limited by SR/IF ratio given by Nyquist theorem to values < 0.5. With multiple threads, $\Delta\varphi \in (0, 1)$. The maximum value of $\Delta C = 2.8645$ is given by 64 threads setting and L1 C/A chip rate relation to the sample rate given by the Single Service configuration. The maximum value of the code NCO accumulator do not cross the maximum of 64 chips, because the algorithm stops the generation procedure before the end of the last chip of the PRN part stored in the shared memory.

NCO variable	Unit	Maximum value	Integer bits	Fractional bits	Δ_f – Single Service [Hz]	Δ_f – Broad-band [Hz]
$\Delta\varphi$	cycle	0.9999	0	32	0.0053	0.23
φ	cycle	0.9999	0	32	0.0053	0.23
ΔC	chip	2.8645	16	48	8.12×10^{-8}	3.55×10^{-6}
C	chip	63.9999	16	48	8.12×10^{-8}	3.55×10^{-6}

Table 6-3 Code and carrier NCO in fixed-point algorithm with 64 threads per satellite channel

In float-based NCO design, the carrier NCO phase increment and accumulator are implemented as single-precision floating-point variables with 24 significant bits. The accumulation operation must be supplemented by truncation of integer part to limit the loss of resolution of the fractional part to 1 bit only. In case of high SR to IF ratio, the float-based carrier phase increment is a small decimal number with multiple leading zeros. In the float format, exponent moves the window of mantisa to low fractional bits of the number. With fewer bits, higher resolution can be reached, as leading zeros are omitted. The precision of float-based accumulator oscillates between the resolution of the carrier phase increment and the resolution of the mantisa. The decrease in resolution down to mantisa bits occurs at the moment when integer part of the accumulator value is > 0 and modulo operation follows. The equation of minimal frequency step is not as straightforward as for the fixed-point NCO given by equation (6-5). The frequency step depends on the magnitude of the value in the accumulator. The maximum value of carrier phase increment and accumulator value are summarized in Table 6-4 for the Single Service scenario run with 64 threads. The carrier increment for step of 64 threads is calculated in double precision and converted to float after modulo 1 operation. The truncated integer number of cycles in marked in the table by brackets. The frequency step of the float-based carrier NCO oscillates between 0.68 Hz and 2.72 Hz. The maximum value of code phase increment and accumulator value are given in the lower part of Table 6-4 with respect to the Single Service scenario run with 64 threads. The frequency step of the float-based code NCO oscillates between 5.45 Hz and 87 Hz. This resolution of the NCO is insufficient for the needs of precise frequency representation and code/ carrier skew.

NCO Variable	Unit	Max. value	Integer bits	Fractional bits	Δ_f
Δ_φ	cycle	(14).3255	0	24	0.68 Hz
φ	cycle	1.3254	1	23	2.72 Hz
ΔC	chip	2.8645	2	22	5.45 Hz
C	chip	66.8635	7	17	87 Hz

Table 6-4 Float-based code and carrier NCO in Single Service scenario with 64 threads per channel

For comparison, an implementation using double precision floating-point based NCO for code and carrier was evaluated. The frequency step values for Single Service scenario with 64 threads are listed in Table 6-5.

The frequency step of the accumulator of the carrier NCO oscillates between 1.27×10^{-9} Hz and 4.06×10^{-8} Hz. The frequency step of the accumulator of the code NCO fluctuates between 5.08×10^{-9} Hz

and 1.62×10^{-7} Hz. Assuming a C/A-code chip length of 293.05 m, the lowest frequency step corresponds to a range-rate resolution of 0.000047466 m/s. The resolution of double-based NCO is sufficient for GNSS signal simulation.

NCO Variable	Unit	Max. value	Integer bits	Fractional bits	Δ_f
Δ_φ	cycle	(14).3255	0	53	1.27×10^{-9} Hz
φ	cycle	16.7395	5	48	4.06×10^{-8} Hz
ΔC	chip	2.8645	2	51	5.08×10^{-9} Hz
C	chip	66.8635	7	46	1.62×10^{-7} Hz

Table 6-5 Double-based code and carrier NCO in Single Service scenario with 64 threads per channel

The fixed-point based, float-based and double-based NCOs differ significantly in the frequency resolution. The float-based implementation does not offer sufficient resolution even for a scenario with low sample. Especially frequency step of the code NCO is too high. The 64-bit fixed-point based implementation offers good resolution for the Single Service scenario. The resolution for Broadband scenario is sufficient for scenarios without heavy dynamics. The double based NCO offers highest number of significant bits and results in the best frequency resolution. Nevertheless, floating point based truncation and oscillation of the resolutions results in irregularities in the generated signal.

The theoretical performance of the signal generation algorithm in terms of speed was evaluated. All three NCO implementations – fixed point, float and double were taken into account. The performance is calculated in cycles of GPU clocks needed for computation of one signal sample of one signal channel of L1 C/A in Single Service configuration. Each NCO algorithm was evaluated in terms of enumeration of all arithmetic operations per sample and the throughput of each operation in terms of parallel operations per clock cycle. The computation is based on CUDA operation throughput published in [69]. Table 6-6 lists the performance for older version of GPU with CC 2.1 and the GPU with CC 3.5, which is deployed in the Test System. The table shows, that the float NCO algorithm outperforms the other NCO designs. At older GPU architecture with CC 2.1., the fixed point NCO is slower by 41 % and the double NCO is slower even by 146%. On the Kepler GK110 GPU architecture with CC 3.5, the gap between float NCO and other designs is significantly smaller. The fixed-point NCO is by 32% slowed. The double based NCO is by 36% slower. The performance of double NCO on Kepler GPU is high enough to consider the implementation of the double NCO for high dynamic scenarios.

NCO design	Cuda CC 2.1 [clock cycles]	Cuda CC 3.5 [clock cycles]
64-bit (code) and 32-bit (carrier) fixed point	0.8229	0.1531
32-bit floating point - float	0.5833	0.1156
64-bit floating point - double	1.4375	0.1573

Table 6-6 Number of clock cycles for generation of 1 signal sample of 1 satellite channel of GPS L1 C/A by one multiprocessor of the GPU

Nevertheless the throughput of arithmetic operations is not the only factor influencing the final signal generation speed. Other important factors are the transfer of data between GPU main memory and the

shared memory, level of parallelization among multiprocessors and data transfer from GPU to CPU. These factors can in case of high load (high sample rate, many channels) overweight the generation speed in terms of number of clock cycles per sample.

6.5 Test Measurements

The quality of the generated signal was tested for the Single Service scenario. The generated digital was verified with the software GNSS receiver developed at ISTA [71]. The receiver features an DIF input interface to process sample files in post-processing modus. The Test System simulator was configured to quantify and store the generated signal samples in the receiver specific format. Four test scenarios with duration of 15 s were chosen for the test measurements. The description and features of the scenarios are listed in Table 6-7.

The first scenario is a field test measurement with the test receiver under open sky conditions. The second scenario is a simulation with float samples and fixed-point NCO. The third scenario is a simulation with double samples and fixed-point NCO. The fourth scenario is a simulation using float samples and float NCO. The receiver was set to process all the scenarios with the same signal processing settings and integration time T_{int} of 0.02 s.

No.	# sats	C/N₀	NCO	Sample	T_{int} [s]
1	--	38.5	--	--	0.02
2	12	53	fixed point	float	0.02
3	12	53.5	fixed point	double	0.02
4	12	54.7	float	float	0.02

Table 6-7 Settings of the test scenarios

The precision of the tracking loop output is evaluated in terms of the root mean square (RMS) of the tracking loop error. The C/N_0 of each scenario is different. Therefore, the tracking look error cannot be compared among the scenarios. The measured error is compared with the theoretical thermal error for the given C/N_0. The thermal error for delay lock loop (DLL) is given by [46]:

$$\sigma_{tDLL} = \sqrt{\frac{B_n}{2\frac{C}{N_0}} D\left[1 + \frac{2}{T_{int}\frac{C}{N_0}(2-D)}\right]}, D \geq \frac{\pi R_c}{B_{fe}}$$

$$\sigma_{tDLL} = \sqrt{\frac{B_n}{2\frac{C}{N_0}}\left[\frac{1}{B_{fe}T_C} + \frac{B_{fe}T_{int}}{\pi - 1}\left(D - \frac{1}{B_{fe}T_C}\right)\right]\left[1 + \frac{2}{T_{int}\frac{C}{N_0}(2-D)}\right]}, \frac{R_c}{B_{fe}} < D < \frac{\pi R_c}{B_{fe}} \quad \textbf{(6-7)}$$

$$\sigma_{tDLL} = \sqrt{\frac{B_n}{2\frac{C}{N_0}}\frac{1}{B_{fe}T_C}\left[1 + \frac{2}{T_{int}\frac{C}{N_0}}\right]}, D \leq \frac{R_c}{B_{fe}}$$

where σ_{tDLL} is RMS of the DLL error in chips, B_n is loop noise bandwidth in Herz, B_{fe} is double-sided front-end bandwidth in Herz, T_C is chip length in seconds, R_c is chipping rate in chips/s, T_{int} stands for integration time in seconds, D is early-to-late correlator spacing in chips. For the DIF test scenario with correlator spacing $D = 1$ and $B_{fe} = 10.23$ MHz, the first equation applies.

The thermal error for phase lock loop (PLL) is given by [46]:

$$\sigma_{tPLL} = \frac{1}{2\pi}\sqrt{\frac{B_n}{C/N_0}\left[1+\frac{1}{2TC/N_0}\right]}, \tag{6-8}$$

where σ_{tPLL} is RMS of PLL tracking loop error expressed in cycles and B_n is loop noise bandwidth in Herz. The thermal error for phase lock loop (PLL) is given by [46]:

$$\sigma_{tFLL} = \frac{1}{2\pi T}\sqrt{\frac{4FB_n}{C/N_0}\left[1+\frac{1}{TC/N_0}\right]}, \tag{6-9}$$

where σ_{tFLL} is RMS of FLL tracking loop error expressed in Hz, factor $F = 1$ for high C/N_0 and $F = 2$ near tracking threshold. The test receiver noise bandwidth settings of DLL $B_n = 0.5$ Hz, PLL B_n=10 Hz and FLL B_n =1 Hz were used in the computation of the thermal error.

An example of the tracking loop error measured by the test receiver is depicted in Figure 6-2, where e_{DLL}, e_{PLL} and e_{FLL} are the tracking loop error of DLL, PLL and FLL respectively. On the left, the error loop output for a chosen satellite from scenario No. 2 is presented. On the right, output for a chosen satellite from scenario No. 4 is given.

The RMS value of the tracking loop error is computed from the output data without the initiation period of 5 s. Table 6-8 compares the RMS error of DLL, PLL and FLL of each scenario with the theoretical RMS thermal error for the respective C/N_0.

No.	σ_{tDLL}	σ_{DLL}	Ratio DLL	σ_{tPLL}	σ_{PLL}	Ratio PLL	σ_{tFLL}	σ_{FLL}	Ratio FLL
Unit	**[chip]**	**[chip]**		**[cycle]**	**[cycle]**		**[Hz]**	**[Hz]**	
1	0.0061	0.0420	6.9232	0.0060	0.0154	2.5674	0.6024	1.8396	3.0539
2	0.0011	0.0075	6.6527	0.0011	0.0020	1.8075	0.1127	0.5151	4.5708
3	0.0011	0.0080	7.5676	0.0011	0.0023	2.1259	0.1064	0.5371	5.0482
4	0.0009	0.0180	19.5006	0.0009	0.0033	3.6092	0.0927	0.4403	4.7516

Table 6-8 Theoretical thermal error RMS (σ_t) and measured RMS error of the tracking loops (σ)

Regarding the DLL error, both fixed point NCO scenarios deliver better precision than the float based NCO scenario. The PLL and FLL show also higher error for float NCO, but the difference is smaller. This confirms the analysis for NCO design presented in Table 6-4 that described the insufficiency of the resolution resulting in low code phase precision and frequency step. The field test scenario delivers higher absolute tracking loop error. Nevertheless, the ratio in comparison to the thermal error is better for DLL and FLL. The reason is probably the limited resolution of the receiver, as the receiver is optimized for lower C/N_0 than the value used in the simulation scenarios.

No significant difference is observable between scenarios No. 2 and No. 3, i.e. between double and float data types for sample representation. The Single Service scenario, the design of float samples left 12 bits for amplitude resolution. When double is used for the samples, the full resolution of the PAC with 21 bits is used and tracking could be even more precise. The measurements show no significant difference. The reason might be that the resolution of the replica in the test receiver is also limited to a single-precision data type and by lookup table resolution for performance reasons.

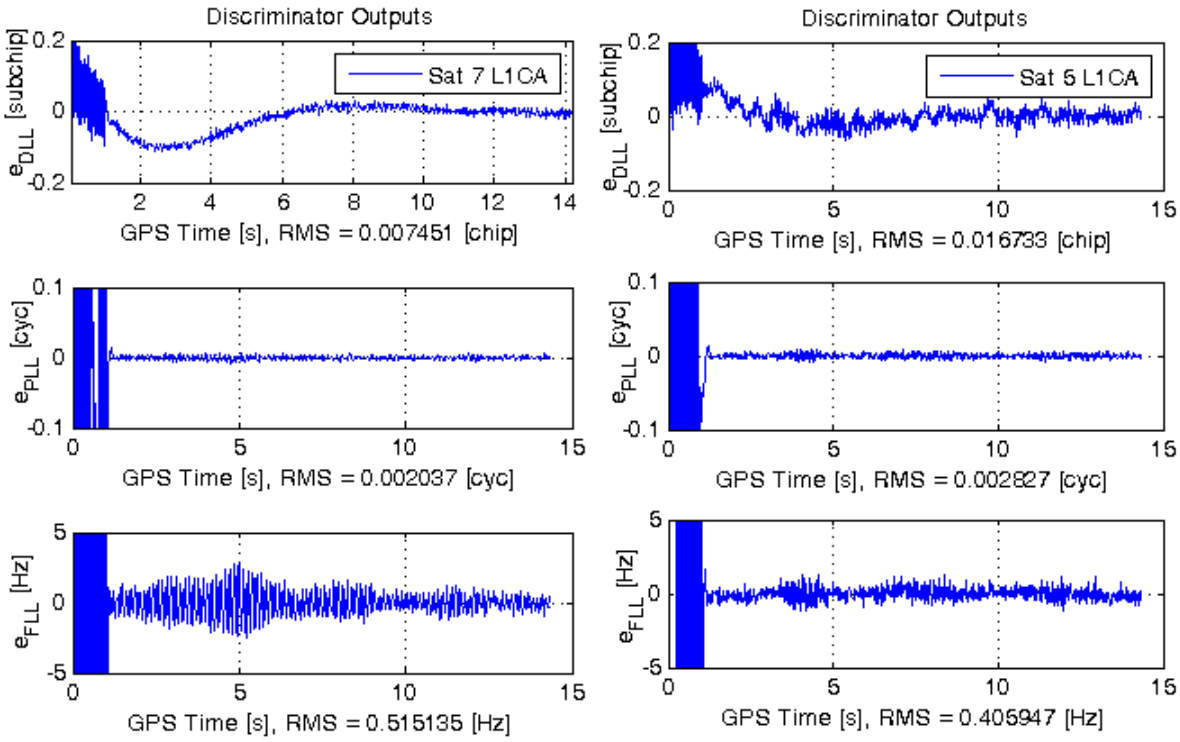

Figure 6-2 Scenario 2 - float samples, fixed point NCO – left, scenario 4 - float samples, float NCO - right

This chapter analyzed the influence of the resolution of the digital signal generation on the quality of the generated signal and on the limitations posed on signal generation scenario settings. Three key variables were chosen for the analysis.

The analysis was applied on the Test System, which is specific due to software implementation for GPU and CPU, where possible data types are very limited and their choice has tremendous influence on instruction throughput and is a key factor to empower real time capability of multiservice broadband signal generation.

The chances and limitations for GNSS signal generation were evaluated for the three variables and showed that the test system is able to generate high quality signal not only for minimal L1 C/A scenario, but for a Broadband configuration with extremely high sampling rate and multiple GPS and Galileo frequencies and services as well. For the Single Service scenario the range of user settings can be very flexible including high relative channel power differences and noise generation. For the Broadband scenario, the user settings are more limited, but still sufficient for standard configurations.

7 Digital Signal Generation

In this chapter, the concepts and algorithms for parallelization of the signal generation on a GPU based simulator architecture are described. Implementation of the algorithms and performance measurements on the Test System are given. The chapter proceeds from common concepts for single service generation, over algorithms specific to the implemented GNSS signal services, to parallelization of generation of multiple signal services and data transfer.

The task performed by the digital signal generation module of a simulator is to compute the digital GNSS signal arriving at the receiver antenna from the satellites in sight. The signal is a sum of signals of the available GNSS signal services. It can be described as

$$S(t) = \sum_{i=0}^{k} S_i(t), S_i \in \{L1\ C/A, E1\ OS, E5ab, \dots\}, \tag{7-1}$$

where *k* is number of available signal services and *S(t)* is the compound GNSS signal. The GNSS signal services differ in number of signal components, modulation schema, PRN codes, amplitude and frequencies of code, carrier and data.

In the framework of this thesis, three signal services were chosen for design, implementation and testing. These are GPS L1 C/A, Galileo E1 OS and Galileo E5ab. GPS L1 C/A was chosen for being most common service used by GNSS receivers. Galileo E1 OS was chosen for being placed in the same frequency band to enable tests with single frequency multiple services scenario. Galileo E5ab occupies the lowest frequencies of the GNSS signal spectrum. Successful simultaneous generation of L1 and E5 can demonstrate the full GNSS broadband generation performance. Additionally, it is the most complex signal in terms of signal components can serve as demonstration of the upper limit on single signal generation performance of the system.

The chosen GPS and Galileo signals are modulated with BPSK and BOC schemes. They use CDMA multiplexing method. Each satellite channel of a service is assigned a specific spreading code sequence with length of thousands of bits. This sequence is typically called PRN code and the bits of the spreading code sequence are called chips.

The signal of the widely spread signal service GPS L1 C/A is modulated with BPSK technique with PRN code rate 1.023 MHz. The spreading code contains 1023 chips. The signal level at receiver antenna at time point *t* is given by

$$S_{L1C/A}(t) = \sum_{p=1}^{m} D_p(t-\tau_p) C_p(t-\tau_p) A_p(t-\tau_p) \cos\left(2\pi f_p(t-\tau_p) + \phi_p(t-\tau_p)\right), \tag{7-2}$$

where *t* is arrival time of the signal at the receiver antenna, *m* is number of satellite channels, τ_p is signal travel time, D_p is the data bit, C_p is the PRN chip, A_p is amplitude, ϕ_p is carrier phase and f_p is the carrier frequency. The equations of the E1 OS and E5ab signal services are given in the respective sections below.

The GNSS digital signal generation task can be decomposed from data point of view to generation of samples of individual signal channels and addition of these samples. The schema of the task decomposition is given in Figure 7-1. Samples of *m* satellite channels are generated and added to the

sample stream of the whole signal service. The samples of one signal service are added to sample stream of the compounded multi-service signal.

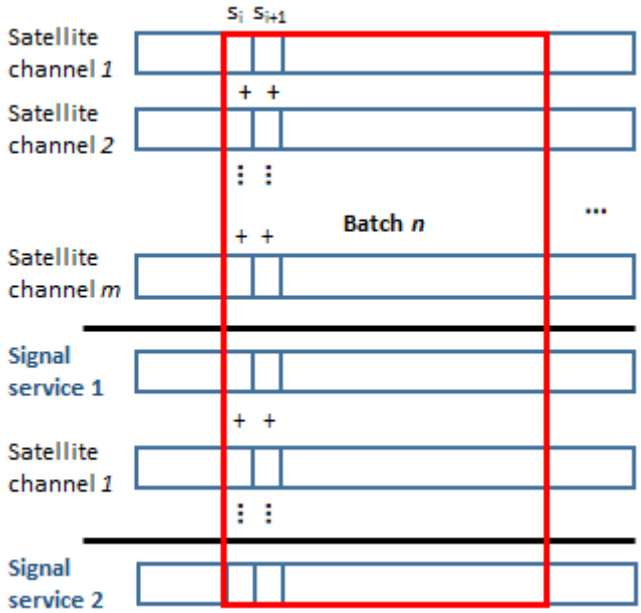

Figure 7-1 Digital signal generation from data point of view

The nature of CPU as well as GPU architecture is data oriented. An effective algorithm designed for execution on such a processing architecture needs to work on big data sets to reach maximal performance. In this work, generation, addition and transfer of samples process successively long sequences of samples called **batches**. First, all channels of one batch are computed and added, as outlined in Figure 7-1. Then, each complete batch is transferred from GPU over CPU to DAC as a whole and in time order to emulate continuous service. The size of a batch is given by the length of the signal microepoch, for which signal parameters are computed and used as being constant. These constant signal parameters for one microepoch are called **fixed parameters** in this text. The microepoch refers to the definition given in the section 5.2.

7.1 Generation of a Single Signal Service

With respect to the architecture of the GPU and the features of CUDA interface, the design of the single service digital signal generation was split into five key parallelization concepts:

1. Parallelization among Multiprocessors and their Cores of a GPU
2. Memory Concept for Modulation Data: PRN Codes
3. Memory Concept for Signal Samples
4. Single Service Signal Generation Loop
5. Memory Concept for Message Bits and Secondary PRN Codes
6. Quantization and Coding of Samples
7. Computation of Signal Samples

These concepts are described on example of the GPS L1 C/A service. The particular modifications for the Galileo E1 OS and E5ab services are given in the respective sections below.

7.1.1 Parallelization among GPU Multiprocessors and their Cores

For the design of parallel execution of the single service signal generation among multiprocessors of the GPU, three objectives were set. First, the algorithm should parallelize the generation of a single service as well as multiple GNSS signal services among the multiprocessors. Second, the algorithm must be flexible with respect to the number of multiprocessors, to be applicable for different GPUs with typically two up to sixteen multiprocessors. Third, the algorithm should minimize data transfer between the multiprocessor and minimize the need to synchronize the execution on the multiprocessors.

The parallelization algorithm for generation of samples of one signal service signal was designed to fulfill these objectives. The schema of the algorithm is given in Figure 7-2. In the figure, x stands for the number of multiprocessors. Each of x successive sample batches of the signal service is generated on one multiprocessor in parallel the other batches. In CUDA, the kernel (i.e. CUDA specific parallel function described in section 5.6) is configured to generate one batch of the signal service. It is launched x-times to be on x multiprocessors in parallel using the CUDA threading concept with x blocks of threads.

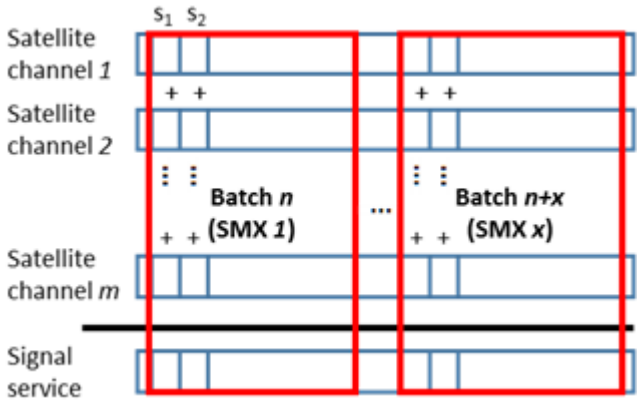

Figure 7-2 Parallelization of signal generation among multiprocessors (SMXs)

As a result, each of x batches of samples is generated on one multiprocessor. Practical tests with GPU Nvidia GeForce GT 460 with two multiprocessors proved the setting of number of block of threads equal to number of multiprocessors on the GPU to be nearly optimal. The setting of twice as many blocks where one multiprocessor generates two batches gave the best results. Higher number of batches per multiprocessor did not bring any additional improvement.

The algorithm for parallelization of the computation of the signal of a single signal service was designed in concord with the CUDA threading concept described in detail in section 4.6. The assignment of the threads in one block to the computation of the individual signal samples is depicted in Figure 7-3. One block of threads comprises $m \times (32 \times p)$ threads, where m is number of satellite channels and p is number of warps. Each channel is generated using p warps, i.e. $p \times 32$ threads. Each thread generates the sample s_i then the sample $s_{(i+32p)}$, sample $s_{(i+64p)}$ and so forth. For batch size of b samples, one thread generates $b/(32p)$ samples of one satellite channel. Immediately after the generation of the sample of the respective channel, the thread adds the sample to the signal service sample using a special CUDA operation for atomic addition.

In this way, each thread holds the data of a single signal channel in its registers. This data comprise the fixed parameters of the signal channel, the code and carrier NCO and other modulation specific auxiliary data. The frequency control word of the NCO is set to be 32*p* times the multiple of the carrier or code phase increment of the respective satellite channel. Further details about the NCO variables were given in section 6.4.

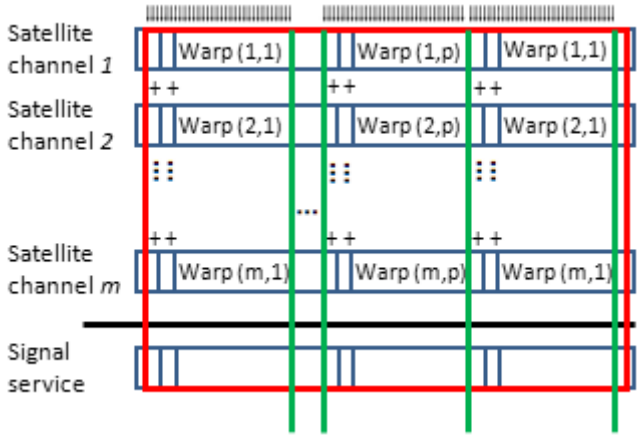

Figure 7-3 Parallel computation of a batch with one block of threads

7.1.2 Memory Concept for Modulation Data: PRN Codes

The challenge specific to GNSS signal generation is the concept for placement of the relatively long PRN sequences. The PRN codes for GPS L1 C/A signal service comprise 1,023 chips. The PRN codes for E1 OS service comprise 4,092 chips for data a 4,092 for pilot component. The PRN codes for E5ab comprise 10,230 chips for each of the four components. Secondary codes have lower frequency, for generation of one batch of samples only one or two following secondary chips are needed.

At the time of generation of each sample, the respective chip must be retrieved. In case of Single Service configuration defined in section 5.5, just 22-23 samples are generated for one chip. In case of Broadband configuration, 977 samples are generated per chip. Per batch, ca. 4,654 and 104 chips are needed for L1 C/A signal generation for the two configurations respectively. The number of registers per thread (64 on CC 3.5) is too small to hold this number of chips along with other data without regular reload from main GPU memory (also called global memory below). The access to the GPU main memory is costly; it takes 400-800 clock cycles.

In this work, the placement of parts of PRN sequences for the generated satellite channels to the shared memory is proposed. It is not possible to place whole PRN sequences to the shared memory, as even in case of shortest GPS L1 C/A codes and 12 channels, the chips would fill whole space leaving no capacity for generated samples. Therefore, a part of PRN sequence (called **PRN part** further on) of every channel is preloaded to the shared memory. The size of this part for each channel is set to $k \times 32$ chips. Figure 7-4 shows the proposed occupancy of the shared memory.

Ideally, the k should be set equal to number of warps per satellite channel. Then all threads are employed in order to load the chips from the main GPU memory, which improves the performance. The procedure of loading the sequences is visualized in Figure 7-5. The choice of the value of k balance the occupation of the shared memory between samples and PRN parts. The higher the number

of samples stored in the shared memory, the lower the number of copies to GPU main memory resulting in higher generation speed. On the other hand, the higher the number of chips in PRN part, the lower the number of reloads from the main GPU memory. For the Broadband configuration with SR = 1.4 Gsps, the best results on the Test system were reached with $k = 2$ for L1 C/A, $k = 1$ for E1 OS and $k = 2$ for E5ab.

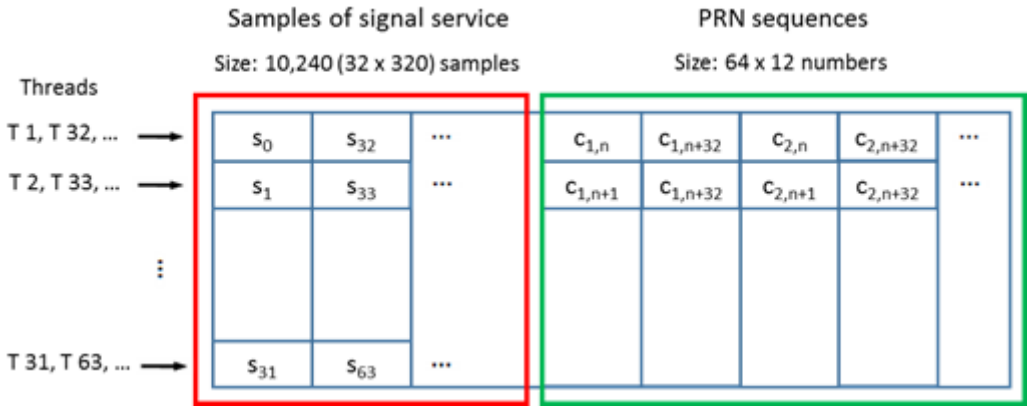

Figure 7-4 Occupancy of shared memory of a multiprocessor – L1 C/A generation

In the similar work presented in [38], usage of texture cache for PRN codes is proposed. Also in the CUDA Programming Guide [69], the suitability of this cache for such computation patterns is stressed. The cache should be used for data that are preferably read-only and are repetitively used in the kernel. At older Nvidia GPU architecture, the texture cache could be used only with special texture objects and texture access patterns. At CUDA CC 3.5 (Kepler GK 110) architecture, the texture cache can be directly allocated for general arrays of variables.

The size of texture cache in CC 3.5 is 48 KB. The L1 C/A codes fit there completely, longer codes and multicomponent signals would need a reload of data. For example, for generation of 12 channels of E1 OS B+C signal with length 4092 chips and 4 B per sample, 383.625 KB would be needed. Even if only 1 B per sample were used, a regular reload of data from GPU main memory would be necessary.

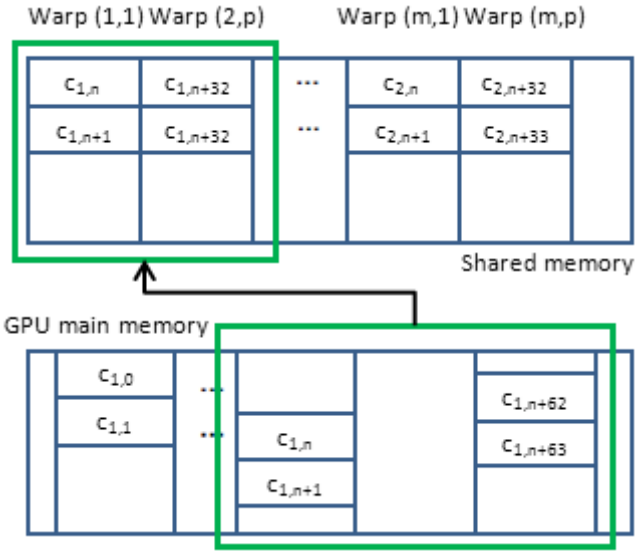

Figure 7-5 Loading of PRN sequences to shared memory

The usage of texture memory for PRNs of L1 C/A signal was tested. The load of the PRN sequences from the device memory to the texture cache was invoked by using special CUDA keyword `const __restrict__`. The load operation could also be enforced with a special function. The performance of the signal generation was tested and yielded following results.

The kernel run time was measured by CUDA Visual Profiler. The mean run time of a kernel was computed as an arithmetic mean from generation of 3,200 kernels. The mean run time for one run of the kernel using texture memory and generating 12 channels using 2 blocks of threads with 102,400 samples was measured as 690 µs. The mean run time of kernel using shared memory was 490 µs for the same settings. In showed, that in contrast to the expectation, the speed of the kernel with texture cache is significantly lower. Additionally, a kernel with PRNs codes in device memory was implemented for comparison. The speed of this kernel with the same settings was 660 µs.

These results and the fact, that the cache is too small for any signal with more complex coding than L1 C/A motivated the decision to stick to shared memory and to deprecate the usage of texture cache in this work.

7.1.3 Memory Management Concept for Signal Samples

The generated signal samples are stored in the shared memory of the multiprocessor. The shared memory is organized to 32 banks and each thread of one warp can access one number on one of these banks in parallel. For optimal usage of the shared memory, the samples of the final signal service are ordered sequentially in the shared memory and aligned to 32, as depicted in Figure 7-4. In this way, a warp generates 32 successive samples and accesses the respective samples on the 32 banks of the shared memory in parallel. The samples are generated as single precision floating point numbers. Discussion of this choice was given in section 6.3. The shared memory is optimized for single precision numbers, access to double or single byte numbers is less efficient. A sequence of 10,240 samples, called **subbatch** further on, is stored in the shared memory. It occupies 40,960 B of space, 8,192 B of space is left for PRN codes and modulation data.

When the generation of a subbatch of samples with all satellite channels is finished, all threads are synchronized. Then the generated signal samples are copied from the shared memory to the GPU main memory. For this task, all the warps of threads are reassigned to transfer the samples of the signal service to the main memory in sequential order. This assignment of threads to copy of samples is depicted in Figure 7-6.

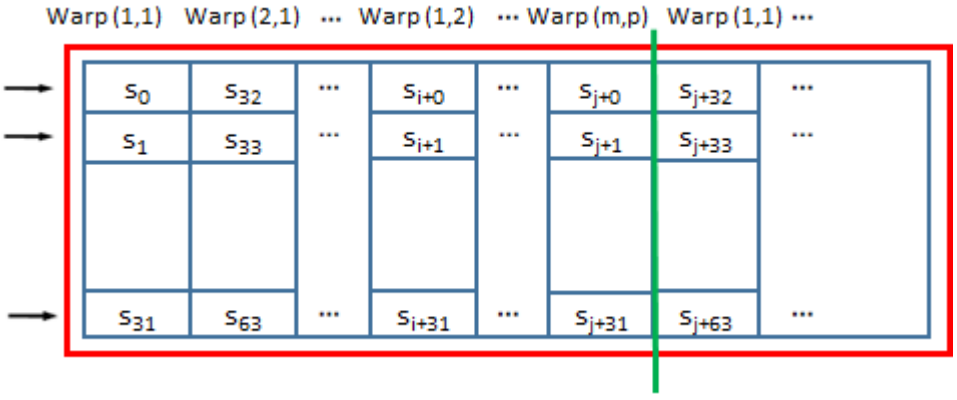

Figure 7-6 Parallelization of transfer of signal samples from shared to GPU memory

The synchronization operation is necessary to secure that the generation of all channels is finished by all threads before the copy. The employment of all threads in the copy process improves the throughput of the operation.

7.1.4 Memory Concept for Message Data and Secondary PRN Codes

In contrast to the primary PRN codes, the rate of secondary PRN codes and the rate of message data is by about three order of magnitude lower. Within one sample batch with 102,400 samples, only one or two symbols are needed within the Test System configurations. The minimum number of samples per message data bit is 91,424 samples for E1 OS with data rate = 250 symbols per second when Single Service configuration with SR = 22.856 MHz would be used. The maximum number is 28 million samples in case of L1 C/A (data rate = 50 symbols per second) when Broadband configuration with SR = 1.4 Gsps is used. Therefore, to load longer sequences of message data or whole secondary codes by a kernel to the multiprocessor memory is not necessary.

The relevant secondary PRN chips and message data are overgiven to the kernel as a part of fixed parameters. They are hold in registers of each thread. The algorithm for generation of the samples proceeds in steps of batch, subbatch and PRN part, which are not aligned to the length of a message symbol or secondary chip. The sample, where the next message symbol or secondary chip starts is called sample with bit flip further on. The number of this sample within a batch is `nSampleInSubbWithBitFlip` and it is overgiven to the kernel in as a part of fixed parameters.

First, a simple algorithm for message bit modulation was proposed. It serves for comparison with a design of a high performance solution. The algorithm places for each sample an if-clause testing of bit flip occurrence into the sample generation loop. The if-clause then overwrites the actual bit with the next bit.

This algorithm would increase the number of clock cycles needed to compute one sample by 1/32. This number is given throughput by the throughput of the comparison operation being 32 operations per clock cycle on CC 3.5 [69]. This value applies only when the result is identical for whole warp, which is the case almost all the time. The exception is when the bit flip occurs. Then the execution of the warp is split to two parts with identical results within each of them.

Thereafter, the idea to run the loop over PRN part only to the bit flip was evaluated. The samples of one PRN part are generated only up to the bit flip. Thereafter, a test of the art of end of the loop is needed. In case of positive outcome, the message bit value is changed to the following bit and an additional loop that finishes generating the samples in the actual PRN part is run.

This algorithm causes additionally one `min()` operation and one if-clause per PRN part. Number of samples in PRN part (`nSamplesInPrnPart`) are for typical 64 chips in PRN part and Single Service scenario (SR = 22.856 Msps) 1,429 samples, for Broadband scenario (SR = 1.4 Gsps) these are 87,585 samples. Therefore the number of clock cycles for generation of one sample is increased by 2/1,429 and 2/87,585 respectively. Performance of both algorithms was measured on the Test System. Using this algorithm, the speed of L1 C/A generation in Single Service scenario increased in contrast to the straightforward implementation by about 10%.

7.1.5 Single Service Signal Generation Loop

The memory concepts and parallelization concept among multiprocessors and their cores were described in previous sections. These concepts sum up to a signal service generation algorithm, which consists of a generation loop with several levels.

The upper level loop runs on the host computer. It generates the batches of samples in a successive manner. One kernel function per signal service is invoked and it is executed $(x \times y)$-times on the GPU. In other words, the kernel is run by x blocks of y threads to generate x successive batches of samples. Each block is executed one multiprocessor. One block of threads comprises $y = m \times 32p$ threads, where each $32p$ threads generate one of m satellite channels.

The part of the algorithm executed in the kernel is given in CUDA C in simplified manner as follows:

```
for (nSubbatch=0;nSubbatch<NUM_SUBBATCHES_IN_BATCH;nSubbatch++)
{
  for (nPrnPartInSubbatch=0;…<ceil(NUM_SAMPLES_IN_SUBBATCH/(nSamplesInPrnPart)); …++)
  {
       syncthreads();
       loadPrnParts(…);
       _syncthreads();
       for (nSubsample=nSatThreadId+(nPrnPartInSubbatch*nSamplesInPrnPart;
              nSubsample<nEndSample;nSubsample+=nThreadsPerSat)
              atomicAdd(psSubbatch+nSubsample,computeSample(…));
       if (nEndSample==nSampleInSubbWithBitFlip){
              fBit=fBitNext;
              nEndSample=min(nSubbatchLength,nSamplesInPrnPart*(nPrnPartInSubbatch+1);
              for (;nSubsample<nEndSample;… )
                     psSubbatch[nSubsample]+= computeSample(…);}
  }
__syncthreads();
addToSampleStreamAndQuantify();
__syncthreads();
}
```

In the kernel, three lower level loops are executed. The top most loop in the kernel runs once for each subbatch with number `nSubbatch` of the batch to be generated. In this loop, the samples of the respective channel are generated and then all threads run at the multiprocessor are synchronized (function `__syncthreads`). The quantization and copy of subbatch to the main GPU memory follows (function `addToSampleStreamAndQuantify`). Before the generation of the next subbatch, the threads are synchronized again, to make sure that all the samples are copied before being overwritten by the next subbatch.

At the beginning of the next lower loop, PRN parts with order number `nPrnPartInSubbatch` comprising $k \times 32$ chips are loaded from the GPU memory to the shared memory (function `loadPrnParts`). When only one warp per channel is used, no synchronization is needed. Otherwise, the completion of the reload of parts of the PRN part as well as the completion of the generation of samples of the preceding PRN part by the other warps generating the same channel must be secured by synchronization.

The lowest loop generates samples within one PRN part. The number of the thread within the warps generating one satellite channel `nSatThreadId` is used to compute the sample `nSubsample` of the

channel within the subbatch to be generated. Each thread then continues with step of `nThreadsPerSat` = 32*p* samples parallel to other threads in the block to generate the next (32*p*)-th sample of its assigned satellite channel. The thread then adds the sample to the subbatch of the signal service in the shared memory `psSubbatch` with the CUDA atomic operation `atomicAdd`. The lowest loop runs up to the `nEndSample`, which is the minimum from the end of PRN part, end of subbatch, and the bit flip.

When bit flip occurs, the actual bit and secondary chip is changed and the loop continues until the end of the PRN part of end of the subbatch.

7.1.6 Quantization and Coding of Samples

The samples are generated in single precision floating type. Each DAC and a receiver specific IF file interface uses a specific input data format. The generated samples need to be quantified and coded for each implemented interface. For maximal performance, the quantization of the samples was parallelized and executed on the GPU. This accelerated the quantification in contrast to CPU significantly. Even higher improvement of the total generation speed was reached by the fact, that the transfer of already quantized and therefore shorter samples from GPU to the host computer is faster.

The interface document of the PCIe-DAC board ICS-1572 used in the Test System defines the sample format to be integer 16-bit number in two's complement format. This format is equivalent to the signed short integer data type in C++ and CUDA C. The quantization operation was implemented on the GPU as follows:

```
(short int)rintf(fQuantCoef*psSubbatch[nSubsample])
```

Float to short conversion is done using type-cast operation `(short int)`. The throughput of this type-cast operation is 32 operations per clock cycle for CC 3.5 [69]. The `rintf()` operation maps to a single instruction, but nominal throughput in operations per clock cycle is not published in CUDA documentation [69].

Even though CUDA offers native functions for rounded conversions of many data types, e.g. `__float2half_rn`, or `__float2ll_rn`, no similar function for conversion from short to float data type is included in CUDA 6.0 and CC 3.5.

For the correct quantification, the amplitude of the signal must be normed to range [-1,1]. Therefore, the maximal amplitude is calculated for each signal service. The quantification coefficient is then set as follows:

```
fQuantCoef= (float)((1<<15)-1)/fMaxAmp
```

7.1.7 Computation of Signal Waveform

The algorithm for signal generation performed by each thread strives to minimize the number of instructions for the sample computation to improve the total performance.

The signal generation equation is defined in (7-6) for L1 C/A. For Galileo E1 OS and E5ab, modulation equations are given in sections 7.3 and 7.4 respectively. Modulation of all CDMA based GNSS signals comprises message bits, PRN codes and sinusoidal carrier wave. The digital signal is generated with direct digital synthesis (DDS) technique using numerically controlled oscillators

(NCO), as explained in section 3.1.3. In section 6.4, the requirements on numerical precision of the NCO variables were analyzed. The choice of 64-bit fixed point variables for code phase and code phase increment and 32-bit fixed point variables for carrier phase and carrier phase increment was explained.

The key issue of the signal synthesis is the design of the NCOs for the computation of code and carrier phase and amplitude, as each operation within the NCOs is executed for each sample. The usage of a single NCO holding code phase was considered. Conversion of code phase to carrier phase is necessary for every sample. The conversion on Test System GPU is costly. It needs a division of a 64-bit number, which takes multiple clock cycles. Alternatively, conversion to single precision number with throughput of 8 operations per cycle (ops/cycle) and truncation of integer part with two bit shifts with 32 ops/cycle could be used. A separate carrier NCO needs only one accumulation (160 ops/cycle) and one shift operation (32 ops/cycle) per sample. It occupies two additional 4 B registers, which is acceptable on Kepler GK110 GPU architecture (64 registers per thread).

The PRN parts are stored in the shared memory. The actual message bit and secondary chip are held in registers and the inner loop runs just up to the end of the PRN part or up to the bit or secondary chip flip.

For the phase to amplitude conversion, a look-up table is a standard technique used in digital signal synthesis. As explained in section 6.3, the look-up table is inconvenient for implementation on a GPU. The Nvidia GPUs include multiple special function units (SFUs), which compute goniometric functions of float numbers in a single processor clock cycle. CUDA accesses these units with *__cosf* and *__sinf* functions, which deliver output resolution of 21.19 bits [70]. The Kepler GK110 multiprocessor comprises 32 SFUs. Another advantage of SFU utilization in contrast to a lookup table is the fact, that the other single-precision and double-precision cores can be employed for the other mathematical operations in sample generation (accumulation, bit shift, type conversion, multiplication) in parallel to carrier wave computation on SFUs.

After the evaluation of the NCOs, the code, data and carrier amplitude are represented as single precision floating-point numbers. They are multiplied to get the final sample value and then added to sum of all channels in the shared memory. Each multiplication is done in one float operation with throughput of 192 ops/cycle. A single multiply-and-add operation was considered for the last multiplication operation and addition to signal service in shared memory. The multiply-and-add operation offers the same throughput as a separated multiply or a separated add operation. However, an atomic operation is needed for the addition to the signal service in shared memory. A slower alternative would be the storage of separate channels and later addition after synchronization of threads. In CUDA, specific atomic add operation with high throughput in available, but no atomic multiple-and-add.

This general concept for signal sample computation was adjusted for each signal service according to number of components and modulation schema. The specific implementation for each signal service is given in following sections.

7.2 GPS L1 C/A Generation

The full description of GPS L1 C/A signal is published in the interface control document (ICD) of the service provider [47]. The signal uses binary phase shift keying (BSPK) modulation with one signal component carrying message data. Each satellite channel uses own PRN sequence with the length of

1,023 chips. The signal modulation equation is given by (7-2). The common signal service generation concepts were explained in previous sections. The kernel for L1 C/A generation is based on these concepts. The algorithm for computation of a sample in subbatch number `nSubsample` and its atomic addition to `psSubbatch` in shared memory is given in CUDA C as follows:

```
atomicAdd((psSubbatch+nSubsample),
       __cosf(__uint2float_rn(unCarrierPhasePerThread) * (PIx2div2upto32) )
       *psPrnPart[ulCodePhaseInPrnPartPerThread>>CODE_PHASE_RESOLUTION]
       *fSigAmpFactor*fBit);

ulCodePhaseInPrnPartPerThread+=ulCodePhaseDeltaPerThread;
unCarrierPhasePerThread+=unCarrierPhaseDeltaPerThread;
```

The value in carrier phase accumulator `unCarrierPhasePerThread` is converted from a fixed-point 32-bit number of cycles with 32 fractional bits to a float number (CUDA function `__uint2float_rn`) and then to radians (`PIx2div2upto32` $= 2\pi / 2^{32}$). The cosine CUDA function `__cosf` is applied. The code phase accumulator is a fixed-point 64-bit number with `CODE_PHASE_RESOLUTION` (= 48) fractional bits. The fractional part is truncated by bit shift operation (>>). Carrier wave amplitude, code chip, signal amplitude `fSigAmpFactor` and message bit `fBit` are multiplied to the final sample. The code and carrier accumulator value is increased by code and carrier phase increments `ulCodePhaseDeltaPerThread` and `unCarrierPhaseDeltaPerThread` respectively.

The performance of the L1 C/A generation was measured on the Test System with benchmarking configuration of IF = 30.69 MHz and SR = 130 Msps. The signal generation speed in Msps for 2-12 satellite channels is depicted in Figure 7-7. A single GPU was deployed in the Test System. For 12 satellite channels, the generation speed of 2,900 Msps was reached.

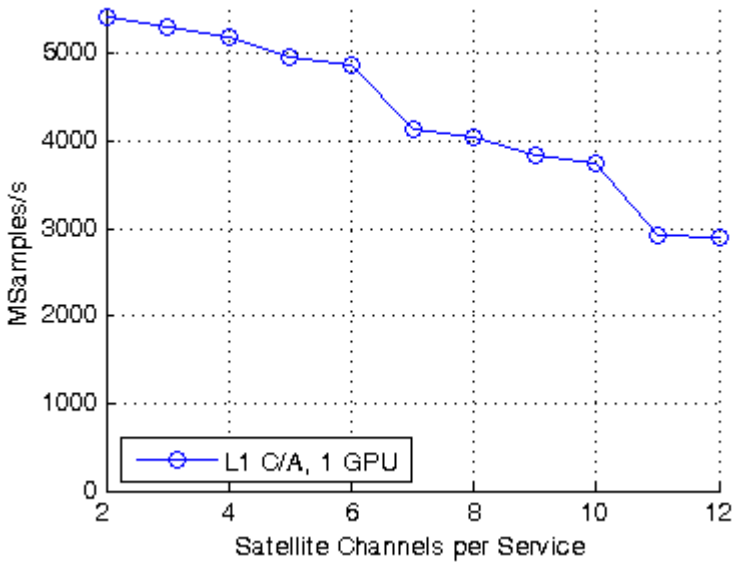

Figure 7-7 Signal generation performance of L1 C/A kernel with fixed-point NCOs using 1 GPU

As the figure shows, the speed of the signal service generation increases for lower number of channels. Naturally, the lower computational load of generation of fewer channel results in higher generation speed. The increase in speed for fewer channels is nevertheless not proportional. The reason is that significant time is consumed by operations that are independent of number of channels. These are the

transfer of samples from multiprocessor to the main GPU memory and the transfer of samples from GPU to the host computer memory. The overhead time for kernel and data copy invocations is also independent of the number of channels. Furthermore, the higher speed enforces a more frequent reload of PRN parts, which causes a minor slow-down for low number of channels. This issue could be easily solved by variable length of PRN parts dependent on the number of channels.

The benchmark values in Figure 7-7 show that the increase in speed for decreasing number of channels is not smooth. The generation of 11 and 12 channels takes almost the same time. The generation of 10 channels is by ca. 800 Msps faster that generation of 11 channels. The reason is that the maximum number of threads per multiprocessor is limited by 1024 and the threads operate in groups of 32 (warps) that execute always the same instruction. In the proposed algorithm, each thread is assigned to a single channel. Therefore the maximum number of warps p that can be employed on one multiprocessor for a generation of a given number of channels m is limited by $32pm < 1024$. The numbers of warps and threads per satellite used for the signal generation of all three GNSS signal services are given in Table 7-1.

# satellite channels	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
# warps (i.e. 32 threads)	16	10	6	6	5	4	4	3	3	2	2
# threads	1024	960	768	960	960	896	1024	864	960	704	768

Table 7-1 Number of warps and threads per satellite channel operating of a single multiprocessor

For the purposes of analysis of precision of digital GNSS signal given in Chapter 6, an alternative L1 C/A kernel with float-based NCOs was implemented. As explained in Chapter 6, it is applicable only for Single Service configuration. Using the same benchmarking configuration, the generation of 12 satellite channels was by 100 Msps, i.e. 3.4%, faster. For lower number of satellite channels, the difference between fixed-point and float-based implementation was decreasing.

The float-based implementation could be deployed in scenarios with a single frequency band and very high number of channels (e.g. multi-channel multipath modelling). Neither this option nor the multipath modelling has been included in the Test system so far.

In the Test system with Broadband configuration, the L1 C/A is generated with IF = 411.42 MHz and SR = 1.4 Gsps. With such high SR and IF, the generation is faster by ca. 70 Msps. The reason for the improvement is the lower number of bit flip events and lower number of PRN parts reload events per sample.

The generated L1 C/A signal was verified in Single Service configuration using the GNSS receiver SX-NSR by IFEN GmbH. No noise was added to the signal and the receiver measured C/N_0 = 75.51 dB-Hz. The power spectrum density of the signal measured by the receiver is given in Figure 7-8. The acquisition output for the PRN 1 with Doppler frequency of 1,251 Hz is given in Figure 7-9. The output of the tracking loops was analyzed and results are given in section 6.5.

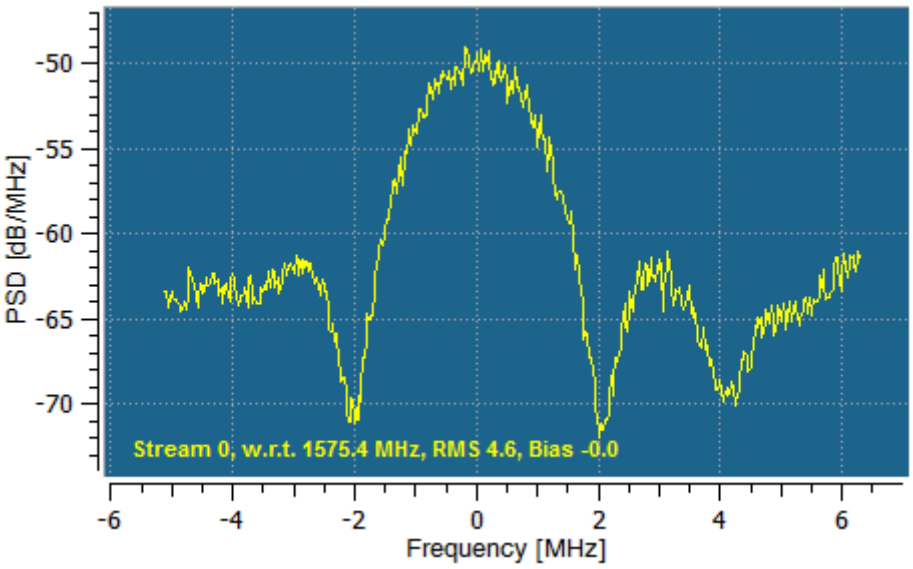

Figure 7-8 Power spectrum density of the L1 C/A signal

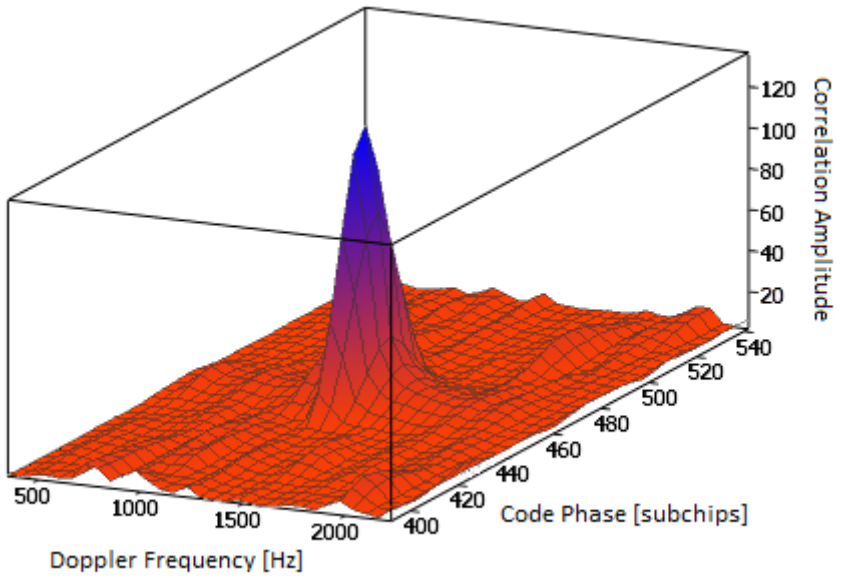

Figure 7-9 Acquisition of the L1 C/A signal

7.3 Galileo E1 OS Generation

The E1 OS signal is defined in the interface control document [63] of the Galileo service provider. The signal is modulated with CBOC (6, 1, 1/11) modulation scheme. It carries I/NAV navigation message. The signal comprises two baseband components: E1-B component is assigned to data message and E1-C component serves as a pilot. The chip rate of both components is 1.023 MHz, data rate of E1-B component is 250 symbols/s. PRN codes of E1-B has one layer with length of 4,092 chips and the code sequence length is equal to one data symbol. PRN codes of E1-C are two-layered. The primary code is 4,092 chips long and the code sequence length is equal to one secondary code chip. The

secondary code sequence is 25 chips long and common for all satellite channels. The Galileo E1 OS signal $S_{E1OS}(t)$ is defined as follows:

$$
\begin{aligned}
S_{E1OS}(t) &= \sum_{p=1}^{m} \sqrt{2P_{E5}}\,(t) s_{E1,m}(t)\cos(2\pi f_{E5}\,t)\,, \\
s_{E1,m}(t) &= \frac{1}{\sqrt{2}}\left(e_{E1-B,m}(t)S_{sc,B}(t) - e_{E1-C,m}(t)S_{sc,C}(t)\right), \\
e_{E1-B,m}(t) &= C_{E1-B,m}(t)D_{E1-B,m}(t), \\
e_{E1-C,m}(t) &= C_{E1-C,m}(t)C2_{E1-C,m}(t),
\end{aligned}
\tag{7-3}
$$

where $S_{sc,B}(t)$ and $S_{sc,C}(t)$ are subcarriers of E1-B and E1-C components common to all satellite channels. Variables $e_{E1\text{-}B,m}$ and $e_{E1\text{-}C,m}(t)$ are binary non-return-to-zero (NRZ) modulated combinations of code and navigation data message of satellite channel m. The symbol $s_{E1,m}(t)$ denotes the (unit mean power) baseband signal.

The subcarrier functions $S_{sc,B}(t)$ and $S_{sc,C}(t)$ are defined by equations

$$
\begin{aligned}
S_{sc,B}(t) &= \alpha\ \mathrm{sgn}\left(\sin\left(2\pi R_{s,a}t\right)\right) + \beta\ \mathrm{sgn}\left(\sin\left(2\pi R_{s,b}t\right)\right), \\
S_{sc,C}(t) &= \alpha\ \mathrm{sgn}\left(\sin\left(2\pi R_{s,a}t\right)\right) - \beta\ \mathrm{sgn}\left(\sin\left(2\pi R_{s,b}t\right)\right),
\end{aligned}
\tag{7-4}
$$

where $R_{s,a}$ and $R_{s,b}$ denote sub-carrier rates equal to 1.023 MHz and 6.138 MHz respectively. The parameters α and β are defined as follows:

$$
\alpha = \sqrt{\frac{10}{11}}, \beta = \sqrt{\frac{1}{11}}.
\tag{7-5}
$$

The subcarrier pattern is applied to every PRN code chip. One period of the subcarrier functions $S_{sc,B}(t)$ and $S_{sc,C}(t)$ is visualized in Figure 7-10.

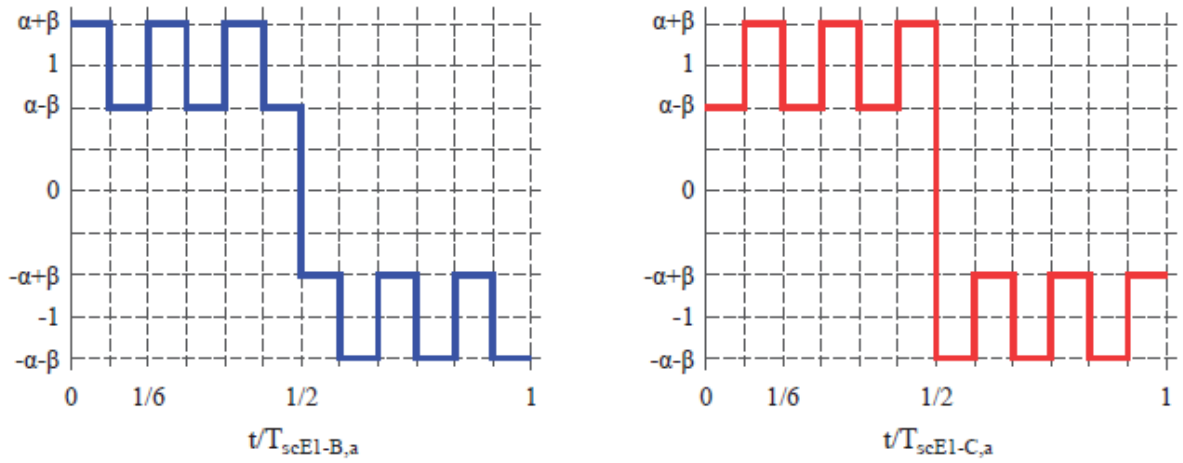

Figure 7-10 One period of the sub-carrier $S_{sc,B}$ (left), and $S_{sc,C}$ (right), [63]

Resulting from the subcarrier structure for pilot a data, the maximum amplitude is as follows:

$$A = \frac{1}{\sqrt{2}}(\,(\alpha+\beta)-(-\alpha-\beta)) = 1.3483 \qquad \textbf{(7-6)}$$

The algorithm for E1 OS generation follows the pattern of L1 C/A signal generation in all common topics. For the length of PRN parts 32 chips per signal channel component were chosen. 64 chips would also fit into the shared memory of the GPU, but 32 chips give for high SR (> 1 Gsps) slightly better performance. The PRN parts of data and pilot components are stored separately in one float number per chip format.

The E1 OS CBOC modulation differs from L1 C/A BPSK modulation in usage of the subcarrier pattern. Possible concepts for computation of the subcarrier pattern were examined. From the equation for E1 OS signal computation given in [63], the equation for subcarrier of both components as given in (7-4) was extracted. Two possible strategies of computation of value of this expression for each signal sample were proposed.

First, the value of the subcarrier can be computed analytically for each sample according to (7-4). In general, the analytical operations are more effective on the GPU than to access some memory representation (e.g. lookup table) of the function. This was the case for the carrier wave computation. The E1 OS subcarrier formula nevertheless contains the *sgn* operation, which is not intrinsic to the GPU architecture. It would need to be implemented as *x/abs(x)*, where function *abs(x)* is compiled on GPU to multiple operations. In total, 7 single instruction operations (additions and multiplications) and two *sgn* functions would be necessary to calculate the formula on the GPU used in the Test System.

The second strategy is to evaluate the periodical subcarrier function using a lookup table. Values of this periodical subcarrier function are then interpreted as a subcarrier code with 12 values per signal component. One bank (32 values with parallel access) of shared memory is assigned to the subcarrier codes (24 float values). The data is loaded from GPU main memory at the beginning of the kernel run. The subcarrier code phase is computed from PRN code phase. The phase computation together with the load of the subcarrier code from the shared memory costs only 5 single instruction operations per sample. As a result, this strategy was chosen for implementation in the Test System simulator.

The storage of subcarrier codes in registers was considered. It was evaluated as inefficient, because 24 additional registers per thread would shrink the number of threads per multiprocessor below the optimum. In the implementation with shared memory, the kernel uses 47 registers per thread. There are 65,536 registers on each multiprocessor in total, which results in 64 registers per thread when maximum number of threads per multiprocessor (1,024 threads) is employed. The constant cache with 8 KB was also considered, but the variables stored in this cache must be accessed independently of the thread number, which is not the case.

The algorithm for E1 OS signal generation is given in CUDA C as follows:

```
atomicAdd((psSubbatch+nSubsample),
(fSigAmpFactor
*__cosf(__uint2float_rn(unCarrierPhasePerThread) * PIx2div2upto32)
*
((psSubCarrData[(int)((double)(ulCodePhaseInPrnPartPerThread<<
      (64-CODE_PHASE_RESOLUTION)>>(64-CODE_PHASE_RESOLUTION))*
      SUBCARR_LENGTH_div2uptoCodeRes)]
*psPrnPartD[ulCodePhaseInPrnPartPerThread>>CODE_PHASE_RESOLUTION]
*fSymbol)
```

```
-
(psSubCarrPilot[(int)((double)(ulCodePhaseInPrnPartPerThread<<
      (64-CODE_PHASE_RESOLUTION)>>(64-CODE_PHASE_RESOLUTION))*
      SUBCARR_LENGTH_div2uptoCodeRes)]
*psPrnPartP[ulCodePhaseInPrnPartPerThread>>CODE_PHASE_RESOLUTION]
*fChipSecPilot))));

ulCodePhaseInPrnPartPerThread+=ulCodePhaseDeltaPerThread;
unCarrierPhasePerThread+=unCarrierPhaseDeltaPerThread;
```

In all common topics, the variables and concepts are identical to L1 C/A generation algorithm given in the previous section. The subcarrier code is stored in shared memory as `psSubCarrData` and `psSubCarrPilot`. The subcarrier code phase is calculated from code phase by truncation of integer part by bit-shift operations (>>, <<), type conversion, conversion to subcarrier phase (`SUBCARR_LENGTH_div2uptoCodeRes` = $12 / 2^{48}$) and truncation of fractional part.

The instruction throughput of the algorithm was evaluated using the instruction throughput data of CUDA operations published in [69]. In this document, the number or operations per clock cycles per multiprocessor is listed for the majority of common arithmetic operations. For the used operations, where the number is not known (e.g. atomicAdd), an estimation was used. The throughput of the algorithm evaluated to ca. 1.01 clock cycles per sample of one satellite channel on one multiprocessor with assumption of sequential execution of operations.

The assumption of sequential execution is nevertheless a simplification. Different instructions can be launched by warp schedulers of a multiprocessor in parallel. For example, 32 load/store operations can run parallel to 96 arithmetic float operations. Furthermore, the parallel launch depends on the availability of data and data locality. For example, if the PRN chips used for 32 successive samples of one warp are placed on the same shared memory bank, a bank conflict occurs and the load/store operation must be split and launched in two successive steps.

7.3.1 Performance and Verification

The performance of the E1 OS signal generation using the described algorithm was measured on the Test System with benchmarking configuration of IF = 30.69 MHz and SR = 130 Msps. No noise was added to the signal and the receiver measured C/N_0 = 75.51 dB/Hz. The generation speed in Msps for 2 to 12 satellite channels is given in Figure 7-11. For 12 satellite channels, the generation speed of 1,770 Msps was reached with a single GPU. Using the Broadband configuration of the Test System, the generation speed increased by ca. 10 Msps.

The number of threads per channels was the same as in case of L1 C/A (Table 7-1). The effect of steep increase in speed for number of satellites that fit the limitations of number of threads is visible in the performance graph in Figure 7-11, especially for 8 and 4 channels. In comparison to the L1 C/A signal generation, the generation speed is just about 60% for a high number of channels. The additional pilot PRN code and subcarrier on both components increase the computational load significantly. For a low number of satellites the difference is about 6%. This demonstrates that the occupancy of the processor and data transfer play an important role.

For the verification, the GNSS receiver SX-NSR and the GNSS signal simulator NAV-NCS from IFEN GmbH were used. The generated E1 OS signal was processed by the receiver. The response was compared to the response to the signal generated by the commercial signal simulator and processed using the similar settings.

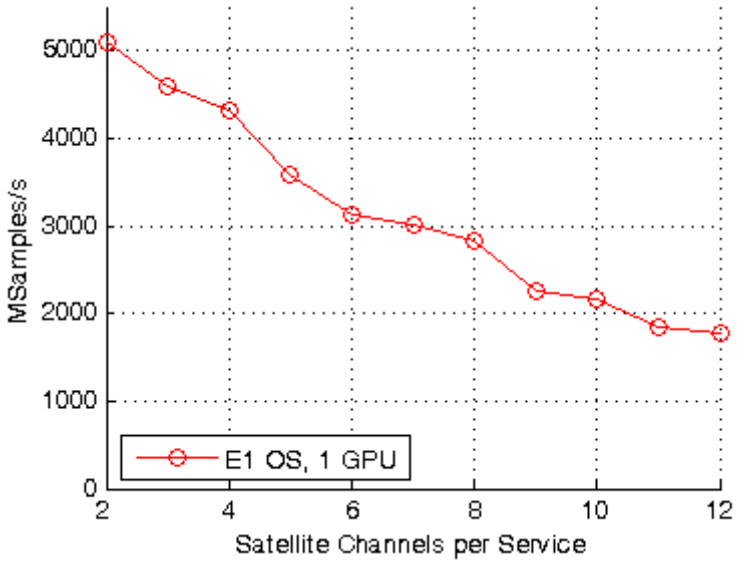

Figure 7-11 Signal generation performance of E1 OS kernel using 1 GPU

The two verification scenarios are listed in Table 7-2. In the scenario no. 1, the RF E1 OS signal was generated with the NAV-NCS simulator and simultaneously received by the SX-NSR receiver. In the scenario no. 2, the DIF E1 OS was generated by the Test System simulator. The generated DIF data were processed by the SX-NSR receiver with matching settings.

No.	Scenario	# sats	C/N_0 [dB/Hz]	T_{Int} [s]
1	NAV-NCS	12	50	0.004
2	TestSystem_E1OS	12	50	0.004

Table 7-2: Definition and settings of E1 OS verification scenarios

The RMS error of the tracking loop was measured by the receiver. For DLL, PLL and FLL, it is indicated in Table 7-3. For comparison, theoretical thermal error RMS was computed for the given C/N_0 based on (6-6), (6-7) and (6-8) for DLL, PLL and FLL respectively. The comparison of both shows similar results for DLL and FLL. In case of PLL, the in contrast to the commercial simulator.

The power spectrum density of the E1 OS signal simulated in the Single band scenario was measured by the receiver SX-NSR and it is given in Figure 7-12. The low bandwidth is visible on the deformation of the side loops of the spectrum. The acquisition output for the E1 OS signal is given in Figure 7-13. It depicts the signal generated without noise, therefore a high correlation value can be observed in the figure.

No.	σ_{iDLL}	σ_{DLL}	Ratio DLL	σ_{iPLL}	σ_{PLL}	Ratio PLL	σ_{iFLL}	σ_{FLL}	Ratio FLL
Unit	**[chip]**	**[chip]**		**[cycle]**	**[cycle]**		**[Hz]**	**[Hz]**	
1	0.0007	0.0023	3.38	0.0019	0.1093	56.05	0.9770	7.2752	7.45
2	0.0007	0.0027	3.89	0.0019	0.0072	3.69	0.9770	7.7756	7.96

Table 7-3: Theoretical thermal error RMS (σ_i) versus measured RMS error of the tracking loops (σ)

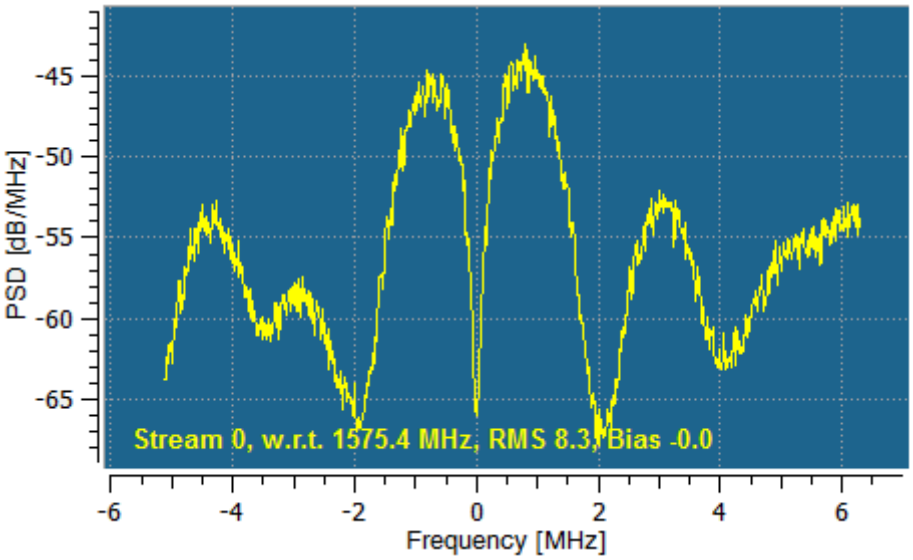

Figure 7-12 Power spectrum density of the E1 OS signal

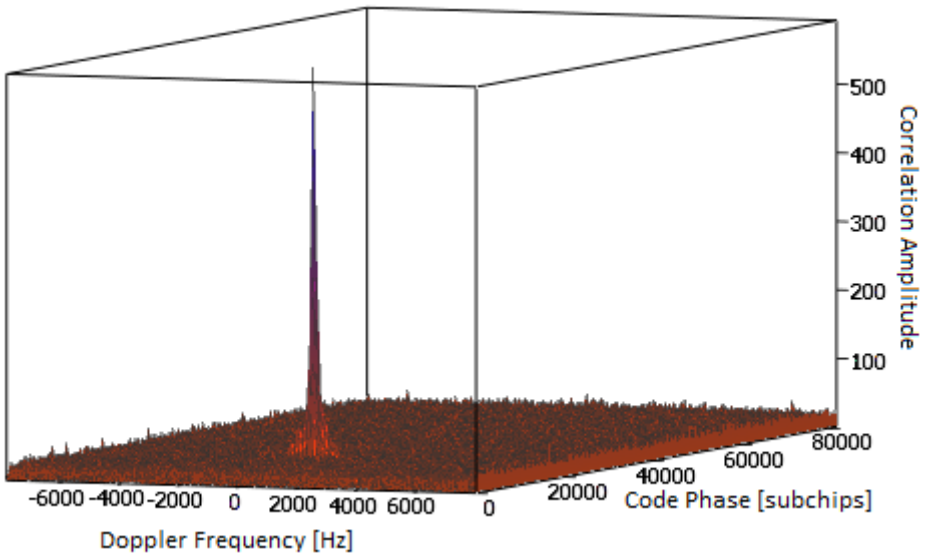

Figure 7-13 Acquisition of the E1 OS signal

7.4 Galileo E5ab Generation

The Galileo E5ab signal defined in Galileo Interface Control Document [63] uses AltBOC (15, 10) modulation. It consists of four signal components E5a-I, E5a-Q, E5b-I and E5b-Q. The E5a and E5b signals can also be received separately as BPSK(10) each with central frequency higher and lower by 15.345 MHz relative to the E5 central frequency 1,191.795 MHz. The signal modulation is given by equation

$$S_{E5}(t) = \sqrt{2P_{E5}}\left[s_{E5-I}(t)\cos(2\pi f_{E5}t) - s_{E5-Q}(t)\sin(2\pi f_{E5}t)\right], \tag{7-7}$$

where P_{E5} is signal power, $s_{E5\text{-}I}$ and $s_{E5\text{-}Q}$ are in-phase and quadrature components of the baseband signal respectively and f_{E5} is the central frequency. The baseband signal $s_{E5}(t)$ is defined as

$$s_{E5}(t) = s_{E5-I}(t) + js_{E5-Q}(t). \tag{7-8}$$

The PRN code and data are combined to binary NRZ modulated functions $e_{E5a\text{-}I}$, $e_{E5b\text{-}I}$, $e_{E5a\text{-}Q}$, and $e_{E5b\text{-}Q}$. Definition of these functions is given by

$$\begin{aligned}
e_{E5a-I}(t) &= \sum_{i=-\infty}^{+\infty}\left[c_{E5a-I,|i|_{LE5a-I}}d_{E5a-I,|i|_{DCE5a-I}}rect_{T_{C,E5a-I}}(t - iT_{C,E5a-I})\right],\\
e_{E5a-Q}(t) &= \textstyle\sum_{i=-\infty}^{+\infty}\left[c_{E5a-Q,|i|_{LE5a-Q}}rect_{T_{C,E5a-Q}}(t - iT_{C,E5a-Q})\right],\\
e_{E5b-I}(t) &= \sum_{i=-\infty}^{+\infty}\left[c_{E5b-I,|i|_{LE5b-I}}d_{E5b-I,|i|_{DCE5b-I}}rect_{T_{C,E5b-I}}\left(t - iT_{C,E5b-I}\right)\right],\\
e_{E5b-Q}(t) &= \sum_{i=-\infty}^{+\infty}\left[c_{E5b-Q,|i|_{LE5b-Q}}rect_{T_{C,E5b-Q}}(t - iT_{C,E5b-Q})\right].
\end{aligned} \tag{7-9}$$

Let *Y-Z* denote the signal component a-I, a-Q, b-I and b-Q respectively. Then in (7-9), symbol $C_{E5Y\text{-}Z}$ denotes the ranging PRN code, $d_{E5Y\text{-}Z}$ denotes the data message and $T_{C,\,E5Y\text{-}Z}$ denotes the chip length of $1 / (1.23 \times 10^6)$ in seconds.

Ranging code $C_{E5Y\text{-}Z}$ is a layered code composed of primary code with length of 10,230 chips and secondary code with length of 20, 100, 4, 100 chips for E5a-I, E5a-Q, E5b-I and E5b-Q signal components respectively. Both primary and secondary code are both defined in 0, 1 format in the ICD. They are combined using XOR operation and translated with 0, 1 = +1.0, -1.0 NRZ modulation to the final $C_{E5Y\text{-}Z}$ ranging code. In this work, $CP_{E5Y\text{-}Z}(t)$ denotes the primary code sequence and $CS_{E5Y\text{-}Z}(t)$ denotes the secondary code sequence for the respective signal component.

The AltBOC modulation is given in the ICD using two alternative definitions. The first definition is analytical, the $s_{E5}(t)$ baseband modulation function is expressed as combination of two subcarrier functions $sc_{E5\text{-}S}$ and $sc_{E5\text{-}P}$. The second definition expresses $s_{E5}(t)$ as a 8-PSK modulation given by

$$s_{E5}(t) = exp\left(j\frac{\pi}{4}k(t)\right) = \cos\left(\frac{\pi}{4}k(t)\right) + \mathrm{j}\sin\left(\frac{\pi}{4}k(t)\right), k(t) \in \{1, \ldots, 8\}. \tag{7-10}$$

The value of function $k(t)$ is given by the value of the quadruple $(e_{E5aI}, e_{E5bI}, e_{E5aQ}, e_{E5bQ})$ and by the value of the variable i_{Ts}. This variable expresses the time t so that it is partitioned into equal intervals

of length $T_{s,E5}$ and then sub-divided in 8 equal sub-periods. The the index i_{Ts} of the actual sub-period is given by [63]

$$i_{Ts}(t) = integer\ part\left[\frac{8}{T_{s,E5}}\left(t\ modulo\ T_{s,E5}\right)\right], i_{Ts}(t) \in \{0, \ldots, 7\} \quad \textbf{(7-11)}$$

The relationship between the sub-carrier interval $T_{s,E5}$, the chip length $T_{C,E5}$ and the index i_{Ts} is visualized in Figure 7-14. The values of $k(t)$ are indicated for the case when quadruple ($e_{E5a\text{-}I}$, $e_{E5b\text{-}I}$, $e_{E5a\text{-}Q}$, $e_{E5b\text{-}Q}$) = (-1,-1,-1,-1). The values of the discrete function $k(t)$ and are given in ICD by a matrix with dimensions 8 × 16. This matrix will be called **k-lookup** table further on.

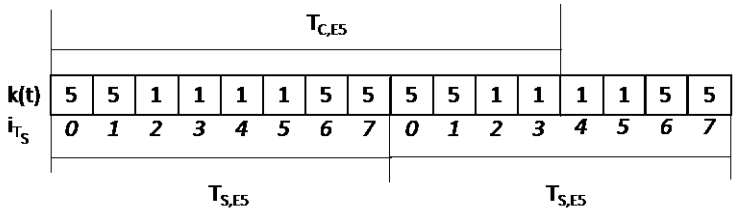

Figure 7-14 Relationship between E5 sub-carrier interval period and chip length

Closer examination of the k-lookup reveals an interesting relationship. When quadruple ($e_{E5a\text{-}I}$, $e_{E5b\text{-}I}$, $e_{E5a\text{-}Q}$, $e_{E5b\text{-}Q}$) with NRZ level values +1.0,-1.0 is translated to 1, 0 format, it can be interpreted as a digital number with four digits. The order of these digital quadruple values is equivalent to digitally expressed zero-based order number of the quadruple seen as index to k-lookup. For better explanation, let us take as an example the quadruple (-1, -1, -1, +1). The translation according to the given pattern evaluates to (0, 0, 0,1), which can be understood as the binary expressed number 1 (i.e. 0001). In this work, this kind of representation of a quadruple of binary values related to E5a-I, E5b-I, E5a-Q and E5b-Q signal components is called **bit-combined format**.

The procedure of processing of PRN codes and symbols for the E5 modulation as given in the ICD can be expressed using the bit-combined format. The schema of the procedure is depicted in Figure 7-15. The pair X-Y denotes the components E5a-I, E5a-Q, E5b-I and E5b-Q. The quadruple ($e_{E5a\text{-}I}$, $e_{E5b\text{-}I}$, $e_{E5a\text{-}Q}$, $e_{E5b\text{-}Q}$) = 0, 1, 1, 0 serves as an example in the figure. Primary and secondary codes are combined using XOR operation and then converted to NRZ format. Code and data symbols are combined to $e_{X\text{-}Y}$ quadruples. These can be then interpreted as binary values and combined to binary numbers. The quadruple 0, 1, 1, 0 evaluates to the zero based number 9 that can be used as index to the k-lookup table.

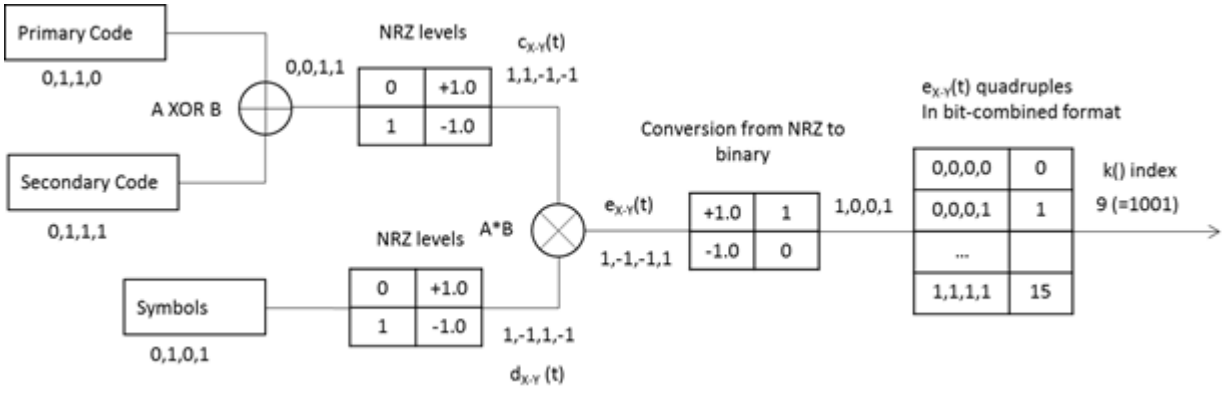

Figure 7-15 Processing of E5 PRN codes and modulation data as defined in Galileo ICD

7.4.1 Concept for PRN Sequences and Data Bits

The algorithm for the parallelized generation of E5 signal was designed. The algorithm follows the common concepts introduced in the previous sections, with the exception of the data format of the PRN chips and application of the bit and secondary chip. The design of this part of the algorithm is given in this section.

The procedure for representation and processing of PRN sequences and data bits in the E5 generation is the key part of the whole algorithm. It enables the comparatively high generation speed of four components of E5 signal in contrast to E1 OS and L1 C/A. The concept of the data processing is depicted in Figure 7-16. It differs from the procedure described in ICD and depicted in Figure 7-15 significantly. The example of the quadruple ($e_{E5a\text{-}I}$, $e_{E5b\text{-}I}$, $e_{E5a\text{-}Q}$, $e_{E5b\text{-}Q}$) = 0, 1, 1, 0, identical to Figure 7-15, is used in Figure 7-16. It demonstrates that both procedures result in the same index value.

The concept of the data processing depicted in Figure 7-16 processes the chips of all four primary PRN sequences as well as the chips of secondary PRN sequences and data bits in the bit-combined format. The combination of primary chip, secondary chip and data symbol to $e_{X\text{-}Y}$ is done in following steps.

First, the secondary code and symbols are expressed in NRZ format and then multiplied to a combined value. Then they are converted from 1/-1 to 1/0 format and transferred to the GPU as a part of the fixed parameters. Then, the kernel running on the GPU loads the PRN parts of primary code in bit combined format from the GPU main memory to the shared memory. Within this operation, the PRN parts are multiplied with this combined value of secondary code and data symbol to retrieve the $e_{X\text{-}Y}$ quadruple. When the end of each secondary chip is reached (this event is called **secondary chip flip** further on), the actual secondary chip and symbol combination is subtracted from the $e_{X\text{-}Y}$ value. In this way, the primary PRN code is extracted and it is then combined with the following secondary chip and data symbol combination.

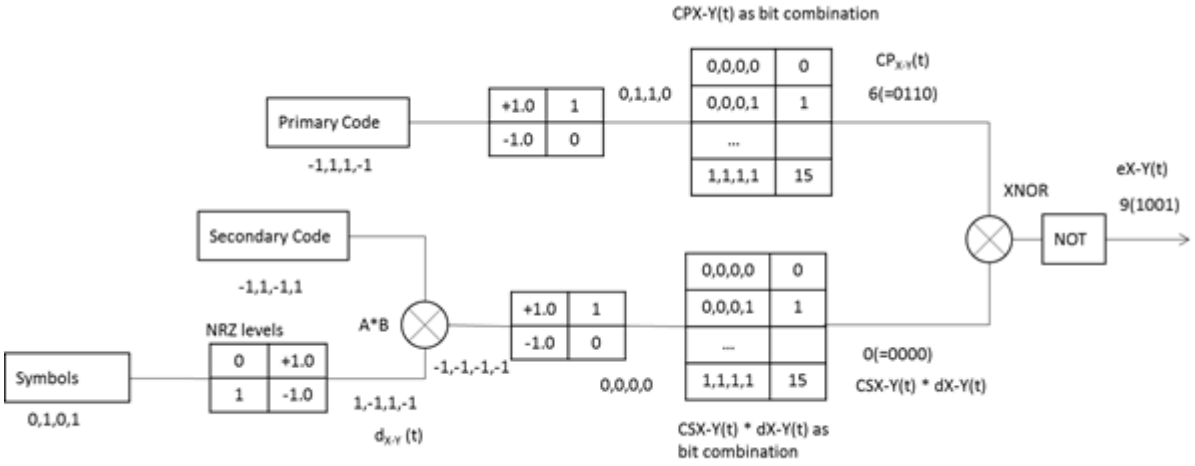

Figure 7-16 Algorithm of processing of E5 PRN codes and modulation data for parallelized operation

For the computation of the individual signal samples, this $e_{X\text{-}Y}$ value is used as the index to the k-lookup table to generate the AltBOC modulation value. The algorithm for the modulation is described in detail in the next section.

This concept is implemented so that the primary PRN sequences for E5a-I, E5b-I, E5a-Q and E5b-Q signal components are hold in a single vector of unsigned integer numbers with 10,230 items. In *n*-th integer number of the vector, the last 4 bits represent in 0/1 format the *n*-th chip of each of the four sequences. When the most significant bit is numbered by 31 and the least significant bit is numbered by 0, then the four last bits express the chips of the PRN codes of the signal components as follows: bit 28 - E5a-I, bit 29 - E5b-I, bit 30 - E5a-Q and bit 31 - E5b-Q.

The application of secondary chip and data symbol is defined in the ICD as multiplication of numbers expressed in -1/1 representation. This operation is equivalent to the logical operation NXOR applied to numbers in 0/1 representation. NXOR stands for the logical complement of the exclusive OR operation, also called XNOR is the literature. For the representation of the quadruple ($e_{E5a\text{-}I}$, $e_{E5b\text{-}I}$, $e_{E5a\text{-}Q}$, $e_{E5b\text{-}Q}$), the 0/1 representation is very effective, because the quadruple can be stored in integer number as last four bits in 0/1 format and can be used directly as the index into the k-lookup table.

The NXOR operation on the bit-combined format of variables was implemented using logical bitwise operations. These operations execute a logical operation on each bit of the variable individually and they are compiled in many cases to a single instruction. In C/C++ and in CUDA C, the bitwise XOR is denoted as "^" and bitwise NOT in is denoted as "~". For integer variables *A* and *B,* interpreted as vectors of *n* bits with last *m* bits used for logical values, the following equality holds:

$$\begin{gathered}\text{For each } (i > n - m)\text{: } A[i] \text{ NXOR } B[i] \\ \text{is equal to} \\ \texttt{~ (A ^ B) - X,} \text{ where } X[0], .., X[n - m] = 1 \text{ and } X[n - m + 1], \ldots., \mathrm{X}[n - 1] = 0.\end{gathered} \tag{7-12}$$

Based on this equation, the combination of PRN sequences loaded from the GPU main memory to the shared memory with the secondary chip and data bit quadruples can be calculated as

```
ps_e_PrnPartBitCombi[I] = (~(nSecChipsAndSymbolsBitCombi ^
pdPrnPartBitCombi[I])) - LEADING_28_ONES_n;
```

where `pdPrnPartBitCombi` is a vector of PRN parts in bit-combined format, where in each integer variable the last four bits are dedicated to chips of E5a-I, E5b-I, E5a-Q and E5b-Q signal components respectively. `nSecChipsAndSymbolsBitCombi` is the combined secondary chip and data symbol in bit-combined format. `LEADING_28_ONES_n` stands for a number with 28 leading ones and 4 zeros equivalent to variable *X* in (7-12).

When the end of a secondary chip is reached, the previous secondary chip and data bit combination is subtracted from the PRN sequences and the following one is applied. This operation can be expressed as

```
ps_e_PrnPartBitCombi[I] = ps_e_PrnPartBitCombi[I] NXOR nSecChipsAndSymbolsBitCombi
NXOR nSecChipsAndSymbolsBitCombiNext
```

where the respective variables are defined as before. Variable `nSecChipsAndSymbolsBitCombiNext` stands for the following secondary chip and data symbol combination of the four signal components in bit-combined format.

The *A* NXOR *B* NXOR *C* operation is equal to *A* XOR *B* XOR *C* operation, where leading zeros evaluate to zero (0 XOR 0 XOR 0 = 0). Therefore, the operation can be expressed in a simplified way using bitwise C/C++ operators as

```
ps_e_PrnPartBitCombi[I] = ps_e_PrnPartBitCombi[I] ^ nSecChipsAndSymbolsBitCombi ^
nSecChipsAndSymbolsBitCombiNext
```

This expression is then applied in the E5 generation algorithm.

7.4.2 Implementation of AltBOC Modulation

For the parallel signal generation on GPU, the AltBOC modulation definition (7-11) with 8-PSK modulation and lookup table *k(t)* was chosen. The reason is that the k-lookup table is small enough to fit into the shared memory where a value can be retrieved in a single operation by a whole warp in parallel. The total number of operations to retrieve the *k(t)* value is dependent on the concept of the storage of the table in the shared memory.

7.4.2.1 Memory Concept for Lookup Table *k(t)*

The k-lookup comprises 128 items ordered as matrix with 8 rows and 16 columns. It is indexed with i_{Ts} index and $e_{X\text{-}Y}$ quadruple. The shared memory is ordered to 32 memory banks, where in one step just one number from each bank can be retrieved by a warp. Each warp generates 32 successive samples of a signal channel. The number of samples per sub-period of a sub-carrier interval depends on the sample rate. As visualized in Figure 7-14, there are twelve sub-periods in one chip and eight sub-period in a sub-carrier interval. During one chip period, the $e_{X\text{-}Y}$ value is constant and the twelve subcarrier values with i_{Ts} = 0, …, 7, 0, …, 3 or i_{Ts} = 4, …, 7, 0, …, 7 are successively retrieved from a single column of the k-lookup table.

This sub-carrier modulation pattern implies two possible concepts for storage of k-lookup data in the shared memory. The first is to prolong of the k-lookup table to 12 rows by repetition of rows with i_{Ts} = 0, …, 4, storing two columns of the table successively 1 one bank column of the shared memory. Eight bank columns are then needed for the storage of the whole table. The second concept is to store the k-lookup in a straightforward manner with four k-lookup table columns on one bank column of the shared memory. The table occupied then four bank columns of the memory.

For both concepts, the resulting access pattern was analyzed. The access of one warp of threads to the k-lookup stored in the straightforward manner is visualized in Figure 7-17 for low SR. The bank conflict can occur only when the threads at the end of a warp start generation of the next chip, and when the k-lookup column of the $e_{X\text{-}Y}$ quadruple in the following chip period is placed on the same memory banks. Supposing uniform distribution of $e_{X\text{-}Y}$ values, probability of the k-lookup columns of the $e_{X\text{-}Y}$ quadruple of the successive chips being placed on the same banks is 25%.

The second criterion for bank conflict to occur is that i_{Ts} index to the table that is used by the first threads of the warp must be reached by the last threads of warp. The bank conflict can therefore occur, only when thread 0 and thread 31 are less than 8 sub-periods far apart. In other words, when the sample rate is lower than four times sub-period rate ($32/8 \times 12 \times f_C$ = 491.04 Msps).

In case of k-lookup being stored in the prolonged manner, the bank conflict can occur only when thread 0 and thread 31 are less than 12 sub-periods far away, in other words when the sample rate is lower than four times sub-period rate ($32/12 \times 12 \times f_C$ = 327.36 Msps). Probability of $e_{X\text{-}Y}$ quadruples of following chip periods being stored on the same banks is 50%. For the Boadband configuration with SR = 1.4 Gsps, both concepts are free of bank conflicts. The concept of straightforward storage was chosen as it consumes less amount of shared memory.

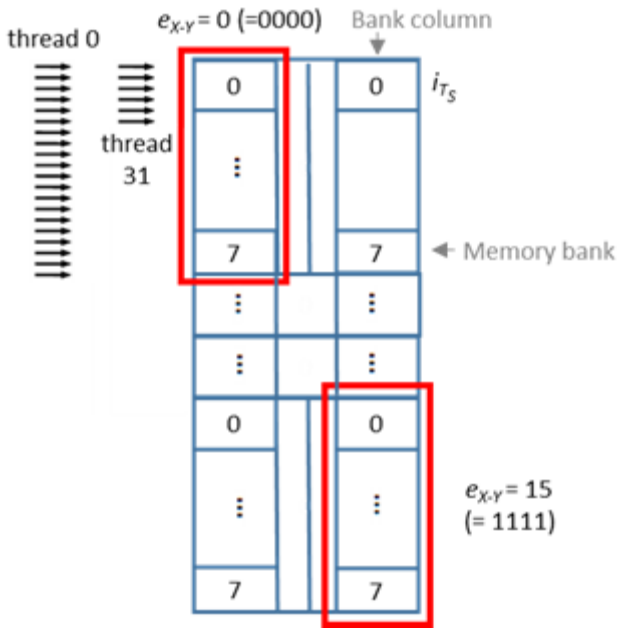

Figure 7-17 The straightforward concept of storage of k-lookup in shared memory

7.4.2.2 Concept for Computation of Sub-Carrier Phase

The sub-carrier phase i_{Ts} is defined by (7-11) and it expresses time t partitioned into equal intervals $T_{s,E5}$ and then sub-divided to sub-periods. In the kernel, the time epoch is input in fixed parameters in form of the code phase from the beginning of the actual data bit. In the inner loop of the signal generation algorithm that generates samples of each PRN part, only the code phase within the primary code sequence is available. The i_{Ts} sub-period starts with 0 at each even code chip, as visualized in Figure 7-14. Therefore, for the even number of chips in primary code sequence as is the case for E5 (10,230 chips), the code phase within primary code sequence is a sufficient base for computation of sub-carrier phase i_{Ts}. The sub-carrier phase can be retrieved as

$$i_{Ts}(t) = (CodePh(t) * 12) mod\ 8, CodePh(t) \in [0,10230), \quad \textbf{(7-13)}$$

where *CodePh* is primary code phase ranging from 0 to 10,230. The code phase variable in kernel is a 64-bit long integer variable in fixed-point format with 16 integer and 48 fractional bits. To keep the coherence between code phase and sub-carrier phase, the subcarrier phase must have the same bit resolution as the code phase. Equivalent data type, the 64-bit long integer in fixed-point format with 16 integer and 48 fractional bits was chosen for the subcarrier phase.

The *mod* operation is available in CUDA C, but its instruction throughput for 64-bit numbers is not published in the documentation. Probably, it is compiled to multiple instructions. Therefore, it is not possible to compare it with other implementations of the sub-carrier phase computation on theoretical basis, only an execution measurement could evaluate the performance. , but the operation can be in case of *mod* 2^x on unsigned integer numbers implemented as two bit-shift operations.

An alternative computation of *mod* operation is possible, when the second operand is equal to 2^x for an integer x. Then, the operation can be executed as two bit-shift operations on unsigned integer numbers. The sub-carrier phase can be then computed from code phase as

```
iTS=(int)((ulCodePhase * 12) << (64-CODE_PRECISION-3) >> (64-3)),
```

where `iTS` is the sub-carrier phase, `ulCodePhase` is the code phase in chips and `CODE_PRECISION` is the number of fractional bits of both phase variables. The algorithm comprises following operations: conversion from double to integer, 64-bit multiply and two 64-bit bit shifts. Note that for code phase > 5,461 chips, `ulCodePhase` $\times$ $12 > 2^{16}$ and therefore an overflow of the 64-bit fixed-point number with 16 integer bits occur. The overflow in the multiplication causes an implicit *mod* 2^{16} operation. Because the *mod* 8 operation follows, no information is lost by the overflow.

Another way to retrieve the sub-carrier phase from code phase is to use a designed sub-carrier NCO. The variable `ulSubcarrierPh` holds the sub-carrier phase and increases with each sample by `ulSubcarrierPhDelta`. The data type of `ulSubcarrierPh` can be set to unsigned 64-bit long integer in fixed-point format with 3 integer and 61 fractional bits. This representation causes an overflow at value of $8 = 2^3$ which is equivalent to *mod* 8 operation. Using this solution, i_{Ts} can be computed as follows:

```
iTS=(int)(ulSubcarrierPh>>(64-3));
ulSubcarrierPh+= ulSubcarrierPhDelta;
```

This implementation needs following operations: 64-bit add, conversion from double to integer and a single 64-bit bit shift. This solution uses the same operations as the previous one minus one 64-bit bit shift. Therefore, this solution was chosen for the evaluation of i_{Ts} in the inner loop of the E5 generation algorithm.

7.4.2.3 Instruction Throughput for Carrier Wave and 8-PSK Computation

For the evaluation of equation (7-10), an effective implementation of the *sin* and *cos* functions for AltBOC 8-PSK modulation is an important issue. Similarly, the evaluation of the carrier wave of the signal S_{E5} as defined in (7-7) needs to be effectively implemented. This issue was already discussed in section 7.2 GPS L1 C/A Generation. It was explained that usage of the SFU functions is more effective than usage of traditional lookup tables. For the special case of AltBOC 8-PSK modulation function, where sine and cosine functions use the same argument, the CUDA specific combined SFU function `__sincosf` was used. It computes both values in a single clock cycle on all available SFUs (32 SFUs per multiprocessor on Kepler architecture). The usage of a lookup table would be an alternative that is slower but still applicable. There are only eight modulation states represented in table *k(t)* and therefore the lookup table would be short enough to fit into the shared memory.

The final E5 signal generation algorithm is given in CUDA C as

```
__sincosf(__uint2float_rn(unCarrierPhasePerThread) * (PIx2div2upto32), &fCarrSin,
&fCarrCos);

__sincosf(psK_lookup_xPiDiv4 [8*
ps_e_PrnPartBitCombi[(int)(ulCodePhaseInPrnPartPerThread>>CODE_PHASE_RESOLUTION)]+
(int) (ulSubcarrPhasePerThread>>(64-3))], &fKSin, &fKCos);

atomicAdd((psSubbatch+nSubsample),(fSigAmpFactor*(fCarrCos*fKCos – fCarrSin*
fKSin));

ulCodePhaseInPrnPartPerThread+=ulCodePhaseDeltaPerThread;
unCarrierPhasePerThread+=unCarrierPhaseDeltaPerThread;
ulSubcarrPhasePerThread+=ulSubcarrPhaseDeltaPerThread;
```

In all common parts, the variables and concepts are identical to L1 C/A and E1 OS generation algorithms given in the previous sections. At the beginning of the E5 sample computation, the amplitude of the carrier wave is calculated for in-phase components as `fCarrCos` and for quadrature components as `fCarrSin`. Then the AltBOC modulation value is computed. The k-lookup table is placed in the memory as vector `psK_lookup_xPiDiv4` with 128 values, where the modulation states were already converted to phase value by $\times\ \pi/4$ operation. The combined primary code, secondary code and data symbol quadruples are retrieved in bit-combined format from vector `ps_e_PrnPartBitCombi` placed in the shared memory. This combined quadruple serves as index to the k-lookup vector. The second part of the index is retrieved from the subcarrier phase accumulator `ulSubcarrPhasePerThread` by mod 8 operation. Sine and cosine values of the modulation states are then combined with carrier wave to the final signal sample and added to the shared memory by an atomic operation. Code, carrier and subcarrier NCO compute the next phase value, in case of carrier and subcarrier, this step is accompanied by automatic *mod* 1 and *mod* 8 operation respectively.

7.4.3 Performance and Verification

The performance of E5 signal generation algorithm was measured on the Test System with the benchmarking configuration of IF = 30.69 MHz and SR = 130 Msps. The generation speed in Msps for 2-12 satellite channels is given in Figure 7-18. For 12 satellite channels, the generation speed of 1,630 Msps was reached with a single GPU. Using the Broadband configuration of the Test System, the speed of 1,892 Msps was reached. The higher chipping rate of E5 in contrast to E1 OS and L1 C/A is probably the reason, why in case of E5 the signal generation with Broadband configuration, significantly higher generation speed for high number of channels is measured than with the benchmarking configuration.

The number of threads per channel as given in Table 7-1was employed for the generation. The convenient numbers of threads in case of 10, 8 and 4 satellites cause again a remarkable increase in generation speed. For a low number of satellites, the speed of E1 OS and E5ab is very similar showing the influence of the tasks independent of number of channels.

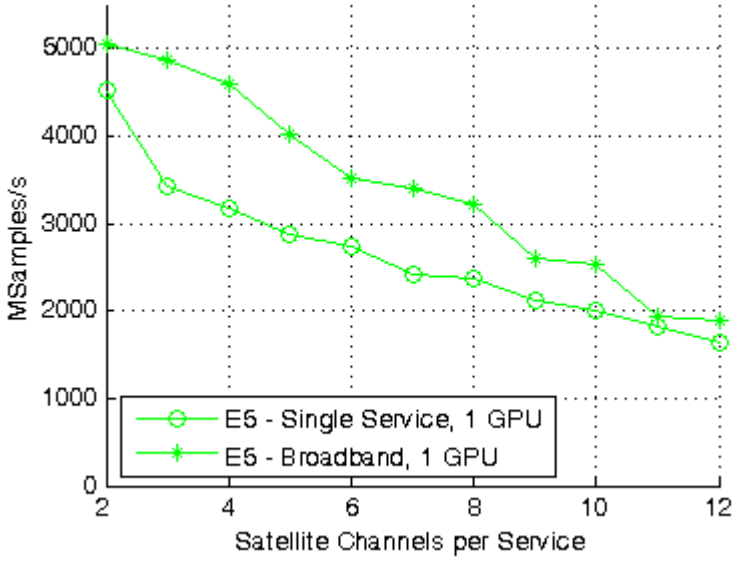

Figure 7-18 Signal generation performance of E5ab kernel using 1 GPU

Comparison of the performance of E5ab generation to performance of E1 OS generation (Figure 7-11) and performance of L1 C/A generation (Figure 7-7) show interesting results. Considering Broadband configuration and 12 satellite channels, the L1 C/A signal with one signal component is generated with the speed of 2,970 Msps, the E1 OS with two components is generated with the speed of 1,769 Msps and E5 with four components is generated with the speed of 1,892 Msps. It shows, that the bit-combined format and related algorithms granted the E5 generation an important speed up in contrast to the separated application of data and pilot codes together with the retrieval of subcarrier from shared memory, as it is the case for E1 OS. For the E1 OS modulation, these optimization strategies cannot be applied. This shows that E5 signal modulation features are more convenient for parallelized generation on a GPU.

The generated E5 signal was verified using GNSS receiver SX-NSR by IFEN GmbH. The receiver processes the E5a and E5b signal services separately with 10.24 MHz bandwidth each. The response of the receiver to the signal generated with the Test System was compared to the response to the signal generated by GNSS signal simulator NAV-NCS.

Four scenarios were tested and the settings are summarized in Table 7-4. In the NAV-NCS scenario, a full constellation with 12 satellites was simulated as RF and received simultaneously by E5a (scenario no. 1) and E5b (scenario no. 2) receiver modules. In the verification scenarios no. 3 and no. 4, one satellite channel with IF = 51.15 MHz was generated as DIF signal. As no noise was added and the the receiver measured C/N_0 = 75.51 dB/Hz. The receiver modules E5a and E5b are used with identical settings for all scenarios.

No.	Name	# sats	IF [MHz]	C/N_0 [dB-Hz]	T_{int} [s]
1	NAV-NCS E5a	12	5.529	51.15	0.010
2	NAV-NCS E5b	12	15.539	51.15	0.004
3	E5a228.56MSs	1	35.805	75.71	0.010
4	E5b228.56MSs	1	66.495	75.71	0.004

Table 7-4 E5 verification scenarios

The power spectrum density of the signal from scenario no. 3 is given in Figure 7-19 as measured by the receiver SX-NSR. The acquisition output for the E5a signal with zero Doppler frequency and zero code phase is given in Figure 7-20. The output of the tracking loops was closely observed and the RMS of the tracking loop error is given in Table 7-5. Additionally, the thermal loop error for the receiver settings was compared with the measured tracking loop error.

No.	σ_{iDLL}	σ_{DLL}	Ratio DLL	σ_{iPLL}	σ_{PLL}	Ratio PLL	σ_{iFLL}	σ_{FLL}	Ratio FLL
Unit	**[chip]**	**[chip]**		**[cycle]**	**[cycle]**		**[Hz]**	**[Hz]**	
1	0.00139	0.03376	24.4	0.00171	0.00422	2.8	0.34177	1.43850	4.2
2	0.00139	0.05773	41.6	0.00171	0.00445	2.6	0.34177	6.44617	18.9
3	0.00008	0.0459	1635.3	0.00010	0.00032	3.2	0.0202	0.1881	9.3
4	0.00008	0.1334	1628.2	0.00010	0.00111	11.0	0.0202	1.5542	76.9

Table 7-5 Theoretical thermal error RMS (σ_i) and measured RMS error of the tracking loops (σ)

The results in the table show, that the thermal error decreases significantly for the high C/N_0 of the verification scenario. The measured tracking loop error decreases only slightly. The reason is probably the limited precision of the receiver tracking loops. The SX-NSR is a CPU-based software receiver that probably generates the signal replica using a lookup table. The precision of the signal replica of the receiver in not published, but considering a lookup table with 256 items, the carrier phase precision would be limited to 0.00195 cycle. The DLL results are little worse for verification scenario, PLL and FLL results are slightly better. The generally worse results for E5b can be observed for both NAV-NCS and verification scenarios.

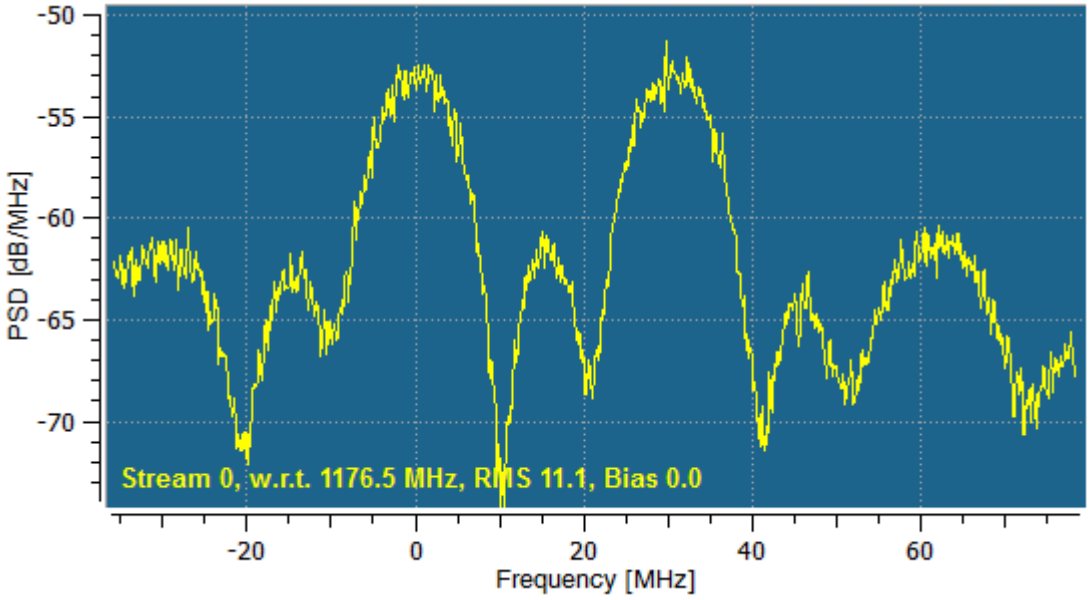

Figure 7-19 Power spectrum density of E5ab signal with respect to E5a frequency

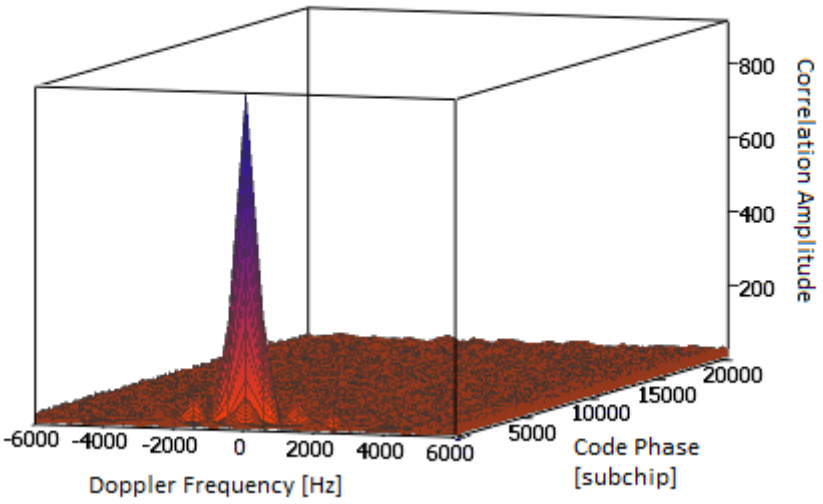

Figure 7-20 Acquisition plot of the E5a signal

7.5 Parallelization - Data Transfer, Multiple Services and GPUs

7.5.1 Data Transfer between Host and GPU Memory

From host computer memory to the GPU main memory, the signal parameters for each microepoch, called fixed parameters are transferred. From the GPU back to the host memory, the generated samples are copied. The data volume of the fixed parameters for one microepoch is very low in comparison to the volume of the generated signal samples.

The fixed parameters for one microepoch are listed in Figure 5-3. They comprise for each satellite channel of each service a data volume of 32 bytes. For three services with 12 satellite channels each, the fixed parameters comprise a data volume of merely 1,152 bytes. It contrast to777 it, the batch of samples of one microepoch comprises a data volume of 204,800 bytes.

With the older PCIe v. 2.0 x16 interface between GPU and host computer, the transfer of the generated samples was the bottleneck and upper limit for the whole GPU-based signal generation performance. In the Test System, a GPU with PCIe v. 3.0 x16 is deployed. The data transfer speed of 12.82 GB/s was measured. Even for the Broadband configuration with SR of 1.4 Gsps, i.e. 2.8 GB/s, this transfer speed is sufficient.

The GPU architecture enables to run computations on GPU in parallel to the data transfer to host as well as in parallel to computations on the host computer. The transfer of signal samples from GPU to host computer can therefore run in parallel to the execution of the signal generation kernels on the GPU. According to the CUDA documentation, fully parallel run should be possible.

The asynchronous concurrent execution between host computation, data transfer and GPU computation is enabled by the CUDA concept of so-called streams. A stream is a sequence of commands that execute in order. All host functions invocations and runs, all data transfer invocations and all device functions invocations are run in so-called default stream. When additional streams are specified, the GPU function runs and memory copies can be executed in these specific streams in parallel to other streams. In each stream, functions and data transfers run synchronously, which means that they are executed in order. If necessary, the streams must be explicitly synchronized. For concurrent run of kernels and memory copies, several rules restricting the parallelization apply [69].

Considering these features, an algorithm to parallelize the execution of a kernel with the memory copy of signal samples and simultaneously with the host operation (fixed parameters computation and further processing of samples) was designed. The algorithm is designed for a variable number of streams. Each stream operates on separated data structures to avoid data access conflicts. In each stream, each kernel is executed is on all multiprocessors, by one or two block of threads on each, generating on batch of samples respectively. This concept of parallelization oven multiple multiprocessors was explained in section 7.1.1. The batches of samples computed by one kernel in one stream are called a **batch set**. In the following text, the number of blocks of threads is referred as `nBlocks`. The data structures playing role in the parallelization algorithm are:

`nBlocks` – the kernel of each service is launched in `nBlock` blocks of threads. Each block is executed on a single multiprocessor.

`nStreams` – number of CUDA streams

`pdBatchSet[nStreams]` – a vector in GPU main memory containing `nStreams` batch sets, each batch set includes `nBlocks` batches

`phBatchSet[nStreams]` – the same as previous, but placed in the host memory

`pdFixedParsSet[nStreams]` – a vector in device memory containing `nStreams × nBlocks` fixed parameters sets

`phFixedParsSet[nStreams]` – the same as previous, but place in the host memory

`Streams[nStreams]` - a vector of CUDA streams

In each stream, at least one explicit synchronization operation must be included to synchronize CPU and GPU computation. The computation and copy of the next set of fixed parameters on the CPU must wait, until the previous set is processed. The processing of a generated batch set of samples must wait, until the generation and copy to CPU is finished.

At first, Single Service configuration was considered for the parallelization. The algorithm for parallelized signal generation and data transfer is given in CUDA C as follows:

```
nRuns = nBatchesToGenerate/(nStreams*nBatchesInSet) + 1;

for (I=0;I<nRuns;I++)
{
  if (I<nRuns-1)
    for (J=0;J<nStreams;J++)
      computeFixedParsSet(phFixedParsSet[J],&nVisSats);

  for (J=0;J<nStreams;J++)
  {
    if (I<nRuns-1)
      cudaMemcpyAsync(pdFixedParsSet[J],phFixedParsSet[J],Streams[J]);
    if (I>0)
      cudaMemcpyAsync(phBatchSet[J],pdBatchSet[J],Streams[J])
  }

  if (I<nRuns-1)
    for (J=0;J<nStreams;J++)
     {
       ThreadsPerBlock[J]=nVisSats*THREADS_PER_SAT;
       syncStream(Streams[J]);
       kernelL1Ca<<<nBlocks,nThreadsPerBlock[J],Streams[J]>>>
             (pdBatchSet[J],pdFixedParsSet[J],pdPrnSeqsAll);
     }
   else
      for (J=0;J<nStreams;J++)
         syncStream(Streams[J]);
   if (I>0)
     for (J=0;J<NUM_STREAMS;J++)
       processBatchSet(phBatchSet[J]);
}
```

In this code, `nRuns` is the number of runs of the main loop of the algorithm. The function `computeFixedParsSet` computes the fixed parameters for one batch set and returns the number of satellite channels in view. The copies of fixed parameters and the generated samples are invocated by function `cudaMemcpyAsync` to a given stream as asynchronous. This means, that the function returns immediately after invocation and does not wait, until the operation is finished. The number of threads in a block is given by the number of satellite channels and a constant parameter `THREADS_PER_SAT` prescribing the number of threads per satellite channel. The synchronization of threads in a stream is invocated by `syncStream` function. The kernel of the signal service to generate `kernelL1Ca` is launched for given number of threads in a given stream and it is always asynchronous. Its parameters are the pointers to already prepared data structures in the GPU main memory. Each generated batch set in CPU memory is then processed by `processBatchSet`. This function is a wrapper for a possible copy to DAC, storage at disc or any other further processing.

An issue of the algorithm design was given by the location of the synchronization point in the loop. Several positions fulfill the objective to synchronize the CPU and GPU operations to get correct results. For a single service and single GPU scenario, two positions were tested using an older GPU (GeForce GT 430, 2 multiprocessors, PCIe v. 2.0. x16). The first was to place the synchronization point just before the kernel invocation. The second was to place it just after the kernel invocation. The first position resulted in overall generation speed that was by 22% faster than a test with the second position. The first position was therefore chosen for implementation.

Furthermore, the placement of the function for processing of the generated batch was examined. |The loop order was nevertheless later changed to optimize the loop for the algorithm for generation of multiple signal services on multiple GPUs.

The performance of the algorithm was examined using the CUDA Visual Profiler. The Profiler measures the run time of the kernel functions, memory copies and streams and visualize the data in a detailed program profile. The profile of the algorithm run for Single Service configuration on the Test System is depicted in Figure 7-21. Each run of the E1 OS kernel is visualized with a cyan beam and labeled as getE1Os, each memory copy between host and GPU is marked brown. The copy of fixed parameters are marked as MemCpy (HtoD) in the profile. The copies of generated samples are labeled as MemCpy (DtoH). The time for copy of fixed parameters set is very short, time to copy the samples much longer. The maximum time consuming operation is to run the kernel to generate the signal. There is a partial overlap between the kernels. The reason is that the start and the end of the run of the individual thread blocks on the individual multiprocessors is sequential. A block of the next kernel can start as soon as any block of the previous kernel has already finished.

The assignment of the kernels and copies to individual streams in visualized in the last four lines of the profile. The test run was launched with four streams labeled as Stream 13, 14, 15 and 16 respectively.

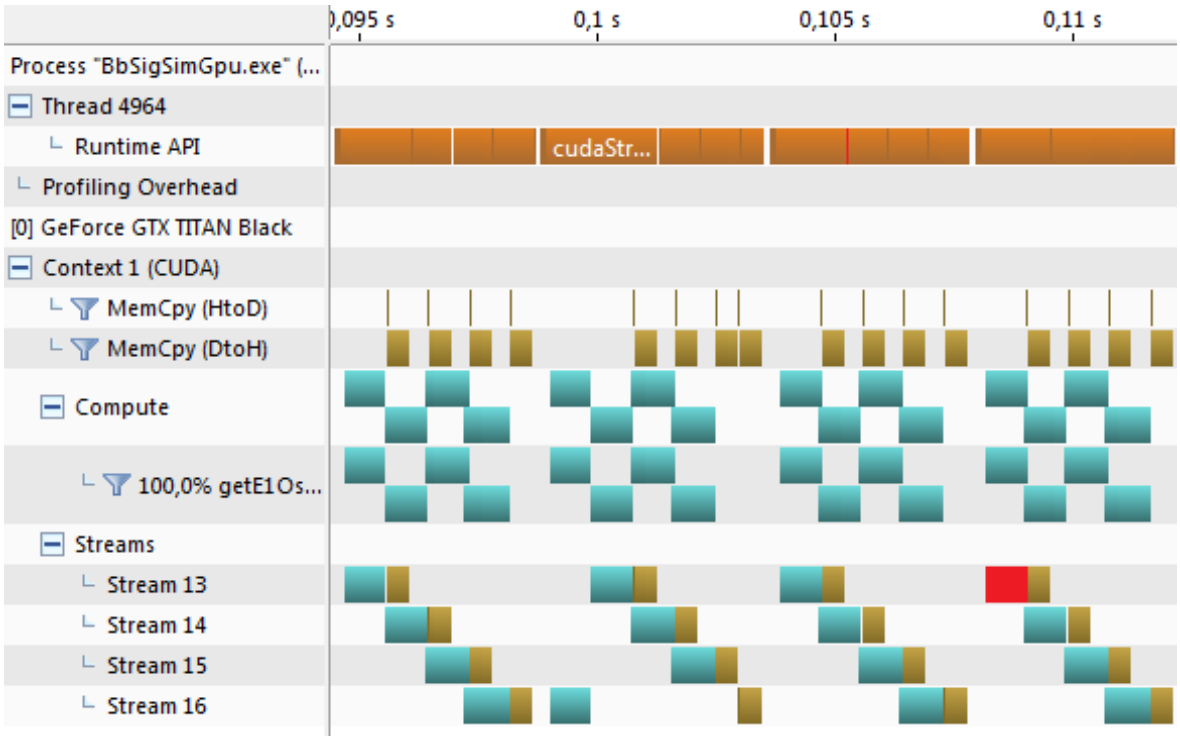

Figure 7-21 Profile of the loop for E1 OS on 1 GPU

In the timeline of the streams, it is visible that the copy of generated samples in one stream is fully overlapped by the execution of the kernel in the following stream. Sometimes, the scheduling of the kernel run on the GPU is done in a different order than as it was invocated. This rare event was captured in the second algorithm run in the profile in Figure 7-21. Even though the memory copy could also directly follow, as it had already been invocated before, the copy is scheduled first after the copies in all the other kernels. This irregularity happens in 1% of algorithm runs, so the influence on the total generation speed is insignificant.

Data transfer and kernel run overlap in all but the last stream. After the copy in the last stream, there is a gap until the computation in the first stream begins. This gap resembles a synchronization of all streams. Nevertheless, the algorithm does not contain any such synchronization point. Each stream is synchronized independently by `syncStream` function. In the profile in the line labeled Runtime API, the invocations of all functions are indicated.

One of the invocations of the kernel in the first stream was marked red and the kernel run was marked in the same way. This example shows that even though the kernel is invocated short after the last copy in the particular stream and the kernel could therefore start just after the kernel in the fourth stream, the start of the execution of the kernel is postponed as if a synchronization of all streams has been commanded. Neither CUDA documentation nor the Kepler architecture documentation deliver an explanation for this behavior. Various loop orders were tested to avoid this implicit synchronization, but without success. Usage of a higher number of streams was tested, to reduce the occurrence of this gap. But with more streams, many more irregularities is scheduling order of the kernel and memory copy runs occur and result in slowdown of overall performance. The best results were reached by the algorithm as described above and with employment of four streams.

7.5.2 Parallel Generation of Multiple GNSS Signal Services

The algorithm was adapted for parallelized generation and addition of multiple signal services. In this adaptation, each signal service is generated by its own kernel. The generated subbatches of samples of each service are added to one common batch in the GPU main memory. The signal services are generated consecutively. One batch set of one signal service at a time is generated by one streams on the GPU. First after kernels of the service are executed in all streams, the generation of the next service begins. The corresponding batches of samples of all signal services are generated in the identical stream, to secure the sequential generation of signal services. In this way, the addition of subbatches to the batch in GPU memory can be done without any atomic operation, which improves the generation speed.

An alternative concept would be to use a common kernel generating all signal services to one common subbatch in the shared memory. Such a solution could generate all services in parallel. However, it would need a more shared memory for the signal modulation data and PRN code than available. The occupation of registers by data of the NCOs of all the signal services would be so high, that the resulting number of threads could not fill the computational capacity of the multiprocessor. Another strategy would be to change the contents of the registers from service to service after each subbatch. This operation would nevertheless consume more time than the load of the samples of other services from the GPU main memory for the addition operation.

The information about the signal service order in the successive generation algorithm is available in the kernel. The first service is therefore assigned to GPU memory without addition. The last service performs additionally the quantization routine. In this way, zeroing of the memory is omitted and performance is improved by ca. 0.1%.

7.5.3 Parallel Generation on Multiple GPUs

On modern PC systems dedicated for high performance computing and gaming, multiple identical graphic cards can be integrated. These GPUs are then connected over a bridge connector. In the context of Nvidia GPUs, this technology is called SLI. The Test System was equipped with two identical GPUs connected in SLI. The signal generation algorithm was adapted for parallelized run on multiple GPUs. The target of the parallelization was to double the performance and design the algorithm to be linearly faster for any further GPUs added to the system.

The generation of a single epoch of the GNSS signal is an independent task. The fixed parameter for the epoch serve as input, the signal samples are returned, and no interdependency with the generation of previous signal epoch exists. The parallelization is therefore based on generation of successive signal epoch by multiple GPUs in parallel. This means, that d-th of D devices in the system generates d-th successive epoch including all signal services. Input data is sent from CPU to each GPU and generated samples are sent back from GPU to CPU are ordered to the final signal stream.

7.5.4 Signal Generation Loop – Multiple Services, Multiple GPUs

The signal generation loop for a single GPU and single service described in section 7.5.1 was redesigned to include the generation of multiple signal services on multiple devices in parallel. The

previous loop accounted for the limitations posed on parallel run in CC 2.1. With CUDA CC 3.5, some of these limitations were abrogated and it was possible to simplify the loop.

In CUDA, the assignment of an operation to a GPU is enabled by the command `cudaSetDevice`. The command selects, on which GPU all following operations should be executed. The switch of GPUs is a costly operation that causes implicit synchronization of all threads.

The algorithm is given in CUDA C as follows:

```
nRuns = nBatchesToGenerate / (NUM_DEVICES *nStreams*nBatchesInSet) + 1

for(I=0;I<nRuns;I++)
for(D=0;D<NUM_DEVICES;D++)
{
  cudaSetDevice(D);
  if(I<nRuns-1)
  {
    for (J=0;J<nStreams;J++)for(K=0;K<nServiceBands;K++)
      computeFixedParsSet(pphFixedParsSet[D][J][K],&nVisSats);
    for (J=0;J<nStreams;J++)
      cudaMemcpyAsync(pphFixedParsSet[D][J],ppdFixedParsSet[D][J],Streams[D][J]);
    for (J=0;J<nStreams;J++)
      syncStream(D,J)
      for(K=0;K<nServiceBands;K++)
  *(pKernel[K]) <<<nBlocks,nThreadsPerBlock[J],Streams[D][J]>>>
          (ppdBatchSet[D][J],ppdFixedParamsSet[D][J][K]);
  }
  else
    for (J=0;J<nStreams;J++) syncStream(D,J);
  if(I>0)
    for (J=0;J<nStreams;J++)
      processBatchSet();
  if(I<nRuns-1)
    for (J=0;J<nStreams;J++)
      cudaMemcpyAsync(ppdBatchSet[D][J],pphBatchSet[D][J],Streams[D][J]);
}
```

In this code, variables and functions are used in the same context as in section 7.5.1. Additionally, `NUM_DEVICES` is the number of GPUs in the system. `pKernel[]` is an array of pointers to the kernels of the individual services. `nServiceBands` is the number of signal services to be generated. The size of all data structures was multiplied by the number of GPUs.

There is a single switch of the GPUs in the loop. It causes implicit synchronization of all threads; therefore, it was possible to remove the synchronization of streams from the loop. Optimal settings of the algorithm were examined for the Broadband configuration. For each GPU, 4 streams and 30 blocks of threads per kernel deliver the best results.

The performance of the algorithm was examined using the CUDA Visual Profiler. The measured values can be imprecise, as the timestamps collection influences the execution of applications running on multiple GPUs in SLI configuration. Nevertheless, the distortion is consistent for each profiling, so the measured profiles can be used to compare different algorithm configurations.

The profile for Broadband configuration of the Test System generating L1 C/A, E1 OS and E5 signal services is depicted in Figure 7-22. It shows the timeline of the computation on the second of the two GPUs used. In the profile, cyan beams mark the E1 kernel, violet mark the E5 kernel and magenta stands for L1 C/A kernel.

The invocation of the GPU switch was colored green in the profile. The gap thereafter indicates the invocations to the next GPU that are not included in this profile. The next series of invocations start with the copy of fixed parameters. One exemplifying invocation and execution of a copy of fixed parameters is marked red in the figure. The GPU switch forces all preceding operations to finish as it causes an implicit synchronization. This synchronization and the synchronization observed in single GPU single service loop described in section 7.5.1 coalesce. Therefore, both the profile in section 7.5.1 and this profile visualize the same pattern of task execution.

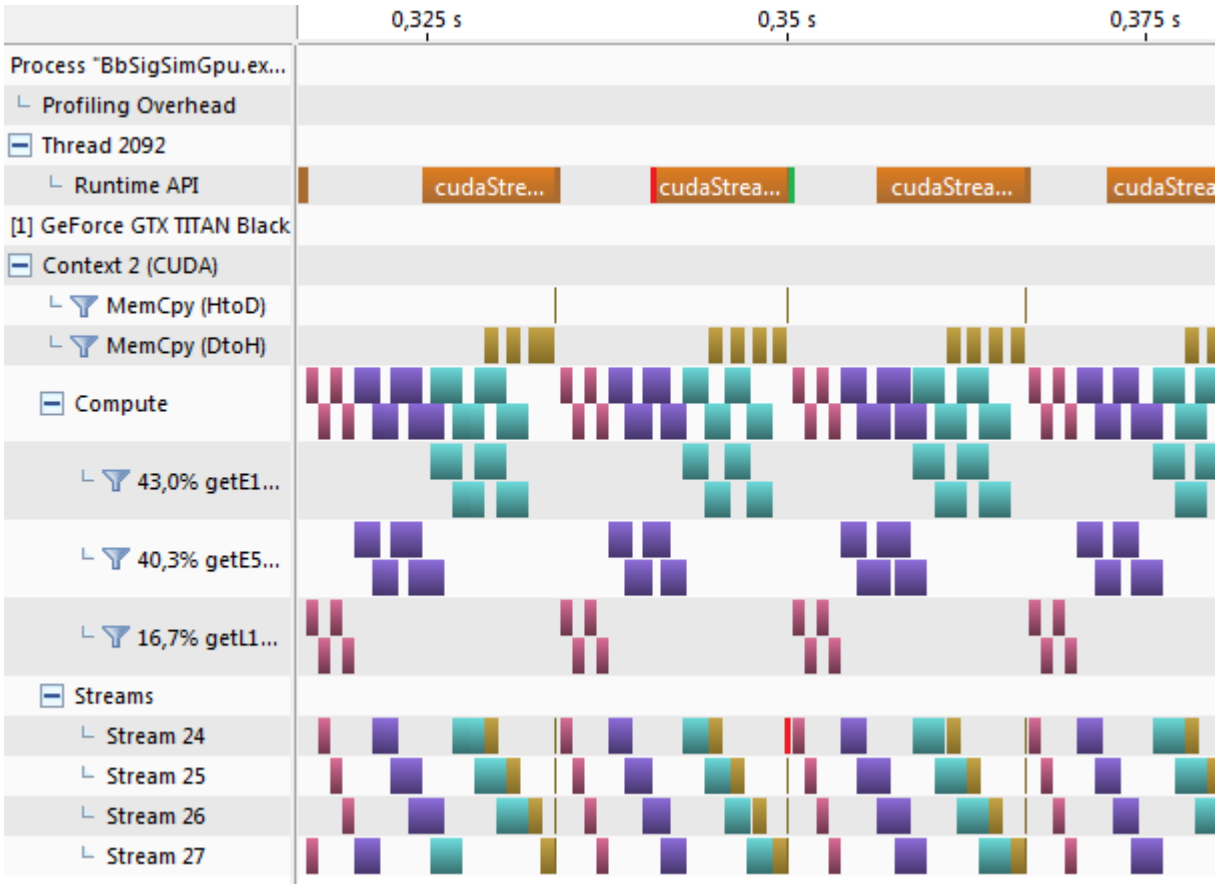

Figure 7-22 Profile of the algorithm for L1 C/A, E1 OS and E5 on the 2nd of 2 GPUs

The sequence of tasks in the two algorithms differs in the position of the copy of samples to the host. In the single GPU loop, the best results are reached when the copy is placed before the following kernel. The placement after kernel gives slightly better results for more GPUs. Again, an irregularity in the scheduling order of the tasks can be observed in the figure.

7.5.5 Performance Measurements

The performance of the multi-GPU multi-service signal generation algorithm was measured on the Test System. For all benchmarking scenarios, the Broadband configuration with settings of 30 blocks of threads, 4 streams and 102,400 samples in batch was used.

The performance of the loop0 for parallel generation of multiple signal services was benchmarked for one and two GPUs in the system. The generation speed for 2 to 12 satellite channels is depicted in Figure 7-23. With 12 channels per service for L1 C/A, E1 OS and E5ab respectively, the speed of 764 Msps was measured on a single GPU.

For each signal service, the number of threads per channel was set as given in Table 7-1. The convenient numbers of threads in case of 10, 8 and 4 satellites cause again a steeper increase in generation speed. For a low number of satellites, the increase in speed is higher than in case of a single service generation. The reason is the threefold computational load in comparison to the time for transfer of samples from GPU to the host computer together with other tasks independent of the number of channels.

The speed of generation of the three services on two GPUs with 12 channels each reached 1,507 Msps. It is twice as much as the speed on one graphic card. With lower number of channels, the increase in speed due to employment of the second GPU decreases. For two satellite channels per service, a speed-up by only 50% can be observed. With so few satellite channels, the computation time decreases below the data transfer time and these two do not fully overlap. With multiple GPUs in system, the computation run in parallel but the data transfer wait are serialized, because they use the shared PCIe backbone. Nevertheless, in case of high computing load due to many satellite channels the algorithm succeeds to linearly increase the performance when the second GPU was added to the system.

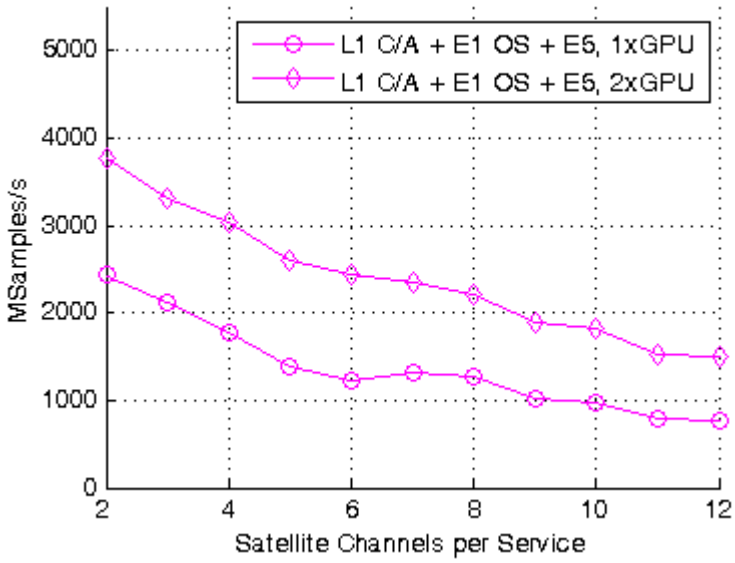

Figure 7-23 Multi-frequency multi-service signal generation performance on 1 and 2 GPUs

8 Digital-to-Analog Conversion and Up-Conversion

For the implementation of the broadband RF GNSS signal simulator architecture a broadband digital-to-analog converter and up-converter modules are crucial. They have a strong impact on the quality of the final output analog signal. The precision of the generated broadband digital signal was analyzed in Chapter 6. The digital signal resolution was outweigh with the dynamic range and other simulation setting ranges as well as with the performance. Consequently, a GPU-based digital signal generation module with multi-GNSS broadband performance was designed and implemented, as described in Chapter 7. The subsequent objective of this work addressed in this chapter is to design compatible digital-to-analog converter and up-converter modules for the broadband architecture. These modules should preserve the performance and output signal quality to the highest possible degree.

The objective of this work is to reach the bandwidth of 427 MHz and real-time capability for the Broadband configuration of the Test System simulator. Research of market-ready and customized components for the digital-to-analog converter was carried out. Chosen narrowband and broadband solutions were implemented in the Test System. The results are given in the following sections.

A research on broadband up-converter module showed that no market-ready solutions are currently available. Nevertheless, the manufacturer National Instruments confirmed feasibility of a development of a tailored solution that would conform to the Broadband configuration specification. The proposed costs of about ten thousand euros would still fulfill the overall target of low-cost architecture, as in a higher volume order the price would decrease significantly. The deployment and benchmarking of an up-converter module is nevertheless out of the scope of this work.

8.1 Broadband Digital-to-Analog Conversion

A research on broadband digital-to-analog converter (DAC) with high conversion rate and high precision per sample was conducted. The DAC chip must be integrated on a platform that features a high-speed interface compatible with the Test System. First, the availability of a high bandwidth DAC chip was examined. As of October 2014, multiple broadband DAC chips were available on the market. The list of the products is summarized in Table 8-1.

In the table, input data rate in Msps stands for the sample rate of the data entering the chip. Majority of the DACs comprise a module that up-samples the input data. The final rate at which the digital signal is converted to analog is called conversion rate. The DACs feature resolution of 14 or 16 bits per sample. This number is sufficient to represent the relative signal levels of GNSS signals channels, as closer explained in Chapter 5. The bandwidth exceeds the target GNSS signal span in all cases except the DAC MAX5891. This chip is deployed on the GE ICS-1572 platform that was chosen for the narrow-band solution.

The spurious free dynamic range (SFDR) exceeds the GPS L1 C/A definition of 40 dBc [47] by at least 15 dBc. Nevertheless, the SFDR is given in the product documentation only for frequencies grouped around the middle of the DAC bandwidth. The spectral purity of the outer frequency bands of the spectrum would need a closer analysis.

Name	Company	Conversion rate [Gsps]	Bits / sample	Input data rate [Msps]	BW [GHz]	SFDR [dBc]	Input interface
AD9129	Analog Devices	5.7	14	1,425	1.4	65	Dual LVDS, DHST
DAC37J82	Texas Instruments	1.6	16	1,230	0.8	81	JESD204B
MAX5879	Maxim Integrated	2.3	14	1,150	2.0	73	Interleaved LVDS
DAC5681	Texas Instruments	1.0	16	1,000	0.35	55	Parallel LVDS
DAC1658D	Integrated Device Technology	2.0	16	1,000	1.0	--	JESD204B
MAX5891	Maxim Integrated	0.6	16	600	--	84	Parallel LVDS

Table 8-1 Parameters of broadband DACs

Company	BittWare	4DSP	Alpha Data	General Electrics
DAC	AD9129	AD9129	DAC5681	MAX5891
DAC module	3F230-FMC	FMC230	XRM-DAC-D4-1G	ICS-1572A
Mezzanine module	S5-PCIe-F	PC820 (RDMA)	ADM-XRC-7V1/VX330T-2	--
Carrier card	--	--	ADC-PCIE-XMC	SPR418A
PCIe version	v. 3.0 x8	v. 3.0 x8	v. 3.0 x8	v. 1.0 x8
Data rate – PCIe theory [GB/s]	8.0	8.0	8.0	2.0
Sample rate - PCIe theory [Gsps]	4.0	4.0	4.0	1.0
Specified sample rate [Gsps]	1.4	1.4	1.0	0.5
RDMA sample rate [Gsps]	--	2.8	--	1.0

Table 8-2 Overview of parameters of PCIe DAC cards

As explained in Chapter 3, the PCIe bus was chosen for the interface between the digital signal generation module and the DAC module of simulator. Therefore as a next step, a research on PCIe cards comprising such a high-speed DAC was conducted. As of October 2014, just three products deploying the DACs listed above were available on the market. These PCIe cards are listed in Table 8-2. The final PCIe card is actually a suite comprising a DAC module where the target chip is mounted, a mezzanine module implementing PCIe interface together with a FPGA and optionally also a separate carrier card with hardware connectors and cooling fans.

A closer investigation revealed that the bandwidth of the DAC is not implemented throughout the circuitry up to the PCIe interface of the PCIe cards. For the transfer and input data rate specified in the Broadband configuration, a customization of the card is needed. For all three PCIe cards, the interface drivers would have to be reconfigured and an IP core for FPGA implementing the high transfer rate would need to be developed. In case of the Alpha Data card, also the filter bandwidth of the DAC set to 350 MHz would have to be adjusted. For comparison, the GE ICS-1572 card used for the narrow-band solution is additionally listed in the table.

8.2 Signal Conversion with Narrowband DAC Card ICS-1572

For the verification of the concept of the simulator architecture, the narrowband DAC card ICS-1572 by General Electrics was installed in the Test System. The card precedes the other products listed in Table 8-2 by about five years. The block diagram of the inner structure of the card is given in Figure 8-1. The card features PCIe interface of version 1.0 with just a quarter of the data rate of other cards for the equal pin number x8. The data sheet specifies that the card should deliver up to 500 Msps conversion rate and output signals with frequency up to 250 MHz for each of the two integrated DACs. First, the performance of the continuous transfer of samples from CPU memory to the DAC was benchmarked. A spectrum analyzer was connected to assess the quality of the output signal. The results are given in Table 8-3. Successful data conversion is marked blue, violet stands for slight distortions in signal spectrum and red marks lack of continuous signal delivery. In case of usage of a single DAC, the speed up to 300 Msps can be reached without any distortion. With usage of both DACs, only 200 Msps can be converted continuously.

SR [Msps]	#	50	75	100	125	150	200	250	300	350	400	450	500
Spectrum quality	2												
Spectrum quality	1												

Table 8-3 Performance of the ICS-1572 in continuous mode

This measurement is in conflict with the specification. In the loop mode of generation, when data are once loaded to card memory and then converted in a loop, the card reached the specified sample rate of 500 Msps even for operation of both DAC channels. The result was discussed with the manufacturer.

The measured performance was confirmed by the manufacturer as the maximal performance of the continuous streaming from the memory to the card. An alternative to the streaming from CPU memory exists. It is the Remote Direct Memory Access (RDMA) streaming directly from GPU to the DAC card. The RDMA capability of the card and the ability to reach the maximum performance of the deployed PCIe v. 1.0 x8 interface, i.e. 500 Msps for each of the two DACs, is documented in [73]. The measured performance published in the document is nevertheless related to the direction from the card

to a GPU. For the other direction, only the statement of the functionality of the RDMA is given without any measurements.

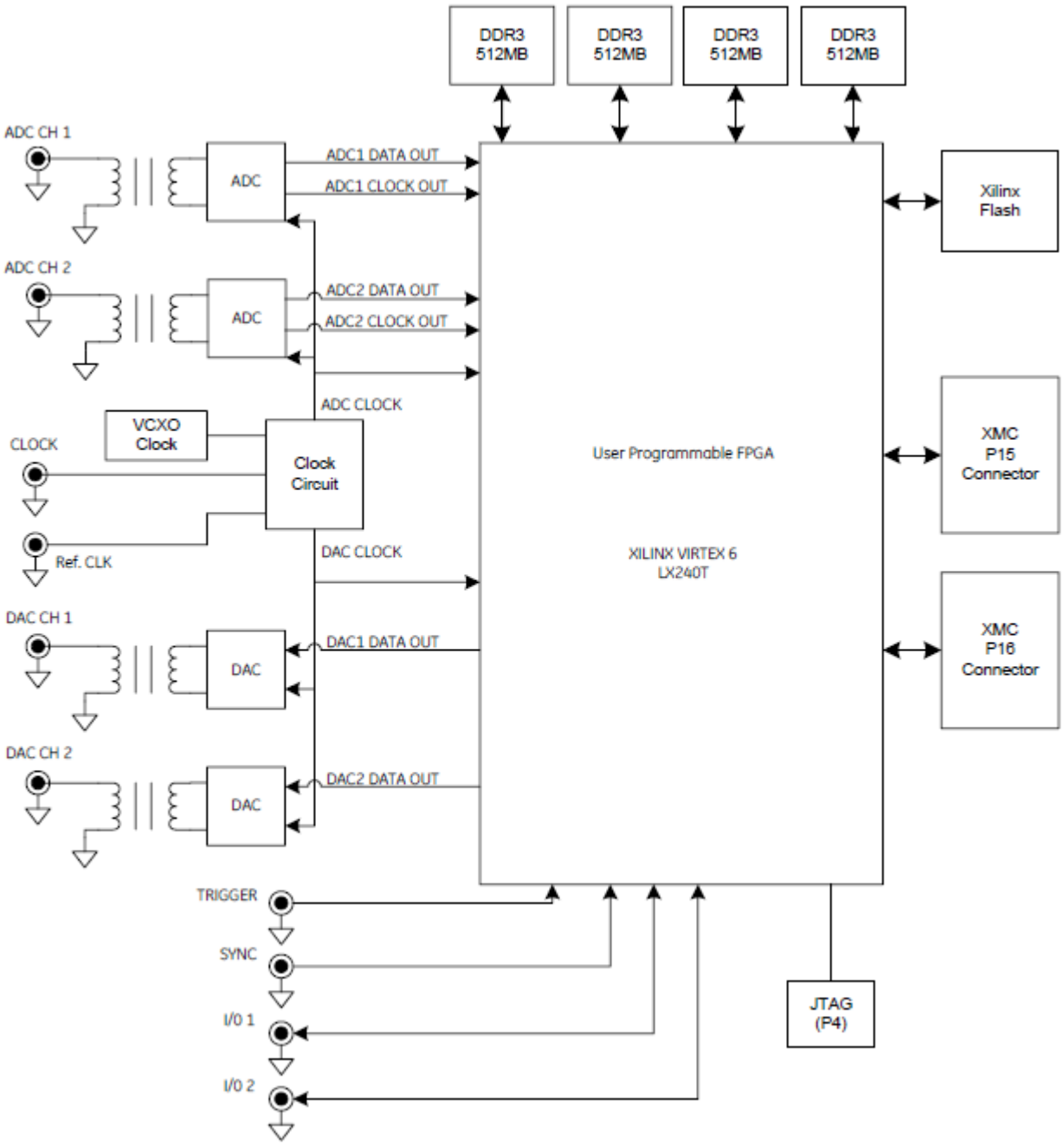

Figure 8-1 Block diagram of the inner structure of the DAC card ICS-1572 [72]

Then transfer of generated samples to the card was implemented in Test System. The generation loop specified in section 7.5.4 was enlarged by the transfer of samples from the host buffer to a special ICS-1572 DMA buffer in the same memory. The measured performance of this transfer is listed in Table 8-4. The conversion rates that were possible to reach in real-time are marked blue; the other ones are marked red. The performance of signal generation on GPU does not play a role in this case as the data transfer itself is limited by 96 Msps only.

Scenario: IF = 15.345 MHz, L1 C/A + E1 OS, 12 channels	Msps					
Configured sample rate	50	75	100	125	150	200
Measured sample rate	62	90	96	94	94	94

Table 8-4 Performance with transfer from GPU over CPU to CPU ICS buffer

Thereafter a direct copy of the samples from GPU memory to the CPU ICS buffer of the card was implemented. The difficulty was to configure the CPU ICS-1572 buffer as non-pageable memory, so that the maximum transfer speed between GPU and host memory can be reached. This maximum speed was benchmarked 12.8 GB/s on the Test System as given in section 7.6.1. The CPU buffer is a DMA buffer that needs to be allocated by a specific library function of ICS-1572. Nevertheless, CUDA offers a function `cudaHostRegister()` that declares an already allocated memory to be non-pageable. The direct transfer using this setting was implemented and reached transfer performance equivalent to performance available when CUDA based non-pageable allocation of CPU buffer is applied.

The designed algorithm uses a different order of commands than the algorithm given in section 7.5.4. The general `processBatch` function was replaced by the `pushSamplesToDacFromGpu` function. This function combines synchronization of each CUDA stream (function `syncStream`) and copy of a batch of samples from GPU to host memory (`cudaMemcpyAsync`). This function includes also function for waiting until the previous batch was converted to analog (`waitIcs`) and copy of samples from host memory to the DAC card (`icsPushBlock`). The simplified description of the algorithm in CUDA C is given as follows:

```
for (I=0;I<nRuns;I++)
for (D=0;D<NUM_DEVICES;D++)
{
  cudaSetDevice(D);
  for (J=0;J<nStreams;J++)for(K=0;K<nServiceBands;K++)
   computeFixedParsSet(pphFixedParsSet[D][J][K],&nVisSats);

  for (J=0;J<nStreams;J++)
    pIcs->pushSamplesToDac()
    {
      waitIcs()
      cudaMemcpyAsync(ppdBatchSet[D][J],pphBatchSet[D][J],Streams[D][J]);
      syncStream(Streams[D][J]);
      icsPushBlock();
     }

  for (J=0;J<nStreams;J++)
      cudaMemcpyAsync(pphFixedParsSet[D][J],ppdFixedParsSet[D][J],Streams[D][J]);
  for (J=0;J<nStreams;J++) for(K=0;K<nServiceBands;K++)
      (*pKernel[K])<<<…>>>(…);
 }
```

In this code, variables and functions are used in the same context as in Chapter 7. The algorithm was optimized to utilize the relatively long time, when samples are transferred from host memory to the DAC card. When a copy of samples to CPU buffer is made directly from GPU, then the stream

synchronization must be placed between the copy of samples from CPU to host and the copy of samples from CPU to the DAC card. Simultaneously, the computation of fixed parameters and their application in a kernel must be interleaved by a synchronization to secure the usage of the correct set of fixed parameters. The number of synchronization events must be as low as possible, which means to use only one synchronization per stream per loop cycle. Multiple loop orders reach comparable performance. This algorithm groups clearly all activities of one loop run and offers therefore good readability.

A scenario with a single L1 C/A kernel and three E1 OS kernels generating signal with SR of 300 Msps and IF of 15.345 MHz was chosen for verification of the algorithm performance. The execution was examined using CUDA Visual Profiler. The profile of the execution order of the algorithm is depicted in Figure 8-2.

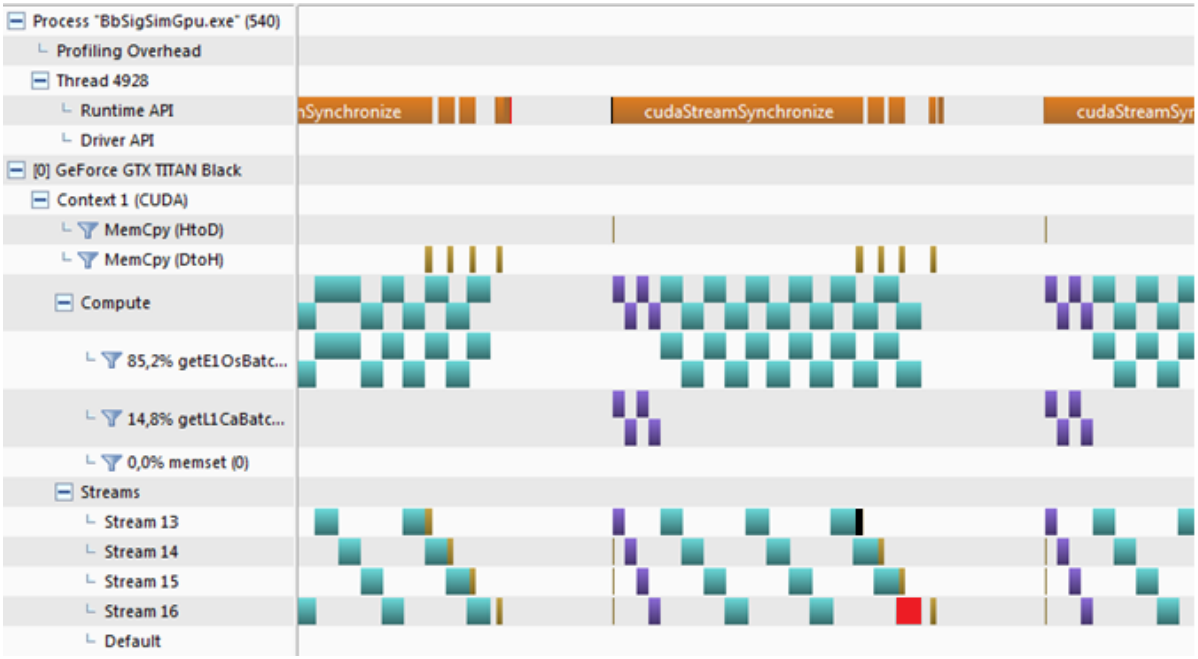

Figure 8-2 Profile of the generation algorithm with DAC card ICS-1572, SR = 300 Msps, IF=15.345 MHz

In the fourth CUDA stream in the figure (stream 16), computation of the last kernel is marked red. Between this computation and the following copy of samples from GPU to the host memory there is an offset, that is nonexistent in the other streams. Existence of this offset shows that the sample generation waits for the conversion of samples by the DAC card, as the copy can start first after the conversion of previous data.

This indicates that a slightly higher performance in terms of higher conversion rate or higher number of satellite channels could be reached.

In the row “Runtime API” containing the invocation of functions there is a gap of about 7 ms between the invocation of the kernels and invocation of the copy of the samples from GPU. The invocation and respective kernel run are marked red in the figure, the invocation of the copy from GPU to the host memory and respective execution are marked black. Obviously, the gap between the invocations is caused by waiting for the DAC card (function `waitIcs`) to convert the previous batch of samples. An interesting issue is the existence of an equivalent gap between executions of device computations in this time. Between the two invocations, there is the computation of fixed parameters (`getFixedPars`)

and waiting for the DAC card to finish the conversion of the previous batch. According to Nvidia documentation, the execution of the device code should start immediately, if there is no other device computation running, and it should be able to run in parallel to host code computation. This is, according to the profile, nevertheless not the case, the computation on GPU waits for the invocation of the next GPU code to start with the previously invocated kernels. Tests with higher SR nevertheless showed that the gap closes with higher computational load, so it is not an obstacle to reach the maximum performance.

The performance of the algorithm was measured for four scenarios and it given in Table 8-5. The first scenario generates L1 C/A and one E1OS signal service with 12 satellite channels respectively. The second, third and fourth scenario add 2, 3 and 4 E1 OS services respectively with 12 channels each. All scenarios were run with IF = 15.345 MHz and SR set consecutively to 50, 75, 125, 150, …, 450 Msps. The sample rate of the overall signal generation throughput with transfer to the DAC card ICS-1572 and a single GPU in the Test System is given in the table.

Scenario	**Measured SR [Msps]**									
Set SR [Msps]	**50**	**75**	**125**	**150**	**200**	**250**	**300**	**350**	**400**	**450**
L1 C/A + E1 OS, 12 channels	51	76	127	182	203	254	305	349	366	367
L1 C/A + 3x E1 OS, 12 channels	51	76	127	154	203	254	305	349	366	366
L1 C/A + 4x E1 OS, 12 channels	51	76	127	154	203	254	305	349	355	366
L1 C/A+ 5x E1 OS, 12 channels	51	76	127	154	203	254	312	312	312	312

Table 8-5 Signal generation performance with GPU to CPU ICS buffer transfer

The results show that the maximum performance is limited to ca. 350 Msps. This rate in terms of Msps is in correspondence with general ICS-1572 performance measurements given in Table 8-3. Within this performance capacity, ca. five signal services with 12 channels could be generated in parallel. The spectrum of the generated signal was tested with a spectrum analyzer. The output is given in Figure 8-3.

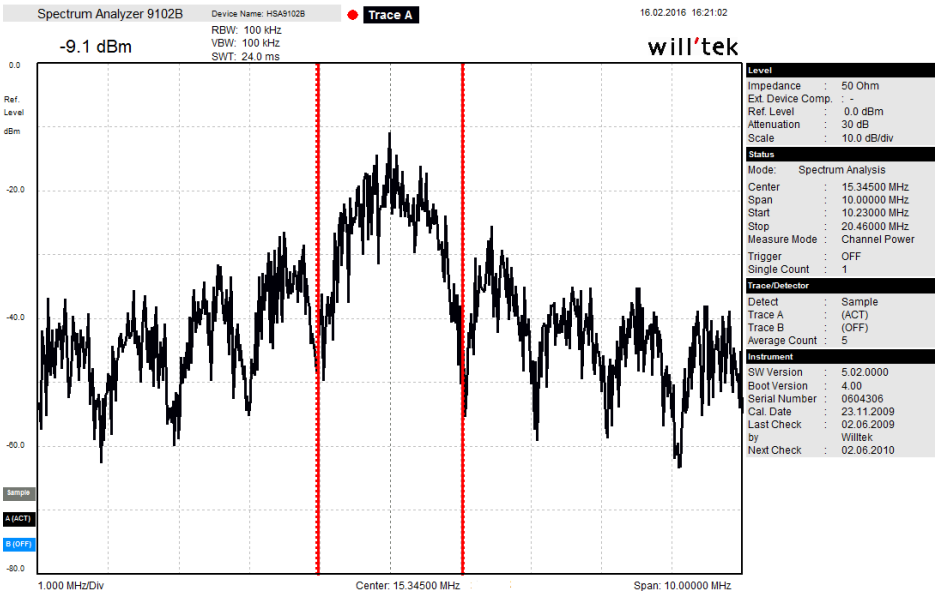

Figure 8-3 Power spectrum of L1 C/A measured by spectrum analyzer

9 Summary and Conclusions

In this thesis, a GPU-based broadband simulator architecture was proposed. The focus of this work was the design of parallelized algorithms for the digital signal generation part of the simulator. These were optimized for maximum performance on GPU. Implementation was done using consumer-level graphic cards and consumer-level PC-system. In this low-cost configuration, real-time performance was reached. Requirements on corresponding broadband DAC and upconverter are on the edge of abilities of market ready solutions. Availability of tailored solutions was nevertheless confirmed, which implies practicability of the proposed simulator architecture. For verification purposes, digital-to-analog conversion was implemented in narrowband for separated single frequency GNSS bands.

The features of the digital signal generated using the GPU platform was analyzed and conditions for sufficient precision were determined. The GPU architecture is optimized for 32-bit floating-point numbers. Broadband GNSS signal can reach sufficient precision when at least 64-bit numbers are used. This contradiction poses a challenge to the design of the signal generation algorithm in a way that would effectively exploit the computational power of the GPU.

For the generation of GNSS signal services, parallel algorithms were designed and optimized for the GPU. The algorithms overcome the low size of the fast GPU memory in contrast to the size of PRN sequences and apply the parallelization concept with hundreds of threads to the generation of GNSS signal. In contrast to it, the GNSS signal comprises only handful of services with dozens of channels and it is generated by NCOs with sequential nature. The generation of GPS L1 C/A, Galileo E1 OS and Galileo E5ab services was implemented. For E5ab with four signal components, an innovative algorithm combining the PRN sequences bitwise was designed and it proved high performance. The generation speed exceeded the speed of generation of E1 OS with only two signal components.

The transfer of generated samples from GPU to CPU was parallelized to the signal generation on GPU to 75%. In case of multiple GPUs, this is a maximum. In case of a single GPU, even though an algorithm for full parallelization was developed, an implicit full synchronization occurs in contrast to the CUDA programming documentation. In case of generation of multiple services, the full parallelization would improve the performance just by 7%.

The digital signal generation algorithm was also adjusted for execution on multiple GPUs in parallel. For two GPUs and high number of satellite channels per service, the maximum performance improvement - to double the generation speed - was reached. As the data transfer takes up ca. 32% of the computation time, for four or more GPUs, the linear performance increase would occur only for higher computational load in terms of number of signal channels.

Tor real-time generation of the broadband GNSS signal, the sample rate of 1,400 Msps was designed. The digital signal generation of 12 channels of GPS L1 C/A and Galileo E1 OS and E5ab respectively, reached even higher throughput. Nevertheless, a performance overhead is necessary for real-time operation including the data transfer to a DAC.

For the broadband digital-to-analog conversion, DAC devices are already available on the market. However, a DAC platform with PCIe interface, necessary for connection to a mass-market PC, is a niche application with slow development progress. A tailored device was purchased and tested in the simulator system.

A broadband upconverter is also not an off-the-shelf component. It is possible to get a tailored solution, but the parameter of the broadband filters are nevertheless still below expectations. The trend toward broadband telecommunication will probably make also this topic solvable in the near future.

To evaluate the capabilities of the broadband architecture, a comparison with commercial simulators can be done. The key figures of merit of market-ready simulators are listed in Chapter 2 GNSS Signal Simulators.

The broadband architecture fits the category of real-time signal simulator. It can also be used for a DIF generator. The replay functionality would be possible with a fast storage medium (e.g. solid state drive - SSD) connected over the PCIe interface. This configuration was nevertheless not tested.

The high-end commercial signal simulators offer the full scale of existing and planned GNSS services. In this work, services GPS L1 C/A, Galileo E1 OS and Galileo E5ab were implemented. Nevertheless, any other CDMA as well as FDMA based service could be added. The performance of the signal generation will be dependent on modulation features. As the experience with E1 OS and E5 shown, the number of signal components and the volume of modulation data can be outweighed by highly parallelized generation operations and a data-compressing storage pattern. This possibility depends on to which extent the modulation computation operation and storage pattern coincide with the specific features of the GPU architecture.

In terms of number of signal channels, the high-end simulators offer up to 160 channels with 640 multipath channels and 4 interference channels. With the chosen hardware, this simulator implementation can generate ca. 48 signal channels with the full bandwidth. This number can be increased by deployment of a professional-level GPU. The linear growth in performance measured with the second GPU indicates, that with higher number of GPUs (up to four can be integrated in one system), such a high number of channels can be reached. In addition, a deployment of a stronger PC-system in terms of memory speed and processor performance would be an easy way for performance increase.

The number of simultaneous carriers offered by the high-end simulators is 9-10. At the broadband architecture, the number of carriers can be as high as the number of channels. Within the given bandwidth of the single broad band, each channel can be generated with any chosen frequency. More limiting is the number of signal services generated in parallel. Each additional service causes overhead by the copy of generated signal from multiprocessor to the common signal in the GPU memory.

Modeling of various signal impairments and additional precision enhancing services is an essential part of functionality of each commercial simulator. Each impairment effect except multipath can be modeled by configuration of basic signal parameters – phase, frequency and amplitude. It can be input to the digital GNSS signal generation module introduced in this work. The multipath reflections could be easily added to the signal generation module. Just an enlargement of the definition of signal channel by multipath channels would be necessary. No change to kernel design would be required, as multipath channels do not need any additional memory for PRN sequences.

References

[1] I. Petrovski, B.Townsend, T. Ebinuma. GNSS Simulators, Part 1: Testing Multi-GNSS Equipment Systems, Simulators and the Production Pyramid, InsideGNSS, July/August 2010

[2] I. Petrovski, T. Ebinuma. GNSS Simulators, Part 2: Everything You Always Wanted to Know But Were Afraid to Ask, InsideGNSS, pp. 48 – 58, September 2010

[3] European GNSS Agency. GNSS Market Report, http://www.gsa.europa.eu/market/market-report, March 2015

[4] P. Berglez, E. Wasle, J. Seybold, B. Hofmann-Wellenhof. GNSS Constellation and Performance Simulator for Testing and Certification, Proceedings of ION GNSS 2009, Savannah, USA, 2009

[5] The TEST CASE Consortium. Website of European GNSS Simulation and Testing Tools Infrastructure, http://gnss-test-portal.eu, 2016

[6] A. Tetewsky, and A. Soltz, D. Fuhry, G. Barton, D. Eyring, B. Goossens, M. Dodds, L. Fava. Validating the Validation Tool: Defining and Measuring GPS Simulator Specification, Proceedings of ION GPS 1997, Kansas City, USA, September 1997

[7] FCC. Guidelines for Testing and Verifying the Accuracy of Wireless E911 Location Systems, FCC OET Bulletin No. 71., http://www.fcc.gov/oet/info/documents/bulletins, April 2000

[8] P. Anderson, E. Anyaegbu, R. Catmur. The Future of Multi-Constellation GNSS Test Standards for Cellular Location, Proceedings of ION GNSS 2012, Nashville, USA, 2012

[9] Spirent Communications. GSS900 GNSS/GPS Simulator, http://www.spirent.com/Products/GSS9000-GPS-Simulator, 2014

[10] Meguro Electronics Corporation. MSG-2051A GPS Signal Generator, http://www.meguro.co.jp/english/product/category/category_01/msg2051a_eng.html, 2013

[11] Orolia Group. Spectracom Products - GSG-51 Single Channel GNSS Tester, http://spectracom.com/products-services/gnss-simulation/gpsgnss-simulators#anchor-1891, 2013

[12] IFEN GmbH, NavX-NCS Professional GNSS Simulator, http://www.ifen.com/products/navx-ncs-professional-gnss-simulator.html, 2013

[13] CAST Navigation. The CAST-2000 GPS Simulation System, http://castnav.com/products/cast_2000.html, 2013

[14] Cobham Group. GPSG Simulators, http://ats.aeroflex.com/products/gps-simulators/gpsg-simulators, 2014

[15] I. Petrovski, T. Tsujii, J.-M. Perre, B. Townsend, and T. Ebinuma. GNSS Simulation: A User's Guide to the Galaxy, Inside GNSS, vol. 5, no. 5, pp. 52–61, October 2010

[16] Rohde & Schwarz GmbH. GNSS Simulator for the R&S®SMBV100A Vector Signal Generator, https://www.rohde-schwarz.com/product/gnss-productstartpage_63493-11461.html

[17] National Instruments. NI GPS Simulation Toolkit for LabVIEW, http://sine.ni.com/nips/cds/view/p/lang/de/nid/204980, 2014

[18] Racelogic Ltd., LabSat 3 GPS Simulator, http://www.labsat.co.uk/index.php/en/

[19] Navsys Corporation. Under Your Control: GNSS Simulation and Testing, http://www.navsys.com

[20] American GNC Corporation. GPS/IMU Realtime Simulator AGNC-2000 RTGIS, http://www.americangnc.com/products/rtgis.htm

[21] P. Boulton, A. Read, R. Wong. Formal Verification Testing of Galileo RF Constellation Simulators, Proceedings of ION GNSS 2007, Forth Worth, Texas, USA, 2007

[22] G. Heinrichs, M. Irsigler, R. Wolf, G. Prokoph. Performance Evaluation of the Multi-Costellation and Multi-Frequency GNSS RF Navigation Constellation Simulator NavX-NCS, Proceedings of ION GNSS 2008, Savannah, USA, 2008

[23] T. Dautermann, M. Sgammini and S. Pullen. Ionospheric Threat Simulation for GNSS Using the Spirent Hardware Signal Simulator, GPS Solutions, vol. 18, no. 3, pp. 365-373, June 2014

[24] GPS World. Simulator Buyers Guide, GPS World, vol. 23, no. 5, May 2013

[25] O. Julien, B. Zheng , L. Dong, and G. Lachapelle. A Complete Software-Based IF GNSS Signal Generator for Software Receiver Development, Proceedings of ION GNSS 2004, Long Beach, USA, 2004

[26] A. Pósfay, T. Pany, B. Eissfeller. First Results of a GNSS Signal Generator Using a PC and a Digital-to-Analog Converter, Proceedings of ION GNSS 2005, Long Beach, USA, 2005

[27] Y. Kou, H. Zhang. Verification Testing of a Multi-GNSS RF Signal Simulator, Inside GNSS, pp. 52–61, July/August 2011

[28] R. Li, D. Zeng, T. Long, L. Zhang, Architecture and Implementation of a Universal Real-Time GNSS Signal Simulator, Proceedings of ION ITM 2010, San Diego, California, USA, 2010

[29] M. G. Petovello, C. O'Driscoll, G. Lachapelle, D. Borio and H. Murtaza. Architecture and Benefits of an Advanced GNSS Software Receiver, Journal of Global Positioning Systems, vol. 7, no. 2, 2008

[30] T. Hobiger, T. Gotoh, J. Amagai, Y. Koyama, and T. Kondo. A GPU Based Real–Time GPS Software Receiver, GPS Solutions, vol. 14, no. 2, pp. 207–216, March 2010.

[31] C. Wu, Y. Qian, X. Cui, M. Lu. The Optimized Method and Algorithms in the CPU&GPU-Based GNSS Software Receiver. Proceedings of ION GNSS 2009, Savannah, USA, 2009

[32] T. Pany, B. Riedl, J. Winkel. Efficient GNSS Signal Acquisition with Massive Parallel Algorithms using GPUs, Proceedings of ION GNSS 2010, Portland, USA, 2010

[33] U. Haak, H-G. Büsing, P. Hecker. Performance Analysis of GPU Based GNSS Signal Processing, Proceedings of ION GNSS 2012, Nashville, USA, 2012

[34] L. Xu, N. I. Ziedan, W. Guo, X. Niu. NAVSDR: A GPU-based Modular GPS Software Receiver, Proceedings of ION GNSS 2015, Tampa, USA, 2015

[35] K. Park, J. Yang, C. Park, M. Lee. Implementation of GPGPU Based Real-Time Signal Acquisition and Tracking Module for Multi-Constellation GNSS Software Receiver, Proceedings of ION GNSS 2014, Tampa, USA, 2014

[36] A. Kneznevic, C.O'Driscoll, G. Lachapelle. Co-Processor Aiding for Real-Time Software GNSS Receivers, Proceedings of ION ITM 2012, Newport Beach, California, USA, 2012

[37] I. Bartunkova, B. Eissfeller. Massive Parallel Algorithms for Software GNSS Signal Simulation using GPU, Proceedings of ION GNSS 2012, USA

[38] Q. Li, H. Li, M. Lu. A CUDA-Based Real-Time Software GNSS IF Signal Simulator, Proceedings of China Satellite Navigation Conference (CSNC) 2012, Guanzhou, China, 2012

[39] L. Tan. Digital Signal Processing - Fundamentals and Applications, 1st edition, ISBN 978-080550572, Academic Press, 2007

[40] A. Wendemuth, E. Andelic, S. Barth, M. Katz, S. Krüger, M. Mamsch, M. Schafföner. Grundlagen der digitalen Signalverarbeitung - ein mathematischer Zugang, ISBN 978-3540218852, Springer Berlin Heidelberg, New York, 2005

[41] W. Tomasi. Electronic Communications Systems: Fundamentals Through Advanced, 5th ed., ISBN 978-0130494924, Pearson, 2003

[42] J. H. Reed. Software Radio: A Modern Approach to Radio Engineering, ISBN 978-0130811585, Prentice Hall PTR, 2002

[43] V. F. Kroupa. Direct Digital Frequency Synthesizers, ISBN 978-0780334380, IEEE Press, 1998

[44] P. Misra, P. Enge. Global Positioning System: Signals, Measurements, and Performance, ISBN 978-0970954404, Ganga-Jamuna Press, 2001

[45] K. Borre, D. M. Akos, N. Bertelsen, P. Rinder, S. H. A. Jensen. Software-Defined GPS and Galileo Receiver: A Single-Frequency Approach, ISBN 978-0817643904, Birkhäuser, Basel 2007

[46] E. D. Kaplan, C. J. Hegarty. Understanding GNSS – Principles and Applications, ISBN 978-1580538947, Artech House, Boston, 2006

[47] U.S. Government. Interface Specification for GPS L1 and L2, revision H, http://www.gps.gov/technical/icwg/IS-GPS-200H.pdf, September 2013

[48] IFEN Gmbh., NavX-NCS Navigation Constellation Simulator User Manual, Poing, 2014

[49] M. S. Braasch. GPS Multipath Model Validation, Proceedings of IEEE Position, Location and Navigation Symposium PLANS 1996, Atlanta, USA, 1996

[50] CSR plc. CSR product website, http://www.csr.com/products/35/sirfstariv-gsd4e

[51] P. Boulton, A. Read, G. MacGougan, R. Klukas, E. Cannon, G. Lachapelle. Proposed Models and Methodologies for Verification Testing of AGPS-Equipped Cellular Mobile Phones in the Laboratory, ION GNSS 2002, Portland, USA, 2002

[52] A. Jahn, S. Buonomo, M. Sforza and E. Lutz. Narrow- and Wideband Channel Characterization for Land Mobile Satellite Systems: Experimental Results at L-Band, International Mobile Satellite Conference, Ottawa, Canada, 1995

[53] Institut für Raumfahrttechnik und Weltraumnutzung, Universität der Bundeswehr. INDOOR - Galileo/GPS Indoor Navigation & Positionierung : Abschlussbericht, DOI: 10.2314/GBV:775691623, Neubiberg, 2013

[54] E. Lutz, D. Cygan, M. Dippold, F. Dolainsky, W. Papke. The Land Mobile satellite Communication Channel – Recording, Statistics and Channel Model, IEEE Transactions on Vehicular Technology, vol. 10, no. 2, May 1991

[55] A. Teuber, H.-J. Thierfelder, A. Wolfe, G. Hein. Fighting the Fading, InsideGNSS p. 47-54, May 2008

[56] M. Paonni, V. Kropp, A. Teuber, G.W. Hein. A New Statistical Model of the Indoor Propagation Channel for Satellite Navigation, Proceedings of ION GNSS 2008, Savannah, USA, 2008

[57] A. M. Saleh, R. A. Valenzuela. A statistical model for indoor multipath propagation, IEEE Journal on Selected Areas of Communications, vol. 5, no. 2, pp. 128-137, February 1987

[58] W. C. Jakes Jr., Microwave Mobile Communications, ISBN 978-0471437208, New York: John Wiley & Sons, USA, 1974

[59] I. Bartunkova, M. Paonni, B. Eissfeller. Satellite to Indoor Channel Characterization with Implementation in a GNSS Software Simulator, Proceedings of ION GNSS 2010, Portland, USA, 2010

[60] D. Parsons. The Mobile Radio Propagation Channel, ISBN 978-0470218242, New York: John Wiley & Sons, USA, 1992

[61] U.S. Government. Interface Specification for GPS L5, revision D, http://www.gps.gov/technical/icwg/IS-GPS-705D.pdf, September 2013

[62] U.S. Government. Interface Specification for GPS L1C, revision D, http://www.gps.gov/technical/icwg/IS-GPS-800D.pdf, September 2013

[63] European Commission. Galileo Open Service System in Space ICD, http://www.gsc-europa.eu/gnss-markets/segments-applications/os-sis-icd, September 2010

[64] Xilinx Inc., Xilinx Homepage - MicroBlaze Soft Processor Core, http://www.xilinx.com/products/design-tools/microblaze.html, 2015

[65] M. J. Flynn. Some Computer Organizations and Their Effectiveness, IEEE Transactions on Computers, vol. c-21, no. 9, 1972

[66] S. W. Smith. The Scientist and Engineer's Guide to Digital Signal Processing, 1st ed., ISBN 978-0966017632, California Technical Pub., 1997

[67] Nvidia Corporation. Nvidia GK 110 Whitepaper, http://www.nvidia.com/content/PDF/kepler/NVIDIA-Kepler-GK110-Architecture-Whitepaper.pdf, 2010

[68] Khronos Group. OpenCL Homepage - The Open Standard for Parallel Programming of Heterogeneous Systems, https://www.khronos.org/opencl/, 2015

[69] Nvidia Corporation. CUDA Programming Guide 6.0., http://developer.nvidia.com/cuda/nvidia-gpu-computing-documentation, 2012

[70] Nvidia Corporation. CUDA Math API Reference Manual 5.5, https://developer.nvidia.com/cuda-toolkit-55-archive

[71] M. Anghileri, T.Pany, D. Sanroma Guixens, J.Won, A. Ayaz. Performance Evaluation of a Multi-frequency GPS/Galileo/SBAS Software Receiver, Proceedings of ION GNSS 2007, Fort Worth, Texas, USA, 2007

[72] GE Intelligent Platforms, Inc., Hardware Reference Manual - ICS-1572 Operator's Manual E11496 Rev. A, http://www.geautomation.com/products/ics-1572a/

[73] GE Intelligent Platforms, Inc., GPUDirect™ RDMA, http://www.geautomation.com/products/ics-1572a/p3597, 2013