TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

Ville Eerola

Design and Silicon Area Optimization of Time-Domain GNSS Receiver Baseband Architectures

Julkaisu 1582 • Publication 1582

Tampere 2018

Tampereen teknillinen yliopisto. Julkaisu 1582
Tampere University of Technology. Publication 1582

Ville Eerola

Design and Silicon Area Optimization of Time-Domain GNSS Receiver Baseband Architectures

Thesis for the degree of Doctor of Science in Technology to be presented with due permission for public examination and criticism in Festia Building, Auditorium Pieni Sali 1, at Tampere University of Technology, on the 26th of October 2018, at 12 noon.

Tampereen teknillinen yliopisto - Tampere University of Technology
Tampere 2018

Doctoral candidate: Ville Eerola
Laboratory of Electronics and Communications Engineering
Faculty of Computing and Electrical Engineering
Tampere University of Technology
Finland

Supervisor: Prof. Jari Nurmi
Laboratory of Electronics and Communications Engineering
Faculty of Computing and Electrical Engineering
Tampere University of Technology
Finland

Pre-examiners: Dr. Hannu Heusala
FiNoC Design Consulting
Finland

Dr. Jari Syrjärinne
HERE Technologies
Finland

Opponents: Prof. Jorma Skyttä
Department of Signal Processing and Acoustics
Aalto University
Finland

Dr. Jari Syrjärinne
HERE Technologies
Finland

ISBN 978-952-15-4217-6 (printed)
ISBN 978-952-15-4235-0 (PDF)
ISSN 1459-2045

ABSTRACT

The use of Global Navigation Satellite Systems (GNSSs) in a wide range of portable devices has exploded in the recent years. Demands for a lower cost while expecting longer battery life and better performance are constantly increasing. The general GNSS receiver operation and algorithms are already well studied in the literature, but the hardware architectures and designs have not been discussed in detail.

This thesis introduces a high level gate count estimation method that provides good accuracy without requiring the hardware being fully specified. It is based on developing hierarchical models, which are parameterizable, while requiring minimal amount of information about the silicon technology used for the implementation. The average accuracy has been shown to be 4%.

Three time-domain, real-time GNSS receiver baseband architectures are described with a discussion about various optimization methods for efficient implementation: the correlator, the matched filter, and the group correlator, which is a new architecture combining some of the features of the two first ones.

Four use cases are defined for different GNSS operating modes: Acquisition, tracking, assisted GNSS, and the combination of the first three modes. A comparison is made for receiver basebands including all necessary blocks

for full functionality to find out which of the three architectures provides the most silicon area efficient implementation.

It is shown that the correlator offers good flexibility, but yields the highest silicon area for acquisition use cases. The matched filter is best suited for the acquisition, but has large overhead when it comes to tracking the signals. The group correlator offers a reasonably good flexibility and area efficiency in all use cases.

The main contributions of the thesis are: Development of domain specific optimizations for GNSS receivers and an accurate gate count estimation method, which are applied for a quantitative comparison of different GNSS receiver architectures. The results show that no single architecture excels in all cases, and the best choice depends on the actual use case.

PREFACE

The work presented in this thesis has been carried out during the years 1999–2017 while being employed in VLSI Solution Oy, u-Nav Microelectronics, Inc., Nokia Oyj, and u-Blox Espoo Oy. The thesis has also been supported by the Alfred Kordelin Foundation.

I wish to express my thanks to my supervisor, Prof. *Jari Nurmi*, for his encouragement, guidance, patience, fruitful discussions, and helpful feedback throughout the work. I would also like to thank the reviewers of this thesis, Dr. Tech. *Jari Syrjärinne* and Prof. *Hannu Heusala* for their helpful feedback.

I would like to thank all my colleagues in the companies I have worked while working on this thesis. I would like to express special gratitude to *Tapani Ritoniemi*, who drove me to try harder to find new solutions to problems that seemed almost impossible at the first glance; *Kim Kaisti*, who was always very enthusiastic about trying out new things and getting them done; and Dr. Tech. *Seppo Turunen* for his insight, inspiring discussions, and example. I also wish to thank my co-authors, *Harri Valio* and Lic. Tech. *Samuli Pietilä*.

Finally, I would like to thank my family and especially my beloved wife, *Sanna* for her support and encouragement during this work.

Hämeenlinna, September 2018

Ville Eerola

TABLE OF CONTENTS

LIST OF PUBLICATIONS

This thesis is based on the following papers published in open literature. In the text, these publications are referred to as [P1], [P2],. . . , and [P6] and are appended in the concluding half of the thesis.

[P1] Ville Eerola and Jari Nurmi, "Correlator Design and Implementation for GNSS Receivers," in *NORCHIP, 2013*, Nov 2013.

[P2] Ville Eerola, "Rapid Parallel GPS Signal Acquisition," in *Proceedings of the 13th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GPS 2000)*, Salt Lake City, UT, Sep. 2000, pp. 810–816.

[P3] Ville Eerola, "Optimizing Matched Filters for GNSS Receivers," in *2017 International Conference on Localization and GNSS (ICL-GNSS)*. IEEE, 2017.

[P4] Ville Eerola, Samuli Pietilä, and Harri Valio, "A Novel Flexible Correlator Architecture for GNSS Receivers," in *Proceedings of the 2007 National Technical Meeting of The Institute of Navigation (ION NTM 2007)*, San Diego, CA, Jan. 2007, pp. 681–691.

[P5] Ville Eerola and Jari Nurmi, "High-level parameterizable area estimation modeling for ASIC designs," *Integration, the VLSI Journal*, vol. 47, no. 4, pp. 461–475, Sep. 2014.

[P6] Ville Eerola and Jari Nurmi, "Area Estimation of Time-Domain GNSS Receiver Architectures," in *2014 International Conference on Localization and GNSS (ICL-GNSS)*. IEEE, 2014.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

3GPP	Third Generation Partnership Project
A-GNSS	Assisted GNSS
A-GPS	Assisted GPS
ADC	Analog to Digital Converter
AFC	Automatic Frequency Control, an alternative term for FLL
ASIC	Application Specific Integrated Circuit
AWGN	Additive White Gaussian Noise
BB	Baseband
BDD	Binary Decision Diagram
BER	Bit Error Rate
BIST	Built In Self-Test
BOC	Binary Offset Carrier
BPSK	Binary Phase-Shift Keying
CAD	Computer Aided Design
CBOC	Composite Binary Offset Carrier, a particular implementation of MBOC
CDFG	Control Data Flow Graph
CDMA	Code Division Multiple Access
CMOS	Complementary Metal Oxide Semiconductor
DFG	Data Flow Graph
DLL	Delay Locked Loop

DSP	Digital Signal Processing
DSSS	Direct-Sequence Spread Spectrum
E911	Enhanced 911
EDA	Electronic Design Automation
EU	European Union
FCC	Federal Communication Commission
FDMA	Frequency Division Multiple Access
FFT	Fast Fourier Transform
FLL	Frequency Locked Loop, an alternative term for AFC
FOC	Full Operational Capability
FPGA	Field Programmable Gate Array
GC	Group Correlator
GEO	Geostationary Earth Orbit
GLONASS	GLObal'naya NAvigatsionnaya Sputnikovaya Sistema, GLObal NAvigation Satellite System
GNSS	Global Navigation Satellite System
GPS	Global Positioning System
HW	Hardware
IC	Integrated Circuit
ICD	Interface Control Document
IF	Intermediate Frequency
IGSO	Inclined Geosynchronous Satellite Orbit
I/O	Input and Output
IOV	In Orbit Validation
IP	Intellectual Property, in HW design context, often interpreted to mean a subdesign licensed from a 3rd party
LFSR	Linear Feedback Shift Register
LNA	Low Noise Amplifier

LSB	Least Significant Bit
MBOC	Multiplexed BOC
MEO	Medium Earth Orbit
MF	Matched Filter
MSB	Most Significant Bit
NAND	NOT-AND is a logic gate which produces an output which is false only if all its inputs are true.
NAND2	Two-input NAND gate
NCO	Numerically Controlled Oscillator
NTT	Number Theoretic Transform
PLL	Phase Locked Loop
PRN	Pseudo Random Number
C/A	Coarse/Acquisition
PVT	Position, Velocity, and Time
QMBOC	Quadrature Multiplexed BOC
RAM	Random Access Memory
RF	Radio Frequency
ROM	Read Only Memory
RTL	Register Transfer Level
SBAS	Satellite Based Augmentation System
SDR	Software Defined Radio
SIS-ICD	Signal-In-Space Interface Control Document
SNR	Signal to Noise Ratio
SR	Shift Register
SRAM	Static Random Access Memory
SS	Spread Spectrum
SW	Software

TMBOC	Time-Multiplexed BOC
TTFF	Time-To-First-Fix
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit, a 1980s United States government program

1. INTRODUCTION

The number of GNSS receivers has exploded with the introduction of personal navigation devices and inclusion of GNSS receivers in mobile phones, which was originally driven by the Federal Communication Commission (FCC) e-Call, Enhanced 911 (E911), mandate [1]. The great potential of ubiquitous location information availability for a wide range of applications ranging from advertising to personal navigation is now taken for granted by the general public. Today, GNSS receivers are becoming a common feature in a wide range of electronic devices ranging from digital cameras to sports watches and tracking tags for valuables. These battery powered, portable consumer devices place great demands on their components for small size, low power consumption and low cost.

Three qualities characterize the performance of any Integrated Circuit (IC): space, time, and energy. Space determines how big the IC is, which is usually also tied to the cost of the device. Time is a measure of how quickly a certain task is performed by it. Finally, any task performed by the IC requires some energy. Energy is the product of power and time. Since time itself is already a performance characteristic, power consumption is often used instead of energy as a performance characteristic of ICs. In any case, these three parameters are not independent, so that it is not possible to optimize just one of the three characteristics without affecting the others. Cost and power consumption are critical figures of merit for any portable

electronic devices, and certainly also for GNSS receivers. The equivalent gate count of digital integrated circuits is an important measure as it affects both the cost and the power consumption [2].

1.1 Research Problem and Scope of the Thesis

While the operation and implementation of the GNSS receivers are well researched subjects, there seems not to exist a good analysis of the different receiver architectures and use cases affecting their gate counts. As GNSS receivers are targeted toward cheaper and lower power applications, good understanding of the receiver gate count becomes more important. The performance metric of time is affected most by the selection of the algorithms, which is covered well in the existing literature. However, the architecture selection will also have an effect on the size of the resulting circuit, and estimating the size, will give the algorithm designers the possibility of taking this metric into account when they are selecting the proper architecture for each use case.

The power consumption of a digital circuit can be divided into a static and a dynamic part. The static part is proportional to the gate count and the dynamic part is proportional to the frequency and operating voltage in addition to the gate count. Thus, understanding the gate count of the circuit is important part of understanding the power consumption. However, in order to reduce the scope of the thesis, we will not consider the power consumption as a parameter to be optimized.

The gate count measure used in this thesis is based on the relative cell area of the implemented digital logic compared with the area of a single Two-input NAND gate (NAND2). Common Electronic Design Automation

(EDA) software for logic synthesis and optimization, such as the Synopsys Design Compiler [3], report the total cell area of the resulting design, which can be turned into the relative cell area by dividing the total area by the area of a single NAND2 gate. The definition of the equivalent gate count can be based on either the area or the number of NAND2 gates needed to implement the same logic. There is a slight difference here, as it is more efficient to create the more complex gates directly instead of building them from the NAND2 gates. The selection of the area-based gate count measure in this thesis is pragmatic: The gate area has more direct relationship to the final silicon area, which defines the cost of the silicon. The product cost also includes other parts, such as the package, but they are out of the scope of this thesis.

In this thesis, we will concentrate on the implementation and optimization of digital receiver channels for GNSS receivers. The target of the optimizations is to minimize the silicon area of the resulting implementation while fulfilling the requirements for performance in a given operating scenario. For this purpose we will take a look of three different receiver Baseband (BB) implementations in four separate cases. The comparison results will then show which architecture is best suited for each case, so that we can do a quantitative analysis for their optimality.

There are several ways the GNSS receivers can be implemented. Figure 1 illustrates some of the possibilities for the baseband implementation. The ones selected for this thesis are marked with a **boldface** font. The first selection to make is to decide whether the baseband algorithms are implemented in Hardware (HW) or Software (SW). Traditionally, all GNSS receivers have been implemented in hardware. The Software Defined Radio (SDR) concept [4] was applied to GNSS receivers only later [5]. We will concentrate on the hardware implementations, since they have a direct

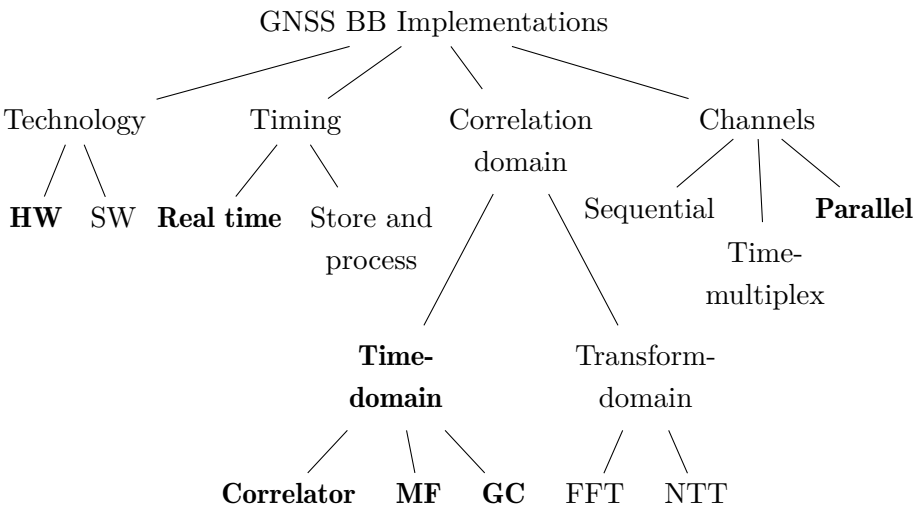


Fig. 1: *Taxonomy of various GNSS receiver BB implementations.*

relationship with the gate counts, whereas the software implementation gate counts depend in a much more complex way from the architecture. It is possible to implement the GNSS receiver in a way that the incoming signal is stored first in a memory, and the processing will read the samples of there instead of processing them in real time as they arrive from the Analog to Digital Converter (ADC) [6]. The store and process implementation allows processing the data with potentially smaller hardware running at a much higher clock frequency compared with the signal sample rate. The second way is more traditional, since it limits the amount of memory needed, and simplifies the control requirements of the implementation. All the early GNSS receivers, e.g. [7], were implemented to process the signal in real time. We have chosen to restrict ourselves in this thesis to the real-time implementations. The most critical operation in a GNSS receiver baseband is performing a cross-correlation between the received signals and locally generated replicas. The correlation can be done in sample-

by-sample fashion, i.e. in time-domain, or in a transform-domain by first transforming the signal e.g. to frequency-domain using a Fast Fourier Transform (FFT) [8] or Number Theoretic Transform (NTT) [9]. There are a multitude of different ways to do the correlation in time-domain, such as traditional correlator [P1], Matched Filters (MFs) [P2],[P3], and Group Correlator (GC) [P4]. The transform-domain processing is done in a block-wise manner and is more popular with SW and/or store-and-process implementations, so we have chosen to concentrate on time-domain implementations. The channels can be operated in sequential, time-multiplexed or parallel, by making different trade-offs. Sequential and time-multiplexing receivers both dedicate a single channel hardware to the reception of one satellite signal at a time and sequencing the channel for the available signals over time. The difference between these two is in the time spent in tracking one satellite. Sequential receivers would spend at least one second tracking a single satellite at a time whereas time-multiplexed receivers would use just a few milliseconds per satellite [10]. Practically all modern GNSS receivers use a parallel channel implementation which processes the channels in parallel even if internally the hardware would be time multiplexed and we will consider parallel receivers in this thesis. We will take a more detailed look on the correlation implementations in Chapter 4.

1.2 Main Contributions

The main contributions of the thesis can be grouped into three main categories and are summed up as follows:

1. Presenting an easy-to-use, accurate method for gate count estimation, which allows parameterizable models to be created for designs at very early stage. The model is built in a hierarchical way from sub-models,

which can later be refined to improve the accuracy. The method is well suited for architectural exploration before the design has been fully refined.

2. Developing domain-specific ways to optimize digital baseband blocks of GNSS receivers using different architectures. These range from arranging operations in optimal order, devising an optimal new way for adding a large amount of numbers, adjusting the sample rate to alter filter length in time, to creating completely new architectures.

3. Comparing different GNSS receiver baseband architectures in multiple use cases showing which of the architectures can be implemented in the smallest silicon area for each given case. The comparison was performed for realistic but limited use cases and can be used as an example for selecting the correct implementation for any given use case.

1.3 Author's Contribution

This thesis summarizes the author's work on GNSS receiver development. The research has been done while working on commercial IC projects, and thus has direct relevance in advancing the state of the art in the GNSS industry. The scientific contributions are captured in the published papers. The commercially valuable contributions are also filed and granted as patents. The publications that are part of this thesis and the related patents as well as the author's contributions to each of them are elaborated next.

1.3.1 Publications

Publication [P1] documented ideas for optimizing correlator-based GNSS receivers that were developed earlier by the author and used in commercial GNSS receiver ICs. Many of the ideas presented in this paper were also filed earlier as patents invented by the author. The highlights include discussion on time-multiplexing the correlator hardware and smartly implementing additional signal processing functions within the correlator to reduce the number of the correlator fingers needed. There was a short section about power optimization using time-multiplexing, but the treatment was very superficial and based on some rough estimates about the switching activity of the logic gates. It is not intended to include this part of the manuscript in this thesis as it is out of the scope of it. The author was the main contributor of the paper.

Publication [P2] describes a search unit based on a Matched Filter and optimized for cold start. The work was done when massively parallel search engines for GNSS were not yet commonly considered for commercial receivers. The developed implementation was extremely area-efficient and it enabled searching all Global Positioning System (GPS) satellites in less than two seconds. The author's contribution to the paper was designing and verifying the search unit as well as being the main contributor to the paper.

Publication [P3] summarizes the work of the author on optimizing MF-based GNSS receivers which was carried out over the course of several years. The main contributions include a way to optimally implement the adder in the MF where a huge number of inputs need to be added together in very short time, as well as ideas to improve the usability of the MF for GNSS receivers by allowing its apparent length in time to be adjusted according

to the incoming signal and allowing further coherent and non-coherent integration to be applied after the MF. Many of the ideas presented in this paper were filed earlier as patents invented by the author. The author was the sole author of this publication.

Publication [P4] presented the Group Correlator architecture as an alternative for the traditional correlator or MF-based GNSS receivers. Extensive simulations were performed to optimize the design parameters, and simulation results were included to show that the new receiver architecture met the performance targets. The author's contribution was to develop the original GC idea into an implementable hardware block and at the same time generalize the idea for multiple sample rates and input signals. The author was the main contributor of the paper, Samuli Pietilä was responsible for the word-length and sensitivity simulations, and Harri Valio's role was advisory as the father of the GC concept.

Publication [P5] introduced the gate count estimation method that has been used in this thesis to evaluate the silicon area estimates of the different baseband architectures. The key advantage of this method is that it enables parameterized models of the design to be developed at a very early stage of the project. The models can be refined when more accurate information becomes available. The estimation method was applied to a variety of real designs and its accuracy was verified to be extremely good. The estimation method was developed by the author while he was working on the development of multiple commercial GNSS receivers. The author also wrote the paper. The role of the second author (J. Nurmi) was advisory.

Publication [P6] continued to expand the gate count estimation method of [P5] by adding a method for mapping embedded memories to a gate count number. This enabled comparison of designs that used different amounts of

embedded memories in addition to the digital logic. Another contribution was to apply the gate count estimation on different GNSS receiver baseband architectures. The receiver comparison results of this thesis were presented in this publication. The author was responsible for the estimation method development, receiver comparison including developing the test cases, and writing the paper. The role of the second author (J. Nurmi) was advisory.

1.3.2 Patents

During the work, which was the basis of this thesis, multiple patents have been filed and granted to protect the inventions. The following patents are related to this thesis but are not included as part of it. The author's contribution was to come up with the original idea and develop it into a practical implementation for the listed patents:

Patent US 6,850,558 [11] shows how to time-multiplex the data path of a correlator to reduce the silicon area and power consumption. These correlator optimizations are explained in more detail in Section 4.2.

Patents US 7,283,582 and US 7,471,719 [12,13] disclose the correlator finger functions, discussed in Section 4.2 This is shown to be a way to reduce the silicon area in the correlator-based GNSS receivers.

Patent US 6,909,739 [14] discloses methods to alter the apparent length in time of the Matched Filter using decimation, and the way to perform coherent and non-coherent integration after the MF. These ideas are reviewed in Section 4.3.

Patents US 7,010,024 and US 7,505,511 [15, 16] describe a way to time-multiplex the MF computation by utilizing multiple input shift registers

and multiple reference coefficient registers. The time-multiplexing is explained in Section 4.3 in this thesis.

Patent US 7,852,907 [17] presents a practical implementation of a Group Correlator-based GNSS receiver. It also introduces a reconfigurable GC that allows switching between search and tracking modes in the receiver utilizing a single hardware block. The GC idea was invented by Harri Valio and Samuli Pietilä earlier [18], but the development into a practical implementation as well as the extensions to the original idea were the author's responsibility. The GC is discussed in more depth in Section 4.4.

Patent US 8,391,335 [19] discloses the multiple input and output version of the Group Correlator. This patent is a good example of how a patent publication can obscure the original idea of the invention so that it becomes almost indecipherable. The idea behind this patent is explained more clearly in Section 4.4.

1.4 Thesis Outline

The thesis is composed of an introductory part and six publications. The introductory part briefly introduces the GNSS receiver technologies to the reader and contains a summary of the material presented in the papers.

Chapter 2 shortly discusses the concepts required for understanding GNSS receivers. The GNSS field is already well established so that we will not explain the basic principles in depth. The reader is referred to any introductory GNSS book, such as [20, 21], for these. We will only briefly touch the most important things that are needed for the work presented in this thesis. We start the chapter by giving a quick view of the current GNSS systems. Next, we will take a look on the blocks needed for building a

GNSS receiver and describe what a general GNSS receiver baseband part contains. We will then review the operations that are performed in the receiver digital baseband and the most important performance criteria for them.

Chapter 3 contains an outline of the gate count estimation method developed by the author. The method is later used to compare the baseband architectures using a selection of use cases. This chapter presents a condensed overview of the contents of publications [P5] and [P6]. The chapter begins with a review of the existing literature and then describes the developed method briefly. Next, the mapping of embedded memories into gate count estimates is shortly explained. The chapter concludes with a simple example case which illustrates how the method is used.

Chapter 4 is by far the longest one, discussing three different time-domain GNSS receiver baseband architectures. We start the chapter by describing the oldest, correlator-based architecture and illustrate two different ways for optimizing the correlator block and show how to alter the basic correlator slightly to perform some more advanced operations. Then we will move on to the MF-based design. Again, we start by briefly introducing the MF in the context of GNSS receivers before showing different ways to optimize the MF-based designs by an efficient implementation of the summation, using time-multiplexing, and enabling further signal integration after the MF. The third architecture considered here is the GC, which combines the characteristics of the correlator and the MF. We describe first how the GC can be derived and then shortly discuss realistic implementations. For each of the proposed three architectures, we present the block diagrams of the GNSS receiver basebands, which are used later in the gate count comparison of Chapter 5.

Chapter 5 is largely based on publication [P6] and contains the receiver comparison results of this thesis. We start the chapter by introducing the test cases for which each compared receiver architecture is parameterized. This makes the comparison fair in the sense that all receivers are configured to fulfill the same set of requirements for the comparison. Next, we present the results with some discussion.

Chapter 6 contains the conclusions and some ideas for future work.

2. INTRODUCTION TO GNSS RECEIVER TECHNOLOGY

In this chapter, we will shortly review the background to the thesis. We start with a quick overview of the global navigation satellite system to introduce the field of technology in next Section 2.1. Then, in Section 2.3, we will introduce the GNSS receivers, providing the framework for the subject of this thesis. Finally, Section 2.4 contains a slightly more detailed look on the signal processing tasks implemented in a GNSS receiver.

2.1 Global Navigation Satellite Systems

Global Navigation Satellite System (GNSS) refers to a satellite-based navigation system, which offers a worldwide coverage and provides its users with information about their Position, Velocity, and Time (PVT). The GNSS is based on a set of satellites broadcasting a signal, which enable the users to compute the PVT at the receiver antenna. The users don't need to transmit any information towards the system. A typical GNSS is comprised of three segments: the Space segment, the Control segment, and the User segment. The space segment consists of the satellites which transmit the signals used for positioning. The ground segment includes a master control station, a set of ground antennas for uploading data to the satellites, and a set of monitoring stations covering various regions of the earth. The user segment refers to the user equipment for receiving the sig-

Table 1: *Current GNSS systems [22–27]*

System	GPS	GLONASS	Galileo	BeiDou-3
Nominal Satellites	24 MEO	24 MEO	30 MEO	24 MEO / 3 GEO / 3 IGSO
Operational Satellites *(as of May 2018)*	31	23	4 IOV[a] 18 FOC[b]	3 MEO / 6 GEO / 6 IGSO [c]
Operational	1995	1993	*2020*	*2020*
Owner	USA	Russia	EU	China

[a] In Orbit Validation
[b] Full Operational Capability
[c] BeiDou-2 satellites shown, as no BeiDou-3 satellites are operational yet

nal from the satellites and computing their PVT. There are several GNSS systems in use and in planning. The oldest and best known GNSS system is the United States Navstar Global Positioning System (GPS). The Russian GLObal'naya NAvigatsionnaya Sputnikovaya Sistema (GLONASS) was available next, and the European Galileo and Chinese BeiDou systems came later.

Table 1 shows a list of the most important GNSSs currently available. All of the systems are constantly evolving by improving the system performance through various means such as adding new signals. GPS, GLONASS, and Galileo satellites utilize only Medium Earth Orbits (MEOs), where the satellites have orbital periods of approximately 12 hours. Some of the BeiDou satellites also use Geostationary Earth Orbits (GEOs), where the satellites appear to stay at the same spot above the earth. Some BeiDou satellites are also in an Inclined Geosynchronous Satellite Orbits (IGSOs),

Table 2: *Some properties of current and future GNSS signals on L1 band. The references point to the corresponding ICD documents that contain full details of the signals.*

Signal	**Carrier freq** (MHz)	**Code freq** (MHz)	**Code length** (chips)	**Sec code** (chips)	**Modulation**	**data rate** (Hz)
Original signals						
GPS L1 C/A [28]	1575.24	1.023	1023	-	BPSK	50
GLONASS L1OF [29]	1598.06 –1603.38	0.511	511	-	BPSK	50
Galileo E1-B [30]	1575.42	1.023	4092	-	CBOC	250
Galileo E1-C [30]	1575.42	1.023	4092	25	CBOC	-
BeiDou B1I [31]	1561.09	2.046	2046	20	BPSK	50
Modernized signals						
GPS L1C-D [32]	1575.24	1.023	10230	-	BOC(1,1)	100
GPS L1C-P [32]	1575.24	1.023	10230	1800	TMBOC	-
GLONASS L1OCd [33]	1600.99	0.511	1023	-		250
GLONASS L1OCp [33]	1600.99	0.511	4092	-	BOC(1,1)	-
BeiDou B1C-D [34]	1575.42	1.023	10230	-	BOC(1,1)	-
BeiDou B1C-P [34]	1575.42	1.023	10230	-	QMBOC	100

in which they appear to trace out a small figure-eight shape in the sky. Typically, the GNSS satellites transmit a multitude of signals aimed for different usage. The satellites are currently transmitting on one to three Radio Frequency (RF) bands: L1, L2, and L5. In the past, the L1 signals were intended for civil use, and the L2 signals were designed for military use only. Recently, some civil use signals have been added to the L2 band as well. The L5 band is a recent addition, and its signal structure is the most complex, even if it is also generally available for civil use. Table 2 lists the current and future GNSS signals at the L1 band of frequencies as an example of the signal properties. The details of the GNSS signals

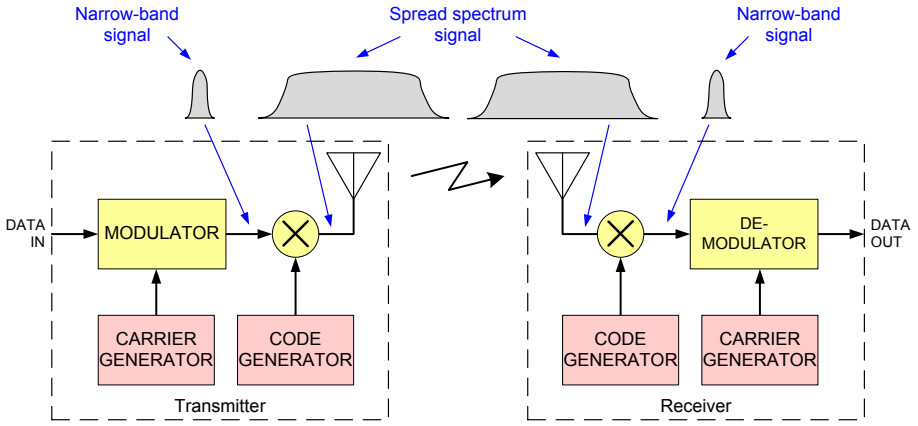


Fig. 2: *Block diagram of a direct-sequence spread-spectrum transmission system.*

are documented in Interface Control Documents (ICDs) published by the owners of the systems. A good source for further information is for example [35, Part B].

2.2 Direct-Sequence Spread-Spectrum Systems

All GNSS systems use Direct-Sequence Spread Spectrum (DSSS) transmission. Good introduction to Spread Spectrum (SS) technique can be found in [36] and an in depth review of its history from [37–39]. There are also several books with a good, in-depth treatment of SS such as [40–42]. In DSSS, the signal is transmitted using a bandwidth that is many times wider than it what would be needed for the data [36]. A conceptual block diagram of a DSSS system is illustrated in Figure 2. In the transmitter, the modulated data is multiplied by a spreading sequence, which typically is a Pseudo Random Number (PRN) sequence, and then it is mixed up to the carrier frequency and fed to the antenna. The receiver multiplies it first by a local oscillator signal to translate the received signal back

to the baseband. Then it needs to multiply the incoming signal by a de-spreading sequence, which needs to be correctly aligned in time with the spreading code in the transmitter. The spreading and de-spreading codes are designed to cancel each other, when the signals are aligned, so that the original modulated data stream will be recovered. If the spreading code is a binary code interpreted as ± 1, then the spreading and de-spreading codes can be identical, which is the typical implementation.

There are two reasons for using DSSS in GNSS. The first is that it enables high resolution ranging [43, 44, Chap. 8], which is crucial for accurate PVT determination. The second is that it enables multiple signals to be transmitted at the same frequency. This is a form of multiple access, called Code Division Multiple Access (CDMA) [45]. The GLONASS system does not rely on CDMA for the transmission from multiple satellites, but it uses Frequency Division Multiple Access (FDMA), where each transmitter uses a separate frequency.

2.3 GNSS Receivers

The function of the GNSS receiver is to acquire and receive the signals from the satellites of each supported GNSS system, and to measure the signal transmit times modulo the spreading duration. Using the transmitted data messages from the satellites and a time-code embedded in the data frames, the receiver can then compute the full signal transmission time and the position of the satellites at that time. With four transmit times it is possible to solve for the four unknowns, namely three coordinates (X, Y, Z) and the reception time assuming that the reception time is common or can be propagated to a common time for all satellites [46]. In a similar fashion by measuring the Doppler frequencies of the received

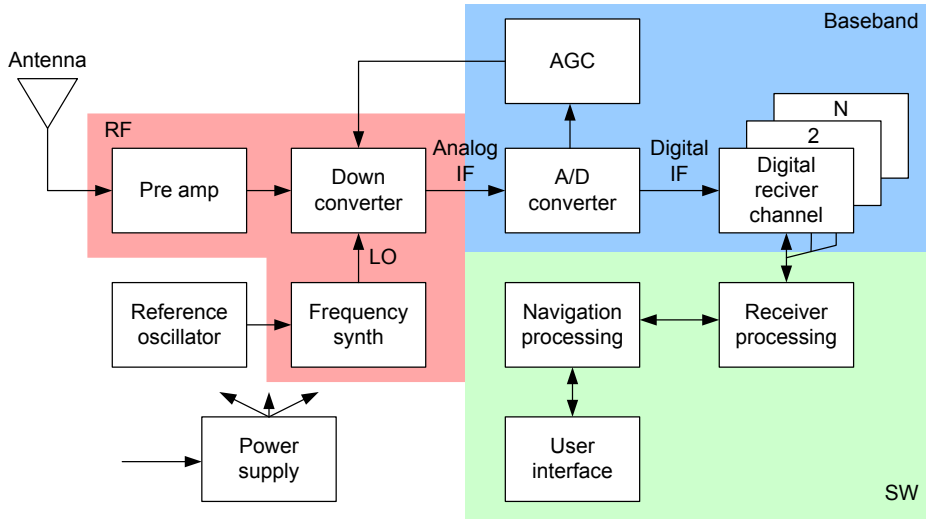


Fig. 3: *Generic GNSS receiver block diagram (adapted from [20]).*

signals, it is possible to solve for the velocity in three dimensions together with the common frequency offset of the local clock of the receiver. This, of course, is a greatly simplified description of what actually happens in a modern GNSS receiver. These calculations are typically performed by the software algorithms running in a microprocessor which is part of the GNSS receiver. There is a vast amount of literature devoted for the PVT calculation and other GNSS algorithms. Good sources for further information include [20, 21, 35].

A generic GNSS receiver is illustrated in Figure 3. The signal is received at the antenna and it is then amplified by a Low Noise Amplifier (LNA) since the signal power is too weak at this point to be directly used. Next, the signal is down-converted from the RF carrier frequency to a lower Intermediate Frequency (IF) for further processing in the baseband. The analog baseband chain also contains filtering and further amplification which is not shown in the figure. The baseband amplification is also ad-

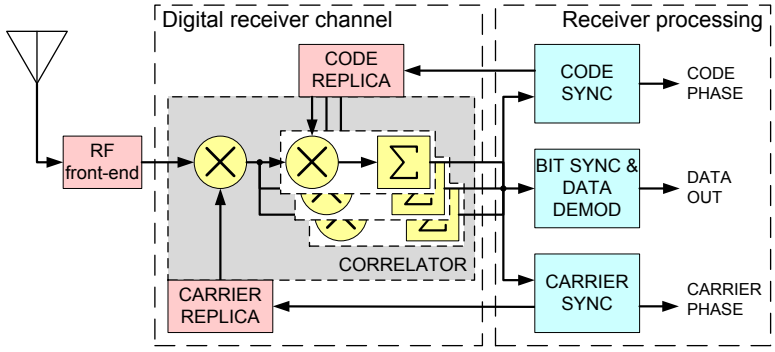


Fig. 4: *Conceptual diagram of GNSS receiver channel structure (adapted from [P6]).*

justable to control the signal level at the ADC. This will ensure that the quantization losses are minimized. As the received signal to noise ratio is low [47], the ADC needs only a small number of bits [48, 49], usually between 1 to 3 [50]. The signal chain until this point contains usually just one path per RF band, as the RF chain bandwidth covers the transmit frequencies of all the satellites in that band. As we need to receive the signals from all of the satellites, the digital part contains one receiver channel for each signal received from each satellite. As each satellite can transmit multiple signals, the receiver might also need multiple channels per satellite. The receiver uses different SW algorithms to control the hardware for receiver control, signal acquisition and tracking. Finally, the navigation processing is responsible for calculating the PVT outputs that the user needs.

In this thesis, we will concentrate on the implementation and optimization of the digital receiver channels. A conceptual block diagram of the receiver channel along with some associated functions is depicted in Figure 4. The digital baseband signal enters the receiver channel, where it is multiplied by a replica of the remaining carrier (including IF, satellite Doppler frequency

and any local oscillator errors) and the locally generated replica of the spreading code. After the multiplication, the signal is accumulated until it is forwarded to the software algorithms for further processing. Mathematically, this function performs a cross-correlation between the incoming signal and the locally generated replicas, and thus it is termed as *correlator*. The correlator is a key component in the GNSS receivers and there are numerous widely different implementations. The correlator itself does not produce the measurements necessary for the PVT computation. The data message transmitted by the satellites is extracted by demodulating the correlator output signal. The message can only be received when the replica code and carrier are aligned with the incoming signal; otherwise, the correlator output is just noise.

In 1988, the first commercial, handheld GPS receiver, the Magellan GPS NAV 1000, was introduced to the market [51, 52]. It had only only one hardware channel in the entire receiver, which was time-shared between the different satellites to facilitate the reception of enough signals for the PVT computation [53]. However, as the digital technology advanced, it soon became possible to implement more parallel hardware channels, which improved the receiver performance [54]. The early receivers were designed for one GNSS system only, and the number of receiver channels was low. With the nominal GPS constellation, there are maximally 11 satellites visible at any place and time [46], so 12 receiver channels became the norm for some time for tracking all visible satellites and one channel tracking a Satellite Based Augmentation System (SBAS) signal. Nowadays it is not uncommon to have several hundreds of channels [55]. This is needed for using more than just one RF band and GNSS and increasing the performance of the receivers.

2.4 GNSS Receiver Signal Processing Tasks

After the incoming signals are down-converted to a lower Intermediate Frequency (IF) and converted to digital discrete time samples, the next step is to process them at the baseband. There are four main tasks to perform:

1. Find the coarse parameters for the local replicas for all signals available
2. Track the incoming signals so that the local replicas stay aligned
3. Receive and extract the messages sent by the satellites
4. Measure the frequency and time of the received signals as accurately as possible

Usually, the functions are performed in sequence, but the last three are continued as long as a particular satellite signal remains available. There are some measures that allow us to determine how well these tasks are performed.

2.4.1 Signal Acquisition

The signal acquisition process always needs to happen first so that we know the parameters needed to generate the local replicas accurately enough for the receiver can start tracking the signals [56]. The signal acquisition is a three-dimensional search problem, where the search limits are known to a certain degree. In a cold start situation [57], the receiver does not have any a-priori knowledge of its own location, the time, the available satellite signals, or their parameters. The system itself gives some limits:

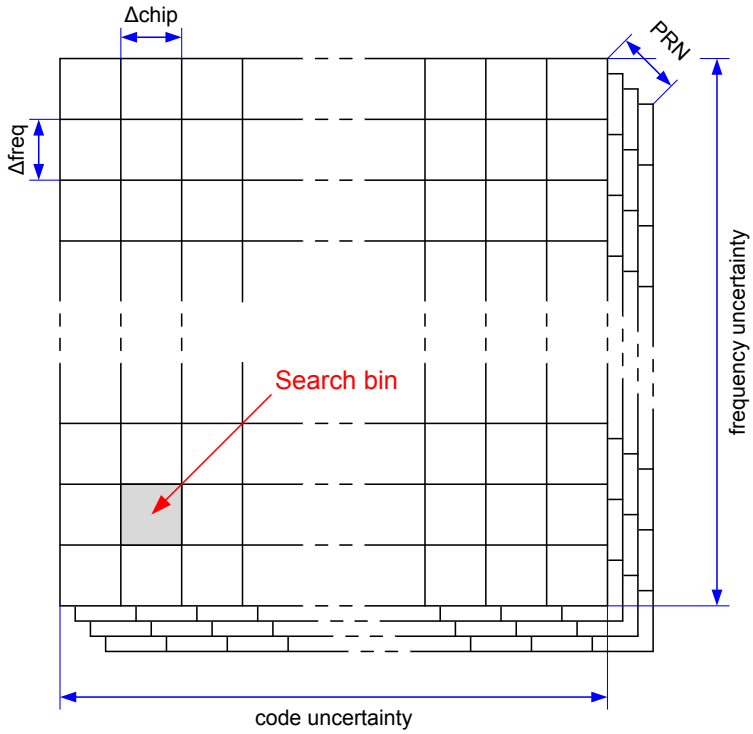


Fig. 5: *Visualization of the GNSS acquisition space (from [P6]).*

There is a finite number of satellites in each system, their orbits and the receiver motion limits provide bounds to the Doppler range, and the local reference oscillator has some known limits of the frequency uncertainty. Only the time is unlimited, but on the other hand, the spreading code is repeating, so we only need to find the signal transmit time modulo the code duration. The search space is divided into discrete points in the three dimensions, where the point distance is set by the search algorithm. This acquisition space is depicted in Figure 5. There are also some other commonly used acquisition scenarios. In the warm start case [57], we do have some a-priori knowledge of the satellite orbital parameters, location

and time, which allow us to predict which satellites are available, and what are their estimated Dopplers. The user motion and the time, within the code period, are still unknown. This results in a bounded search space in two dimensions. The hot-start case [57] adds more exact satellite orbital parameters and time down to a few code chips. Also, in many cases the receiver motion can be guessed accurately enough to make the frequency uncertainty very small.

The important performance criteria for the search phase are the sensitivity or minimum signal level for successful acquisition, the acquisition time, the probability of detection and false alarm, and the frequency and time estimation accuracy for the found signal. These criteria are not independent and improving one usually means making one or more of the others worse.

2.4.2 Signal Tracking

The purpose of the signal tracking is to keep the locally generated replica carrier and code as well aligned as possible to the incoming signal. The art of signal tracking is a well-researched topic in communications, but the GNSS application brings some additional requirements. For general communication applications it is sufficient that the tracking is good enough for the data to be received successfully. For the GNSS receivers, this is often not good enough. The phase of the generated replicas is measured and used for determining the signal reception time. The high precision GNSS receivers are able to determine the position down to the millimeter level or better. At 1.5 GHz carrier frequency, this translates to carrier phase accuracy of about two degrees, which is much better than what is typically required in data receiving. The phase tracking accuracy will become significant for the Bit Error Rate (BER) only when it exceeds 18 degrees [58, Chap 10].

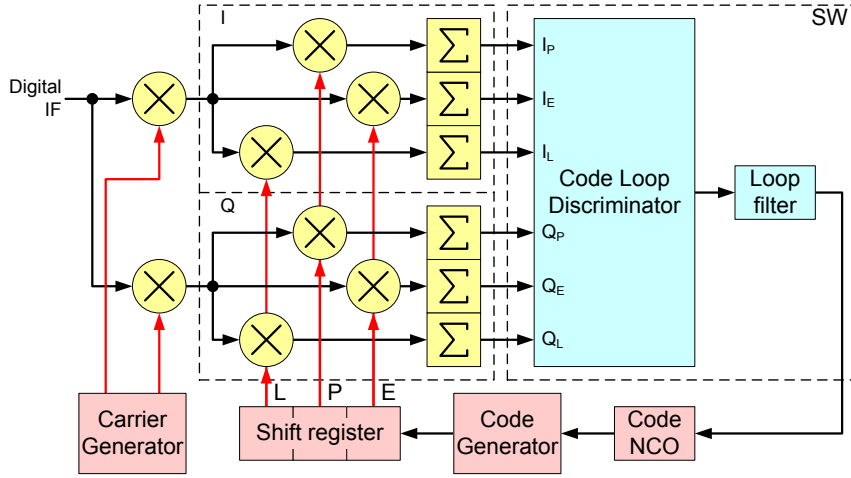


Fig. 6: *Generic block diagram of GNSS receiver code tracking loop (adapted from [20]).*

Typical code tracking in GNSS receivers is implemented using a Delay Locked Loop (DLL) [59], which utilizes two differently timed correlators for advanced (*early*) and delayed (*late*) versions of the replicas. The carrier tracking is implemented with either a Phase Locked Loop (PLL), usually employing a Costas-loop [60,61] or a Frequency Locked Loop (FLL), which is also sometimes referred to as Automatic Frequency Control (AFC) [62]. It uses the precisely timed (*prompt*) correlator, which provides the highest signal level. The PLL is more accurate, but it needs a higher Signal to Noise Ratio (SNR) compared with the FLL [63]. Good references for spread spectrum signal tracking can be found in [40, 41]. GNSS specific good general references are [20, 64]. A typical implementation for code tracking is illustrated in Figure 6, and a carrier tracking implementation is shown in Figure 7.

The main performance criteria for GNSS signal tracking are the sensitivity, the tolerance to receiver dynamics (velocity, acceleration and jerk), and

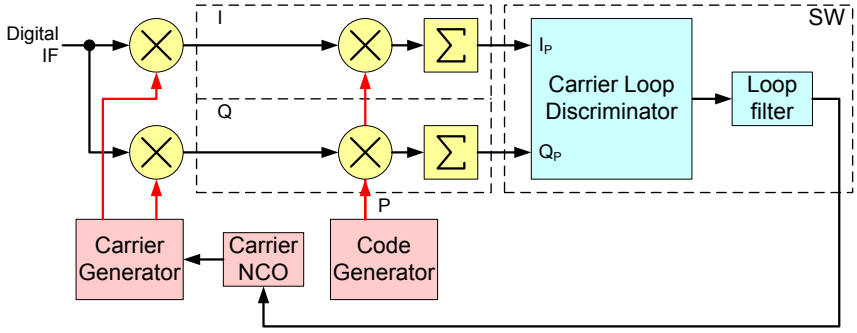


Fig. 7: *Generic block diagram of GNSS receiver carrier tracking loop (adapted from [20]).*

the tracking accuracy. Again, they are not independent, but depend on each other in addition to the tracking loop algorithms used.

2.4.3 Data Reception

Each GNSS satellite transmits data messages containing information about its orbit and clock, called *ephemeris*. This allows the receiver to compute the exact location of each satellite and the signal transmit time. This information is needed for computing the PVT at the receiver. The data messages also contain the rough orbital parameters of the other satellites, called the *almanac*, to allow predicting when they are visible to the receiver. The satellites also transmit some other information, which is not needed for the PVT computation. The exact format and content of the data messages are defined in detail in the Signal-In-Space Interface Control Document (SIS-ICD) of each GNSS system. The data reception uses the prompt correlator output signal, and the data demodulation and message decoding is usually done in the receiver SW [65]. Thus, it is not an important part of the receiver baseband hardware.

The data reception performance is measured by the sensitivity and bit or frame error rates. Typically, the data reception sensitivity is worse than the acquisition and tracking sensitivities, so it determines the cold start sensitivity of the whole GNSS receiver. In some cases e.g. with Assisted GNSS (A-GNSS), the data can be provided for the receiver through a different communication channel, which enables it to operate with weaker signals that would be possible without the assistance.

2.4.4 Measurement Processing

The accurate measurement of the carrier and spreading code timing of the received signal with respect to the receiver local clock is the key to high accuracy PVT determination. The receiver position can be computed from the signal propagation times, and the velocity from the derivative of time, i.e. frequency. The accurate time at signal reception time is a by-product of determining the position. In similar fashion the receiver clock frequency will be solved as a by-product from the velocity computation. There are numerous well known algorithms for PVT determination and the reader is referred to for example Kaplan [66] as an introduction or the GNSS Handbook [67–69] for a more thorough treatment of this subject.

From the hardware perspective, there are two common methods for performing the raw measurements. The first and oldest is to keep track of the phase and frequency of the generated replicas inside the receiver software implementing the signal tracking. This is doable, as the replicas are generated using Numerically Controlled Oscillators (NCOs), whose frequencies are updated at regular intervals. It is then possible to compute their phases at the next update time, and this process can be continued as long as the tracking continues. If the channel NCO update times are identical, then it

is trivial to get the measured phases and instantaneous frequencies at a common time. However, if the channels are sampled at the end of the code periods, which aligns the integration periods accurately with the bits of the satellite data message, then each channel has different update time, and it becomes more tedious task to align the measured values to a common reference time. Implementing the replica phase and frequency sampling in hardware allows sampling the values at a common instant by storing the values at sample registers when a hardware signal is toggled. This hardware measurement method has been used in some receiver implementations, especially those with individual dump for the channel outputs.

3. GATE COUNT ESTIMATION[1]

Design optimization for digital ASICs[2] and IP[3] blocks is easier and more efficient when the design space exploration can be done early in the design process, before the implementation is frozen. Financial feasibility of the design may depend on its manufacturing cost and ensuring rapid time to market will demand knowledge of the design complexity early in the project. Modern digital designs are based on pre-built digital gates for all but the most performance critical parts and silicon area is almost directly proportional to the number of gates. Thus, the gate count is a good metric for the complexity of a digital circuit. The gate count also has a relation to the power consumption via the capacitance accountable to the gates. Furthermore, the testability and the test time are affected by the complexity so it affects to the cost of the chip also via the cost of the testing.

3.1 Earlier Work

The idea of estimating the design complexity using silicon area, gate count, or Field Programmable Gate Array (FPGA) resource usage is not a new one. Several approaches have been proposed in the literature through the years.

[1] The material in this chapter is based on the publications [P5] and [P6].

[2] Application Specific Integrated Circuits

[3] Intellectual Property

Shannon [70] can be credited for being the first to propose the estimation of the switch count for implementing Boolean functions. He proposed that the *upper limit* of the complexity of Boolean functions is proportional to the exponential of the number of inputs. Müller [71] used the same approach applied to the gate count estimation for implementations of Boolean functions.

Kellerman [72] proposed that the area of a function depends only on the number of conditions which must be differentiated by a one or a zero output and presented a formula for the computation of the area estimate based on that idea. Pippinger [73] and Cook et al. [74], amongst others, have studied the relationship between the entropy and the area complexity of Boolean functions. The entropy measure based estimation method was then expanded to the multiple-output Boolean functions by Cheng and Agrawal [75].

The earlier work was based on *randomly generated* Boolean functions with a small number of inputs, and the estimates developed using those greatly overestimated the gate count of real circuits. This problem was pointed out by Nemani and Najm [76], who proposed that typical circuits are far from random in their structure and developed a new *linear measure* of Boolean functions, which is dependent on the complexity of the on and off-sets of the function. Using the new method gave more realistic results for typical circuits. The estimation method was further developed by Büyükşahin and Najm [77], whose work enabled estimation at a higher level of abstraction by using a Boolean network representation of the circuit instead of Register Transfer Level (RTL) level description. In the end, using these kinds of methods still requires the knowledge of the accurate Boolean functions of the logic, which may not be available at early phases of the design. Another approach by Akers [78] was based on representing the Boolean functional

graphically as *Binary Decision Diagrams (BDDs)*. Bryant [79] applied this method on integer multiplication in order to estimate its complexity.

Enzler et al. [80] described a method for FPGA design complexity estimation. Starting from a Data Flow Graph (DFG) or block diagram, they created a characterization vector describing the features of the design, which consists of numbers and word lengths of design building blocks such as adders and multipliers as well as some other block characteristics. The vector can then be mapped into the FPGA resource use, which is similar to the gate count of an ASIC. In addition, they also use the method to generate timing estimates, which is possible to do in FPGA, as the blocks and routing have predictable delays. Their method resembles the one in this thesis, but our method offers more freedom in the characterization since it is not tied to the fixed vector format. Since our method allows selecting the basic functions in which the design is mapped to, it can be tuned to different classes of designs. In our method, it is also possible to increase the accuracy of the estimation incrementally by adding more basic blocks to the estimation process over time.

The references are summarized in Table 3 illustrating the used design description (input). The table also lists the accuracy of the method if it is reported in the publication. For a further introduction to the area estimation methods, Meeuws [82] provides a good introduction to the hardware characteristics estimation techniques in the scope of hardware/software partitioning. As a summary, it is obvious that most of the earlier work has concentrated in finding formulas for mapping a small, combinatorial, and accurately described logic function to an estimate for the hardware complexity. Also, most of the earlier efforts on the gate count and area estimation were aimed to develop the algorithms to be integrated into Computer Aided Design (CAD) tools for building comprehensive frame-

Table 3: *Area estimation references comparison (from [P5])*

Ref	Input	Cost	Accuracy	Notes
[70]	Boolean functions	switches	n/a	Theoretical work
[71]	Boolean functions	gates	n/a	Theoretical work
[72]	Boolean functions	diodes	15%	for random logic
			33%	for realistic logic
[74]	Boolean functions	diodes	n/a	Theoretical work
[73]	Boolean functions	gates	n/a	Theoretical work
[75]	Boolean functions	gates	30%	
[76]	Boolean equations	gates	22%	Based on on/off sets of function
[77]	Boolean network	gates	24%	
[80]	Block diagram / DFG	FPGA cells	12%	Intermediate vector representation
[81]	VHDL	FPGA cells	3.5%	Intermediate CDFG representation
[P5]	Block diagram / DFG	gates	4.0%	Parameterizable / extendable models

works like Abdelhalim et al. in [81]. In contrast, the method presented in this thesis aims to provide a tool for early stage estimation for architecture exploration at a high level of design abstraction, which would work on complex and realistic designs.

3.2 Estimation Method Summary

Development of integrated circuits is very complex and time-consuming process and optimization is more efficient when done at higher design abstraction levels. However, no such systematic methods existed for ASIC design, and architects needed to resort to rules of thumb, their experience

in the field and ad-hoc rough estimations. The method proposed here will offer a way to early estimates of the gate counts of incomplete designs quickly and efficiently. The method is especially suited for area estimation of new data flow modules.

The successful use of the proposed method requires only gate size information about a few key gate types in the standard-cell library that will be used in the synthesis of the design. In many cases, default gate size values could be used for the primitives with only a minor loss of accuracy. Different alternative designs can be compared successfully against each other with good accuracy as long as the same sizes are used for the primitives.

3.3 Parameterized Model Creation

The proposed gate count estimation method is based on a bottom-up modeling approach, but the design will be processed from top to down. The developed model of the design for the gate count estimation can be parameterized so that it will be easy to explore the effect of parameter changes to the silicon area. The models are built from parts at three abstraction levels:

1. *Primitive library*, which includes elements that correspond directly to standard-cell gates of the implementation technology.
2. *Basic block library*, which is built from a pre-defined, but extensible set of functional blocks such as adders, multipliers, and registers. The basic blocks are re-usable and parameterized in such a way that they can be used in modeling of many different designs.
3. *Design level models*, which are specific to the design under estimation.

Table 4: *Primitive library for gate count estimation (from [P5]).*

Primitive	**Size**	**Notes**
2-input NAND (NAND2)	1 gate	1-gate area definition
Inverter (INV)	1 gate	Slightly larger than a minimum inverter
Register bit (FF)	5 gates	With synchronous reset
Full-adder (FA)	5 gates	
2-input MUX (MUX2)	3 gates	
Half-adder (HA)	3 gates	
2-input XOR (XOR2)	3 gates	

The primitive library contains typical standard cells, which should be available for any chosen implementation technology. The gate sizes of these key cells, which form the primitive library, are the only required information needed for the proposed method. The primitive library, which was used in [P5] and [P6] is listed in Table 4, where the default gate counts for the cells are also given.

The basic block library contains parameterized, low level building blocks, which are used in most designs. It is also possible to add sub-blocks, which are used many times in the design, into the basic block library to allow their re-use. To get the best advantage of the new basic blocks, they need to be parameterized for maximal re-use and developed carefully to maximize the accuracy of the estimation. For the basic blocks, it is usually a reasonable first guess to base them on their minimal area implementation. It is possible that, for example, some very high-speed designs might need a different implementation with inserted pipeline stages. The basic building blocks

Table 5: *Examples of basic block library elements from [P5].*

Function	Size
Adder	$\mathcal{G}(\text{add}(N)) = N \times \mathcal{G}(\text{FA})$
Subtractor	$\mathcal{G}(\text{sub}(N)) = N \times (\mathcal{G}(\text{FA}) + \mathcal{G}(\text{INV}))$
Increment by 1	$\mathcal{G}(\text{inc}(N)) = N \times \mathcal{G}(\text{HA})$
Absolute value	$\mathcal{G}(\text{abs}(N)) = N \times (\mathcal{G}(\text{XOR2}) + \mathcal{G}(\text{HA}))$
Multiply (uns.)	$\mathcal{G}(\text{mult}_{\text{uu}}(N, M)) = N \times M \times (\mathcal{G}(\text{FA}) + \mathcal{G}(\text{AND2}))$
Multiplexer M-to-1	$\mathcal{G}(\text{mux}(N, M)) = N \times (M - 1) \times \mathcal{G}(\text{MUX2})$
Register	$\mathcal{G}(\text{reg}(N)) = N \times (\mathcal{G}(\text{FF}) + \mathcal{G}(\text{MUX2}))$

are usually modeled with a set of basic primitives, but they could also contain other basic blocks as parts. Some representative, simple basic blocks are listed in Table 5, where the notation $\mathcal{G}(\mathcal{B})$ means the gate count $\mathcal{G}$ for a block $\mathcal{B}$. There can also be more complex basic blocks such as register banks, signed multipliers, mixers, and so on. They are developed using the simple basic blocks and the primitives as parts.

3.4 Estimation Method Flow

The gate count estimation method is split into two distinct phases. The first one is building a model of the design and this process is illustrated in Algorithm 1. The process starts by creating a data-flow graph or a block diagram of the design to be estimated. This will be done in top-down fashion so that a sub-block diagram is made for each block at the top level. At each level, we can use new sub-blocks, basic blocks or primitives from the libraries. The process is repeated until we have just basic blocks or

Algorithm 1 The model creation (from [P5])

```
1: procedure CreateModel(block)
2:     Divide to sub-blocks as needed
3:     for all sub-blocks do
4:         CreateModel(sub-block)
5:     end for
6:     Define parameters as desired
7:     Create a data-flow graph / block diagram
8:     Map remainder to basic blocks or primitives
9: end procedure
```

primitives left at the lowest level. Next, the design can be parameterized from bottom-up so that the lower level parameters can be derived from those at the next level up. For example, one common parameter is the word length for all data elements.

The next phase uses the model with a suitable set of parameters to compute the estimate, as shown in Algorithm 2. The process simply sums up the gate count of all instantiated sub-blocks, basic blocks, and primitives recursively applying the parameters as given by the user at the top level.

There are a few caveats in the use of this method. The first one is the accuracy of the primitive library. In most cases, the default values will suffice, but there could be cases where the relative sizes of the primitives will dictate which of the two architectures leads to a smaller size. This can happen e.g. when the size of the register cell compared with the sizes of combinatorial cells is smaller or larger, and the two architectures differ in the number of register bits needed. In such cases it is important to use the correct values for the chosen implementation technology during the

Algorithm 2 The model evaluation (from [P5])

```
1: function EvalModel(block, params...)
2:     count ← 0
3:     for all B ← instantiated sub-blocks do
4:         count ← count + EvalModel(B, params...)
5:     end for
6:     for all B ← instantiated basic-blocks do
7:         count ← count + EvalModel(B, params...)
8:     end for
9:     count ← count + ∑ (gate count of primitives)
10:    return count
11: end function
```

process. Fortunately, it is also possible to change the primitive cell sizes during the process and just re-evaluate the estimate. The modeling phase does not need to be repeated. The other shortcoming is that the presented methodology does not take the routing area into account. The result of the estimate is really the gate count value and not silicon area. If a physical size estimate is needed, the routing area needs to be estimated. In some cases, one could also use the average gate count per mm^2 value given by the silicon vendor.

3.5 Gate Count Estimation Accuracy

In [P5], we have shown that the average gate count estimation accuracy was 4%. The accuracy was computed simply as the average of the absolute values of the relative errors between the estimated gate count given by the model and the actual gate count as reported by the logic synthesis

tool used. In computing the accuracy, we used blocks from two different GNSS receivers and a Digital Signal Processing (DSP) processor core. The gate counts of the blocks ranged from 20 to 250 kilo-gates. The designs included some embedded memories, but these were excluded from the accuracy analysis. A detailed description of the architectures of these blocks can be found in [P5]. In the paper, we also had a number of smaller examples, which were compared with actual gate counts from the synthesis tool. These smaller cases had an average error of 3.2%. The gate counts of these smaller cases ranged from 0.4 to 12 kilo-gates.

The accuracy evaluation method follows its intended use. We started from a high level description and created the estimation from it. Then using the same description, a VHSIC Hardware Description Language (VHDL) implementation was developed and synthesized. Finally, the gate count given by the estimate was compared against the gate count of the actual implementation. Only high level description of the design was used for the gate count estimation model development. For the GNSS receivers, we used the design specifications. For the VS_DSP, the input document was its user's manual [83]. The actual gate counts were obtained from the implementation documentation of the corresponding ICs.

There were some outliers when comparing the estimates with the actual numbers. In one case, where the estimate was too optimistic, we determined the error to be due to some unmodeled interfacing and Built In Self-Test (BIST) logic for the embedded memories that was included in the actual implementation. In another case, where the estimate was too high, we found out that the reason was that the modeling did not take into account all the optimizations that were possible in the implementation. Interestingly, these outliers happened in sub-blocks of a larger design and the top level

Table 6: *Properties of memories used to build the SRAM model in Table 7.*

Parameter	Minimum	Maximum	
Type	Single-port, high speed		
Ports	1	1	
Size	5120	491520	bits
Word length	8	32	bits
Words	512	15360	
Parallel words	4	16	
Rows	32	1024	
Columns	40	512	

estimation accuracy was extremely good. This shows that the modeling errors tend to cancel out when applying the method for large designs and the overall accuracy is very good.

3.6 Memory Mapping to a Gate Count Estimate[4]

Embedded memories are used for data storage for most reasonably complex designs, as their data storage density is much better than what can be achieved with register cells. In the case of architecture exploration, the amount of memory is often different for the different alternatives. It is thus very important that the embedded memories can also be included in the gate count estimates. We have chosen to map the area of the memories to a gate count value based on the number of gates per area information which we can obtain from the silicon vendor. The memory model needs to be created for each type of memory and for each implementation technology separately.

[4]The memory gate count estimation method was originally presented in [P6].

In conjunction of the implementation of a GNSS receiver, we had obtained from a silicon vendor a set of parameters for variably sized and configured embedded memory blocks in a 65nm CMOS technology. For the purposes of building a memory model for the GNSS receiver comparison, we selected a subset of 168 single-port Static Random Access Memories (SRAMs) from this set, as listed in Table 6. As it can be seen, the range of the memories used as the model basis was very wide, so the resulting model should fit reasonably well to a wide range of uses.

The simplest way to model the memory area is to assume that each storage element (bit) consumes a certain amount of area, and there is a fixed area overhead. This can be expressed as:

$$\text{gates} = (a1 \cdot b + \mathcal{A}) \times \mathcal{C}, \tag{1}$$

where b denote the number of bits in the memory; the model parameter $a1$ models the area of a single memory bit; $\mathcal{A}$ is the memory overhead area; and $\mathcal{C}$ is the gate density constant (in gates/mm^2) given by the silicon vendor for that particular technology. The model parameters can now be estimated by fitting this model to the memory area information for the differently configured memories from the silicon vendor. The average accuracy computed as average of the absolute values of the relative errors of the resulting *area* estimation (i.e. without the gate density taken into account) was 10%. The errors for differently sized memories are detailed in Table 8.

A better memory model can be based on a typical layout of embedded memory shown in Figure 8. The gate count estimate for the memory is then given by:

$$\text{gates} = (h0 + h1 \cdot r) \times (w0 + w1 \cdot c) \times \mathcal{C}, \tag{2}$$

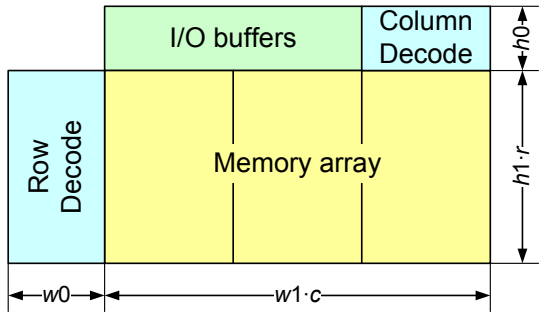


Fig. 8: *Conceptual floorplan of an embedded SRAM (from [P6]).*

Table 7: *Memory area parameters used in the evaluation*

Parameter	**Value**
w0	53.3 μm
w1	1.266 μm
h0	26.3 μm
h1	0.613 μm
kgates/mm^2	500

where r and c denote the rows and columns in the memory array, respectively. The constants $h0$ and $w0$ model the height and width of the peripheral area of the memory block containing the address decoding circuitry and I/O buffers; $h1$ and $w1$ model the height and width of a single memory bit. The resulting estimated parameter values from this SRAM model using Equation (2) are shown in Table 7. Using this revised model, the obtained overall average error of the area estimation was 4%. Again, the detailed errors for differently sized memories can be found in Table 8. As shown in the table, the modeling error tends to get smaller with increasing size. Interestingly, the 130-kbit memories have a somewhat larger error for the second prediction model of equation (2), equaling that of the simpler model in equation (1). The errors of these two models get closer to each

Table 8: *Memory modeling errors for different memory sizes.* ***size*** *denotes the memory size in kilobits;* ***err1*** *and* ***err2*** *are the error of the first (eq. 1) and second models (eq. 2) respectively as the average of absolute values of the relative errors; and* ***N*** *is the number of different configurations.*

size	err1	err2	N
5	34.1%	9.4%	12
10	17.7%	6.7%	24
20	10.9%	3.5%	36
40	8.5%	3.0%	36
64	8.6%	2.9%	32
128	4.2%	2.2%	12
130	5.9%	5.9%	8
320	2.0%	1.6%	4
480	2.4%	2.0%	4

other with increasing memory sizes, but for smaller memories the more complex model is clearly better. A still more accurate model could be obtained if the number of parallel words in the memory array would be taken into account, but it would complicate the use of such a model especially at the early estimation phase, where such details would be unknown. Another idea could be to use the multiplexing factor in the model, but make the model itself select the smallest memory configuration automatically.

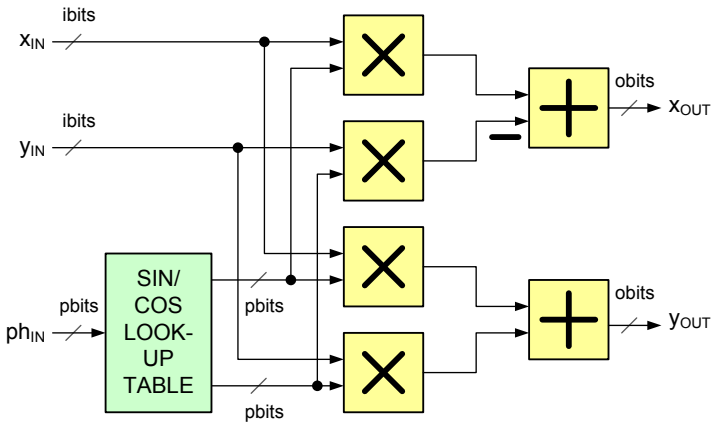


Fig. 9: *Schematic picture showing the complex mixer (from [P5]).*

3.7 Small Example[5]

To give some idea of using our gate count estimation method, we will show how it can be applied to a complex mixer model. The complex mixer is a common block in many communications related designs. The mixer multiplies two complex numbers together using four real multipliers and two adders. One input of the mixer in this case is the phase of a complex sinusoidal signal, which is commonly generated by a table look-up. This implementation is shown in Figure 9. The parameters of the mixer model are:

ibits – Input data word length

pbits – Input phase word length

obits – Output data word length

The development of the gate count model for the mixer is very straightforward for the arithmetic part. The signed multiplier (MULT_SS), the adder

[5]This example is taken from [P5].

(ADD), and the subtractor (SUB) basic blocks can directly be used here. We can also easily count the required number of elements from the block diagram.

The look-up table modeling poses a trickier problem. In our case, the look-up table is implemented as combinatorial logic instead of a Read Only Memory (ROM), due to pipelining constraints and its small size. The table contains a full wave of $\sin(\omega)$ and $\cos(\omega)$. As we only wanted to get a quick model for the look-up table, we did not go into great lengths in order to optimize its size. The most straightforward method to create a model for it is to write a few simple VHDL models of the table with a different number of input and output bits and fit a suitable model to the synthesis results. We can model the table as a number of Boolean functions. According to Shannon [70], the complexity of a Boolean function is bounded by an exponential of the number of inputs, so we use an exponential function as the basis of our model. Since we know that there is a certain amount of redundancy in the data we will use another parameter in the model to represent the amount of redundancy. Our model of the gate count of the table uses a function of the form:

$$f(x, \alpha, \beta) = \alpha \cdot x \cdot \beta^x, \tag{3}$$

where α and β are the model parameters and x is the word length of the input and output. By fitting our model with empirical data obtained by synthesizing the table for a few cases, we can determine suitable values for the two parameters. One would expect that the parameter β would be close to 2 and α would reflect the redundancy factor and two outputs. Using a small set of synthesized tables resulted in $\alpha = 1.4$ and $\beta = 1.8$ as the parameter values. Since sine and cosine tables are common in many

Algorithm 3 Complex mixer gate count estimation model (from [P5])

```
1: function COMPLEX_MIX(ibits, pbits, obits, tech)
2:     % Complex multiply
3:     mult  ← MULT_SS(ibits, pbits, tech)
4:     add   ← ADD(obits, tech)
5:     sub   ← SUB(obits, tech)
6:     mix   ← 4 * mult + add + sub
7:     % Look-up table
8:     lut   ← 1.4 * pbits * 1.8^pbits * tech.NAND2
9:     return mix + lut
10: end function
```

communication applications, it is natural to make a basic building block from the table for future uses of it. The complete model is shown in Algorithm 3.

In [P5] we have compared the model with the synthesized VHDL implementation and the average of absolute values of the relative errors was found to be 2.4%.

4. BASEBAND HARDWARE OPTIMIZATION

In this chapter, we will review three GNSS receiver baseband architectures, and show how they could be optimized in various ways. There has been little discussion about such optimizations in the open literature except in some obscure patent publications. First, we will shortly introduce the parts necessary in all GNSS receiver baseband implementations. The first architecture we will present here is the correlator-based receiver architecture, as it is the oldest and best known [84]. We will see how the basic structure can be tuned to fit various optimization goals. Next, comes a treatment on the use of a Matched Filter (MF) as the correlating device in a GNSS receiver. We will conclude the chapter with an overview of the Group Correlator (GC), which is a recent addition to the GNSS receiver architecture scene.

4.1 General GNSS Receiver Baseband Considerations

Before jumping to the three receiver architectures discussed in this thesis, we can shortly review what a generic GNSS receiver baseband needs to contain in order to perform its operation. The core function in all GNSS receivers is the computation of the cross correlation between the incoming signal and the local replicas for each signal we wish to receive in parallel. In addition to the correlation, every receiver needs a couple of common

blocks. Since they are common, we can concentrate on the correlation implementation in the architecture optimization question. Nevertheless, also the common blocks should be carefully optimized to get the best possible implementation.

The GNSS receivers were discussed in Section 2.3. A generic receiver was illustrated in Figure 3 (on page 18) and the receiver channel in Figure 4 (on page 19). Each receiver channel needs generators for code and carrier replicas besides the correlator. Usually, there will be more than one delayed version of the PRN code generated for each channel, but only one carrier replica is used in each channel. In some cases, the replica generators could be shared between the channels by e.g. time-multiplexing the hardware. The replica generators use a NCO to generate the replicas at the right rate, which the SW algorithms control by updating the frequency of the oscillator. In case of the carrier replica, multiple bits of the NCO phase value are used to drive a lookup table, which provides the sinusoidal replica value. For the code generation, only the overflow indication from the NCO is used to drive the PRN sequence generation. The PRN code generation depends on the GNSS specification. The code can be generated quite easily with a Linear Feedback Shift Register (LFSR) generator for GPS Coarse/Acquisition (C/A), GLONASS, and BeiDou B1I, but Galileo E1-B and E1-C codes can only be generated from a table, which is provided in the Galileo SIS-ICD [30]. Publications [P5] and [P6] provide some examples of the replica generation blocks.

The correlator usually provides coherent integration up to one code period length. Thus, the correlator will produce outputs at a rate, which is in the order of a millisecond. This is a low enough rate to continue the integration using the receiver software, but if there are very many receiver channels with many different delayed code versions, the rate might still become too

high for the SW to cope with. In such cases it would be advantageous to follow the correlator block by a hardware-based integrator block. By continuing the coherent integration, the bandwidth of each channel becomes smaller while the integration time gets longer. For acquisition, this reduces the frequency bin size and the search would require more bins to cover the same frequency range. It is possible to have a second carrier removal stage before the coherent integrator block, which can split the correlated signal to multiple frequency bins, which are integrated individually. This saves replicating the resource hungry correlator block for each frequency bin to be searched. As the apparent signal frequency is not fully predictable for extended periods of time and the transmitted data bits are not fully predictable, it becomes impossible to continue the coherent integration beyond approximately one data bit period [85]. If the sensitivity requirements cannot be met with the coherent integration, GNSS receivers commonly employ non-coherent integration, which discards the signal phase by taking a norm[1] and then continues to integrate the resulting real valued signal. The post correlation integration is widely known in GNSS literature [86]. Van Diggelen [87] has presented formulas for computing required coherent and non-coherent integration times to meet given sensitivity requirements. The integrator block implementation is illustrated in Figure 10. It was also shortly presented in publications [P5] and [P6].

4.2 Correlator

In this section, we discuss the various implementation possibilities for a correlator-based GNSS receiver baseband. We will also describe methods

[1] Common implementations of a norm include the magnitude of the complex signal, and the sum of squares of the real and imaginary parts.

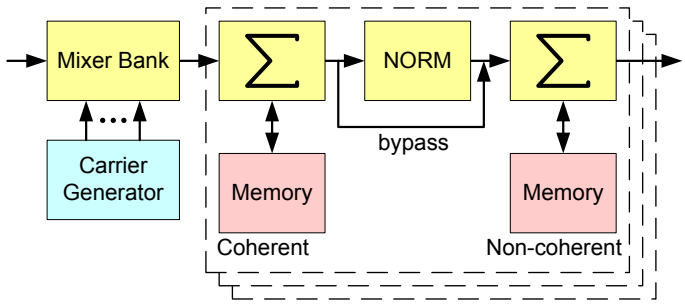


Fig. 10: *Common integrator block used for the gate count comparison (adapted from [P6]).*

for its optimization and show how some advanced functionality can be implemented with minimal hardware complexity.

4.2.1 Correlator Functionality

As the definition of the actual meaning of a correlator as a building block of a GNSS receiver has been somewhat unclear, we will define it here. The term *correlator* in this thesis means a GNSS receiver building block which performs a cross-correlation operation[2] between the received signal and a single version of a locally generated replica of it. The output of the correlator is defined as:

$$C(\tau, t, f) = \int_{t}^{t+T} r(t)c(t-\tau)e^{-j2\pi f(t-\tau)}dt, \tag{4}$$

or in discrete time (where T_s is the sample period):

$$C(\tau, n, f) = \sum_{k=0}^{N-1} r\left((n+k)T_s\right) c\left((n+k)T_s - \tau\right) e^{-j2\pi f((n+k)T_s-\tau)}, \tag{5}$$

[2]The cross correlation of two periodic signals $x(t)$ and $y(t)$ is defined as $R_{x,y}(\tau) = 1/T \int_0^T x(t)y(t-\tau)dt$, where T is the period of the signals

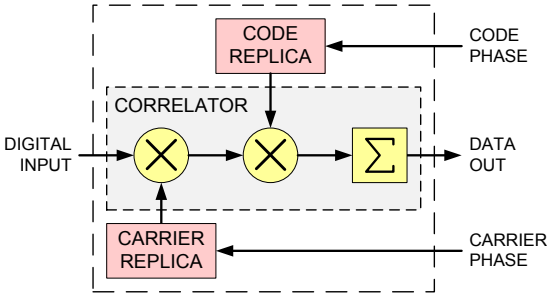


Fig. 11: *Conceptual diagram of GNSS receiver channel structure (adapted from [P6]).*

where $r(t)$ is the received signal, $c(t)$ is the locally generated replica of the spreading code, and $e^{-j2\pi ft}$ is the local replica of the (residual) carrier. Usually, the received signal is complex valued at a low or zero IF, the carrier replica is complex valued, and the spreading code typically contains only the values ± 1. A signal flow graph of a correlator is shown in Figure 11.

All recent GNSS receivers use several correlators for efficient operation. Usually, these are organized as several receiver channels. Each channel is responsible for handling the reception of one signal from a single satellite. At least four channels are needed to enable the simultaneous reception of signals from at least four satellites [46], but often the number of channels is much larger to allow the simultaneous reception of more satellites for better performance. Each receiver channel typically has multiple correlators, which share local replica generator blocks for the code and carrier and which use differently delayed versions of the replica code. These are often called *correlator fingers*. Modern GNSS receivers have at least 3 correlator fingers per channel: two (early and late) are used for code tracking, and one (prompt) is used for carrier tracking and data demodulation (see subsection 2.4.2). Many modern receivers employ a greater number of correlator fingers to implement advanced signal processing algorithms

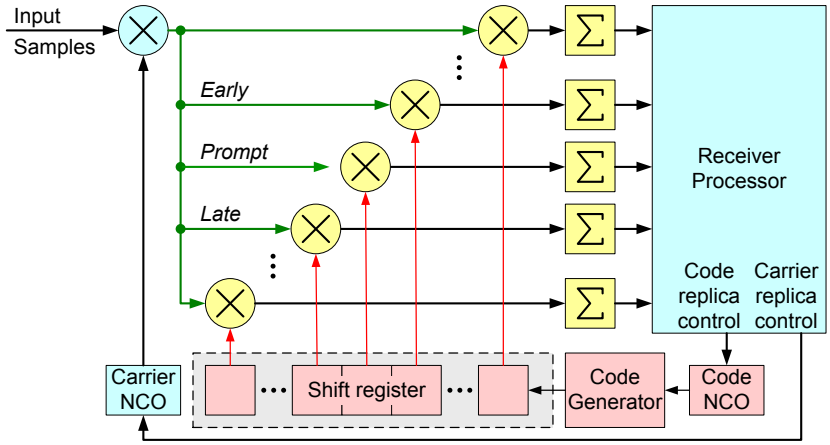


Fig. 12: *Traditional correlator channel architecture (From [P1]).*

for reducing multipath effects [88] or improving performance of receiving modern GNSS signals [89, 90]. More correlation fingers will also improve the signal acquisition performance [91]. The correlator output is further processed by receiver software for signal acquisition, detection and tracking [63]. An example of a correlator channel in a GNSS receiver is shown in Figure 12. The most common order of the operations, as shown here, is to multiply the incoming signal first by the carrier replica as this more complex multiplier is only needed once, and then by the code replica, which reduces to a conditional sign reversal of the signal.

The GNSS industry has used the number of correlators as a figure of merit in marketing. Unfortunately, due to competing on who has the biggest number, the actual definition of the term *correlator* has been deviating from the original, strictest definition. This has led to the idea of counting *equivalent correlators*. This count, in its simplest form, is defined as the

number of cross-correlation operations between the received signal and different variations (in delay and/or frequency) of the locally generated replica within a defined time [92].

4.2.2 Word Length Optimization

The GNSS receiver operates on signals that mostly contain noise. This makes them special in the word length optimization sense. Typically, the input to the baseband has only a few bits [65], which also makes the baseband blocks different from those used in many other signal processing applications, where the signals contain more bits and have a higher SNR.

In conventional DSP applications, consecutive samples of processed data are highly correlated and integration increases the signal amplitude directly proportional to the integration time. Also, depending on the frequency of the signals, the mean value of the integrated signals can change. For GNSS applications this is not true. There, the processed samples can be modeled as additive white Gaussian noise with zero mean, so that integration increases the standard deviation or amplitude proportionally to the square root of integration time, and the mean stays as zero. Care must be taken however, as the SNR can become positive at some point in the processing chain. If this happens, the word length optimization needs to be done as in conventional receivers. In the early stages of processing, it is possible to limit the maximum amplitude using hard limiters, as long as the number of clipped samples remains reasonably low. Thus, in many cases we can limit the signal values to, for example, 3 times the standard deviation (σ) without any noticeable degradation to the performance. Publication

[P1] contains more detailed treatment of the word length optimization considerations for correlator implementation with some examples of the possible savings.

4.2.3 Time-Multiplexed Correlator Architectures[3]

Time-multiplexing allows the reduction of parallel hardware elements related to stateless operations, but offers no reduction of storing state variables. Usually time-multiplexing is implemented by increasing clock frequency directly proportionally to the number of parallel operations that are multiplexed. Time-multiplexing is usually employed to save silicon area at the expense of a slight increase in power consumption due to higher clock frequencies. By carefully examining and ordering the operations performed in correlators, it is possible to implement time-multiplexing without really increasing the clock frequency.

The spread-spectrum signals require large input bandwidth and high sample rates in the input stages, yet the output of the correlator is usually delivered at 1 kHz or even slower rate. We also need to repeat the same processing for multiple correlator fingers as well as multiple receiver channels. All this make GNSS receivers attractive for time-multiplexed implementations.

In traditional correlator signal processing flow as shown in Figure 12, we first multiply the incoming signal by the carrier replica since the input word length is small, and thus the silicon area of the mixer area is also the smallest. Next, the correlator fingers are implemented by multiplying the mixer output with the differently delayed versions of the code replica with

[3]The content of this subsection is a summary of Publication [P1]. These ideas were first published in the patent US 7,010,024 [11].

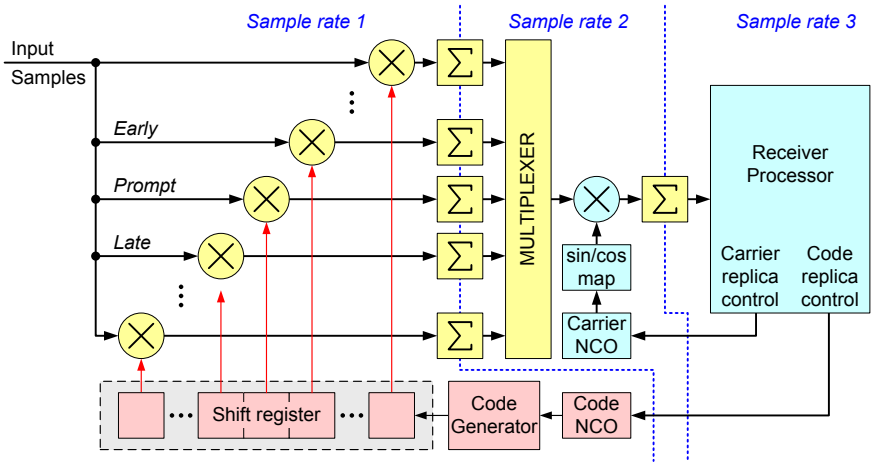
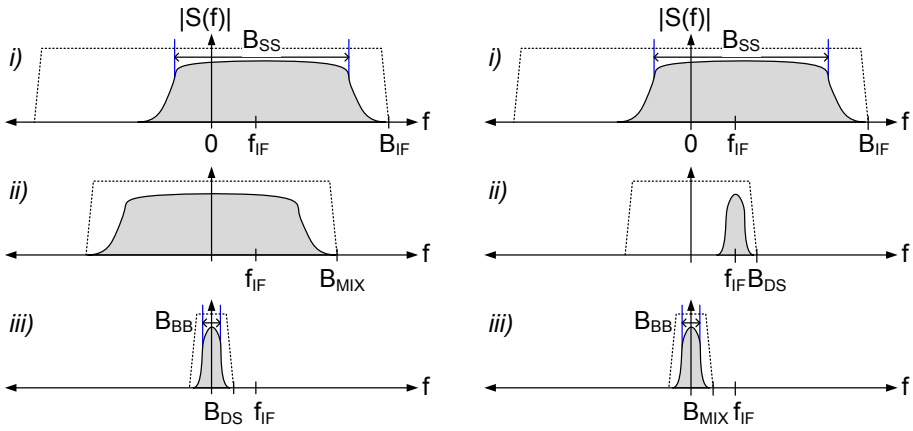


Fig. 13: *Re-arranged correlator with code mixer placed in front of the carrier mixer (from [P1]).*

values of ± 1, which is also a simple operation. We repeat this hardware for each correlator finger in the receiver channel. The carrier mixer and the associated carrier replica generator are the most complex elements of this implementation and they run at the input sampling rate.

From the spectrum plot Figure 14a, we can see that the required signal bandwidth is reduced only after the multiplication by the code replica (also called de-spreading), and we need to maintain the high sample rate until the final integration. Assuming a low IF, we we can first multiply the received signal by the code replica, which will result in a low bandwidth signal. We can then decimate the de-spread signal down to a lower sampling frequency before performing the carrier replica mixing. The sampling frequency at the carrier replica mixer still needs to be high enough to pass any center frequency variations of the incoming signal as well as the residual IF. Thus, the final integrators are still needed, but they are running at a much lower rate. This arrangement then allows multiplexing the mixers and final



(a) *Spectrum within traditional correlator. i) incoming signal, ii) Signal after carrier mixer, iii) signal after multiplying with code replica*

(b) *Spectrum within rearranged correlator. i) incoming signal, ii) signal after multiplying with code replica, iii) Signal after carrier mixer.*

Fig. 14: *Signal spectra within different correlator implementations(adapted from [11]).*

integrators for all correlator fingers as well as multiple receiver channels while keeping the operating frequency constant. The time-multiplexed correlator structure is illustrated in Figure 13 and the spectrum of the signal at different stages in this re-arranged correlator are shown in Figure 14b.

4.2.4 Advanced Correlator Functionality[4]

The number of correlator fingers per channel has increased through the times and modern GNSS receivers tend to use a large number of them,

[4]The advanced correlator functionality discussed in this subsection was first published in the patent US 7,283,582 [12], and was also presented in Publication [P1]

Table 9: *Generation of the example strobe shapes for discriminators in Figure 15. The notation $r[C \pm n]$ denotes replica delayed or advanced by n chips. The* Simple strobe *is also called narrow correlator [95]. (From [P1])*

Strobe	**Generation**
Simple	$r[C-1] - r[C+1]$
Bipolar symmetrical	$r[C-2] - 2r[C-1] + 2r[C+1] - r[C+2]$
Asymmetrical	$r[C-2] - 2r[C-1] + 2r[C+2] - r[C+4]$

as modernized GNSSs use more complex signals and advanced algorithms require additional fingers:

- One Binary Offset Carrier (BOC) tracking algorithm [93] uses two early / late fingers per channel.
- The strobe correlator [94] can be used to improve tracking accuracy in multi-path cases. Example strobe cases are shown in Table 9, and corresponding discriminator shapes are illustrated in Figure 15.
- In many cases advanced functionality can be achieved by adding or subtracting correlator outputs.

Unfortunately, correlator fingers are relatively expensive to implement in HW due to the large amount of repeated functionality. The correlator is a linear system, so adding correlator outputs is equivalent to adding code replicas before multiplying with the input. The resulting structure is depicted in Figure 16. We use the term *correlator finger function* to mean using a linear combination of differently delayed code replica versions as the replica in a correlator finger. We can save silicon area using correlator finger functions instead of operating on correlator finger outputs. Some example cases of this are shown in Table 10. In all cases, the code generator

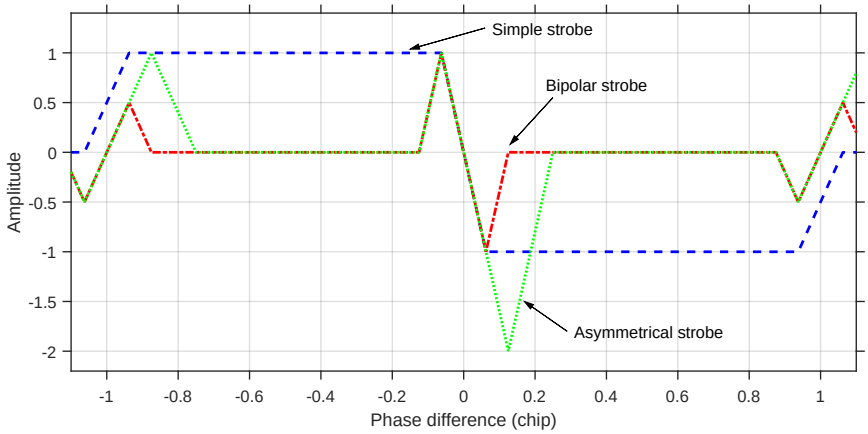


Fig. 15: *Examples of strobe correlator discriminator characteritics (from [P1]).*

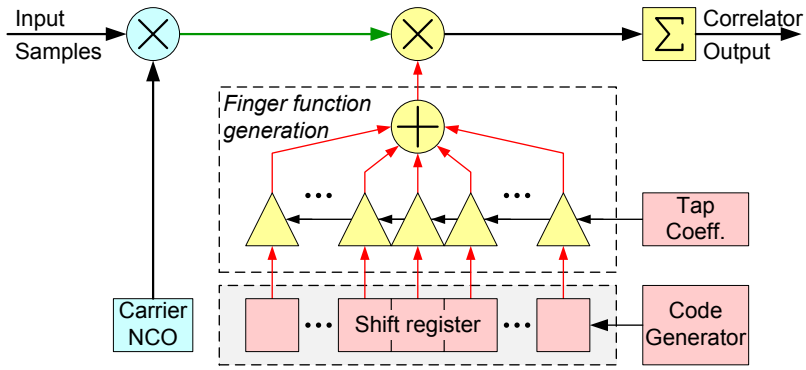


Fig. 16: *Correlator finger function example implementation (Full) (from [P1]).*

produces 16 differently delayed versions of the replica code. The total number of **fingers** is shown in the table together with the finger function enabled fingers shown in parentheses. The first case is a normal correlator, the second is a special version, where each finger can independently select the replica from the delay line, and the two latter use finger functions, which enable creating the discriminator directly in the correlator. The **gain selection** column lists the available choices for the gain of each finger

Table 10: *Finger function implementation examples (from [P1]).*

Case	Fingers	Gain sel.	Adder	gates	diff
Simple Correlator	5 (0)	*n/a*	*n/a*	4364	-
Delay selection	5 (5)	0/1	OR	4674	+7.1%
Simple functions	3 (2)	0, ±1	2-bit	4074	-6.6%
Strobe functions	3 (2)	0, ±1, ±2	3-bit	4144	-5.0%

function element. The **adder** column shows how many bits are needed for the finger function adder or if an *OR* gate was used instead. The **gates** number show the estimated gate count for each implementation, which all use otherwise identical structure. Finally, the **difference** column shows the difference in a percentage compared with the simple correlator.

4.2.5 Correlator-Based Receiver for Gate Count Comparison[5]

The correlator-based baseband architecture used for the receiver comparison is illustrated in Figure 17. This architecture uses time-multiplexing for minimizing the gate count. Time-multiplexing allows using a single data path to process multiple channels (PRN replicas) and/or carrier replicas. The different code phase fingers are processed in parallel within the correlator block. We assume that the carrier and code generators for the receiver are maximally time-multiplexed to optimize the receiver gate count. We will ignore the fact that this arrangement will lead to optimistic area estimation and it might be difficult to realize. We should also note that, when this receiver is used in the acquisition mode, we will get an enormous

[5]Publication [P6] contains the details of the correlator architecture selected for gate count comparison.

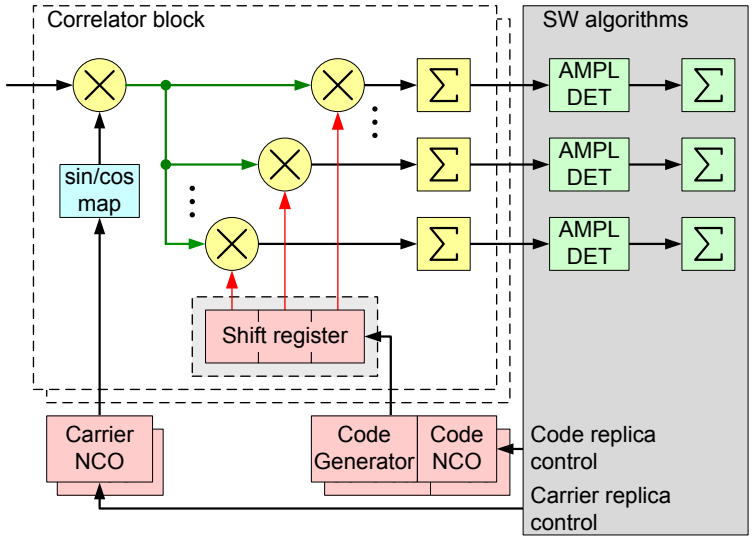


Fig. 17: *Correlator-based GNSS receiver architecture used for the gate count comparison (from [P6]).*

amount of outputs within each coherent integration interval. It would be possible to reduce the processing load in SW by adding a post-integrator block after the correlators to process the outputs in hardware. The post-integrator is not included as a part of the correlator in the area evaluation in the gate count comparison between the different architectures.

4.3 Matched Filter

At the turn of the century, it became feasible, especially in the economic sense, to implement full code searches in GNSS hardware. A further driver for this was the FCC e-Call (E911) mandate [1] with strict requirements for the acquisition time and sensitivity. The cold start problem with rapid, full code uncertainty search had been an uninteresting problem before, as

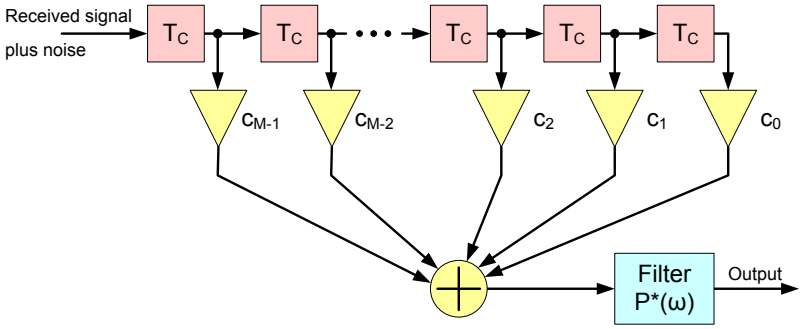


Fig. 18: *A tapped delay line implementation of a matched filter for an M-chip PRN sequence (from [P3]).*

performance in this scenario had not been viewed as a critical or important performance parameter for GNSS receivers, which had typically been used in the continuous tracking mode. The cold start had been thought as an infrequent event since receivers typically operated long times in tracking the signals while the user was navigating along a route.

4.3.1 Matched Filter Functionality

Matched Filters (MFs) [96], [40, pp. 815–832] are devices which continuously compute the correlation between a known reference signal and the received signal and produce the maximal output when the correlation is the strongest. By definition, they are optimum detectors for signals embedded in Additive White Gaussian Noise (AWGN). The structure of a MF is shown in Figure 18. The incoming signal is first captured in a tapped delay line and the output of each tap is multiplied by the corresponding coefficient from the reference signal before being added together. The summing function is followed by a passive filter matched to the basic pulse shape of an individual chip. The last filter can be ignored for rectangular chips.

Looking at the structure we can assess the difficulty of implementing the MF in hardware. The MF length is usually equal to a single PRN code period, as it can then easily compute a full code length correlation. Using a shorter MF, we will suffer from a worsening of cross-correlation performance, which makes it more difficult to separate between signals from different satellites. We could also detect an incorrect timing of the incoming signal due to the non-ideal autocorrelation properties of partial PRN codes, which would lead to failed acquisition [97]. The PRN code lengths for current civil GNSS signals intended for direct acquisition vary between 511 (GLONASS) to 4092 (Galileo). The MF must produce one output for each incoming sample, which means that the input and the output need to be updated at least at the chipping rate, which is between 511 kHz (GLONASS) and 2.046 MHz (BeiDou). For GNSS applications we can usually assume that the tap multiplications are trivial to implement, and there are few bits per sample at the MF input stage. The summation element needs to add as many inputs as there are taps in the filter within the time between two output samples, which turns out to be in the range of $4 \cdot 10^9$ operations/s. There are three possible ways to solve this problem:

- Pipeline the MF implementation by storing partial results.
- Fully pipeline the summation by using a transposed filter structure.
- Implement the summation in a smart way.

The first two options need to store intermediate MF output values, which requires a larger number of storage elements. This is wasteful for hardware area and power consumption. Furthermore, using area efficient storage such as Random Access Memory (RAM) is not feasible since we need parallel

access of all the values at the same time, which is not possible with RAM. Trying to time-multiplex the MF calculation logic only makes the problem harder. Thus, we will take a look at the third option.

4.3.2 Reduction Adder Tree[6]

We will start by considering the problem of adding N 1-bit numbers. This can later be expanded to multi-bit numbers. The idea behind the method builds upon the ideas of carry-save adders [98] and tree-adders [99].

We can build a network of one-bit full-adders in a tree structure which achieves relatively low delay in the order of $\mathcal{O}(\log_3 N \cdot \log_2 N)$ times the delay of single full-adder delay. The $\log_3 N$ term is due to the number of inputs that need to be added to get the Least Significant Bit (LSB), and the $\log_2 N$ term is due to the final carry propagation chain from the final adder producing the LSB of the result to the Most Significant Bit (MSB) of the result. The number of bits in the result is equal to $\lceil \log_2 N \rceil^*$. The generation of the reduction adder network is a relatively straightforward process and is fully explained in [P3]. The resulting structure for adding 15 inputs is shown in Figure 19. It is possible to show that the number of elements we need to implement an N-input adder using single bit full-adders is:

$$N_{\text{add}} = \sum_{i=0}^{\lceil \log_2 N \rceil} \frac{N - 2^{i+1} + 1}{2^{i+1}} = N - \frac{1}{N} - \log_2 N \approx N - \log_2 N \qquad (6)$$

It is easy to automate the generation of the adder network with a simple program, which can generate e.g. a VHDL netlist for the reduction adder.

[6] The implementation described in this subsection was described by the author in [P3]. It is believed to be the original product of the author developed in 1999.

* $\lceil x \rceil$ is the ceiling of x, i.e. the smallest integer n such that $n \geq x$.

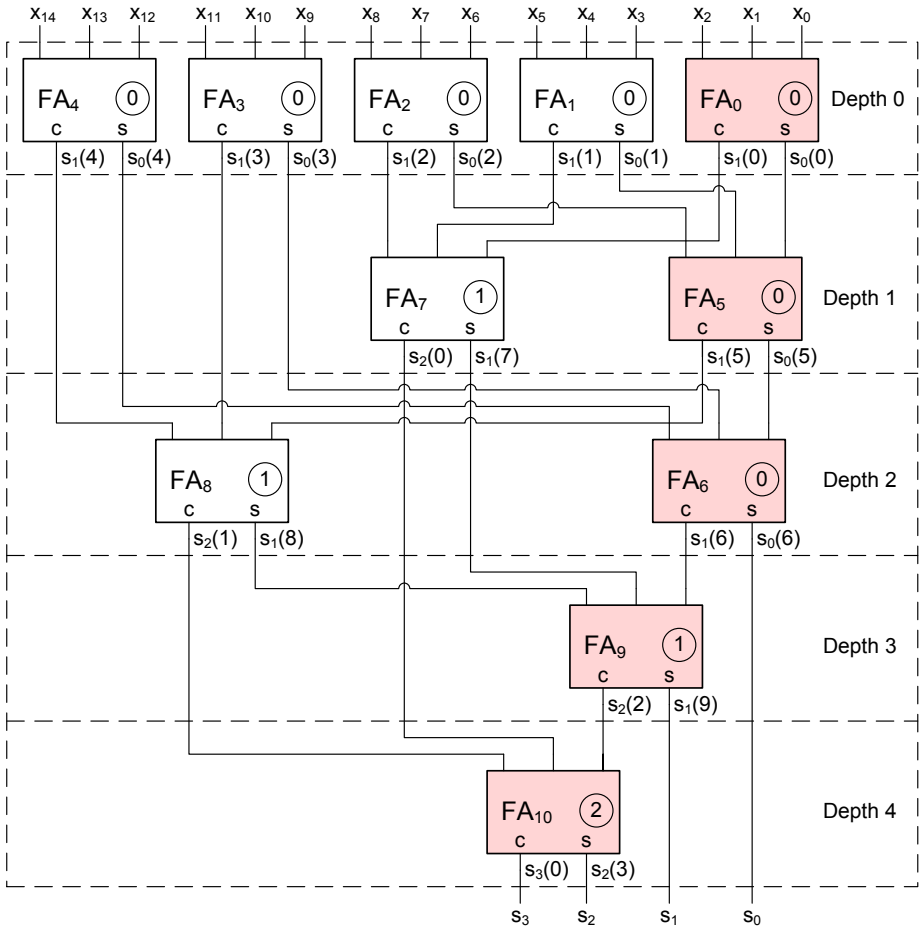


Fig. 19: *15-input binary reduction adder (bitcount) (from [P3]).*

For the MF implementation, we should consider the input and the coefficients being signed, and thus the interpretation of the input to the adder needs to be considered. A useful interpretation would be that a bit value of '1' would mean a match between the input and the corresponding coefficient, and '0' would mean no match. Thus, we will get the number of successful comparisons by adding the 1-bit comparison results together.

However, it is possible that we have a total match of the PRN code to the input signal, but the sign of the input data is just reversed. In this case, we get zero successful comparisons. Also, the worst-case match occurs when exactly half of the comparisons are wrong. However, in this case, the result from our adder would actually be $N/2$. For the original interpretation of the inputs and coefficients the results would be N, $-N$, and 0, respectively. To accommodate this we need to multiply the result by 2 and subtract N from it.

The preceding discussion was only about adding 1-bit inputs. Multi-bit inputs can be handled by two different ways:

- We could simply extend the algorithm for generating the adder tree to consider multiple input bit weights while assigning the adder elements.
- We can use time-multiplexing for processing each input bit weight at a time, and then include a post processing stage which combines the individual 1-bit sums.

The latter method also suggests a way of handling larger additions by smaller summation blocks.

4.3.3 Input Multiplexing with 1-bit MF Core[7]

It is possible to use the proposed summation block as a building element for a system with either larger input word length or, equally well, multiple inputs. We can easily handle multiple inputs in a time-multiplexed way using the structure illustrated in Figure 20. The input signals arrive in

[7]The basic idea for this structure was first presented by the author in [P2], and also patented as [15]

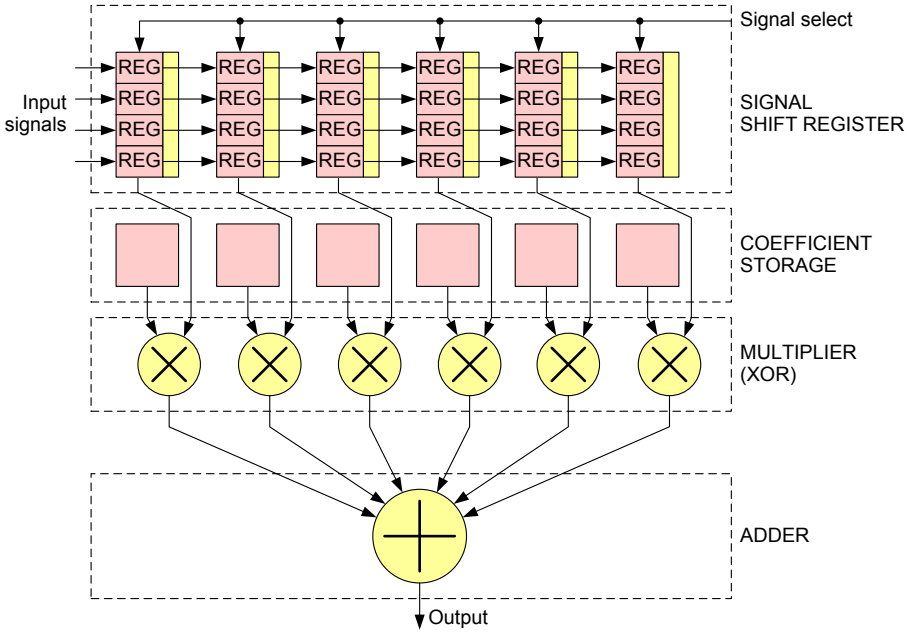


Fig. 20: *Multiplexed MF input structure (adapted from US 7,010,024 [15]).*

parallel to the shift registers, and after each sample period, we calculate the MF output for all parallel inputs in sequence. We multiply each input with the same coefficients, but this structure could also be extended to cover multiple reference signals as well [15, 100]. To use the same structure for multi-bit inputs, it would be advantageous to use a number representation where negation is done by inverting all bits, as this eases the multiplication by the ± 1 values of the spreading code.

We know that any non-negative valued integer n can be represented as a linear combination of $M = \lceil \log_2 n \rceil$ powers of 2:

$$n = \sum_{i=0}^{M-1} b_i 2^i, \qquad b_i \in \{0, 1\}, \tag{7}$$

where $(b_{m-1}, b_{m-2}, \ldots, b_0)$ is the M-bit representation of the number. To extend the representation to cover negative numbers, we have a few alternatives, e.g. sign-magnitude, two's complement, and one's complement [101]. By using one's complement representation, we can negate a number by simply complementing all of its bits. One's complement numbers can be represented as linear combinations of the bits by assigning a weight of $-(2^M - 1)$ to the highest bit. We can now write our signed numbers as:

$$n = \sum_{i=0}^{M} b_i w_i, \qquad b_i \in \{0, 1\}, \quad w_i = \begin{cases} 2^i & \text{for } i < M, \\ -(2^M - 1) & \text{for } i = M \end{cases} \tag{8}$$

Now, applying the MF to the signed M-bit numbers can be written as:

$$\begin{aligned} y(n) = \mathrm{MF}(x(n)) &= \sum_{k=0}^{N-1} x(n+k)c(k) \\ &= \sum_{i=0}^{M} \widehat{\mathrm{MF}}(x_i(k))w_i \end{aligned} \tag{9}$$

Where $\widehat{\mathrm{MF}}(x)$ is a modified MF utilizing one bit coefficient multiplication. This can be implemented in a time-multiplexed way as shown in Figure 20, but the output values need to be collected from M consecutive outputs of the MF hardware block. Also, the output is represented using one's complement notation with *multi-bit* digits and must be transformed to a normal binary representation for further computations.

4.3.4 Integrating the MF Output Signal[8]

We can improve the signal detection statistics by integrating the correlated signal for longer duration. The integration can be either coherent (normal integration) or non-coherent (integrating the norm, e.g. the absolute value, of the signal). The spread spectrum modulation used in GNSS utilizes a periodic spreading code. By using a MF which is matched to the spreading code, i.e. whose length equals the code period, the outputs from the MF separated by a multiple of the code period will correspond to correlation results with identical code offset positions. It can be shown, that adding up the MF outputs which are separated by one code period is equivalent to performing correlation over multiple code periods. Similarly, we can integrate the MF outputs non-coherently for each code offset after using a norm operation on the MF outputs. This allows performing parallel long correlation for all of the searched code phases. As the output rate of the MF is equal to its input rate, we will not need to run the post-MF accumulation at a rate greater than the MF outputs become available. The structure of such post-MF integration is illustrated in Figure 21.

The post-MF integration is an efficient way to improve the acquisition sensitivity of the MF-based system. It is also possible to put a post-MF mixer bank before the coherent integration to allow searching for the signal from a wider frequency range than the bandwidth after the coherent integration would otherwise allow. This is more efficient than performing the correlation itself multiple times. It should be noted, however, that the MF itself will limit the usable bandwidth approximately to the reciprocal of the code period.

[8] The patent US 6,909,739 [14] by the author describes the method described in this subsection.

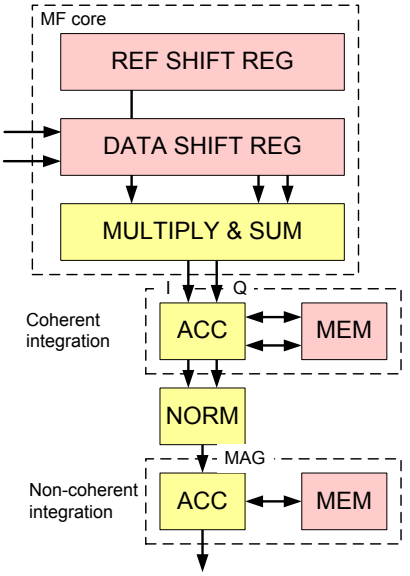


Fig. 21: *Post-MF integration structure (adapted from US 6,909,739 [14]).*

4.3.5 Input Decimation to a Multiple of the Chipping Rate[9]

The MF is matched to a fixed length signal and if the signal is sampled, then it will be matched to the reference signal of a fixed number of samples. The frequencies of the GNSS signal transmitted by the satellites will be subject to Doppler shift due to the satellite motion with respect to the receiver and the sampling frequency might be slightly different from its nominal value. These effects will change the duration of the spreading code slightly from the nominal value. With the post-MF integration, we can use very long integration times, and eventually it will happen that the code phase of the received signal will shift with respect to the local time base so

[9]The idea shown in this section have been published and patented by the author in US 6,909,739 [14]

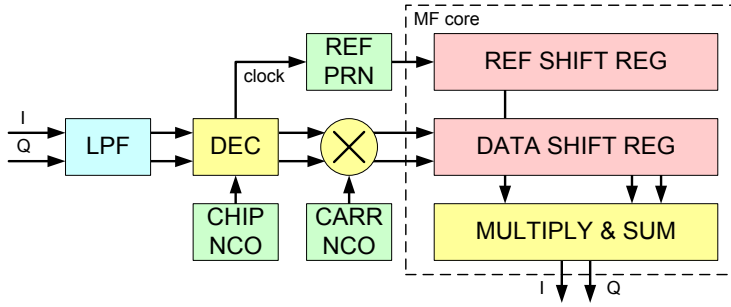


Fig. 22: *Block diagram of MF input decimation to match incoming signal frequency (adapted from US 6,909,739 [14]).*

much that the timing of the MF outputs will no longer match the timing at the beginning of the integration with respect to the timing of the input signal. This will ruin the post-MF integration, as the code offsets will no longer line up to the same storage elements.

Fortunately, it is possible to remedy the situation by altering the sampling frequency slightly so that it will again match to the apparent received signal code frequency. A practical implementation has been proposed by the author in [P2] and US 6,909,739 [14] as depicted in Figure 22. The incoming signal is decimated to an integer multiple of the chipping rate before feeding it in to the MF by an integrate-and-dump filter which is driven by a NCO delivering exact chipping rate (or integer multiple of it), it results on average an exact match of the length of the incoming chips to each MF tap. This allows the MF to be kept in sync with the incoming signal. There are two advantages of this setup: it allows further integration of the MF outputs, and it also allows signal tracking by dynamically adjusting the NCO frequency.

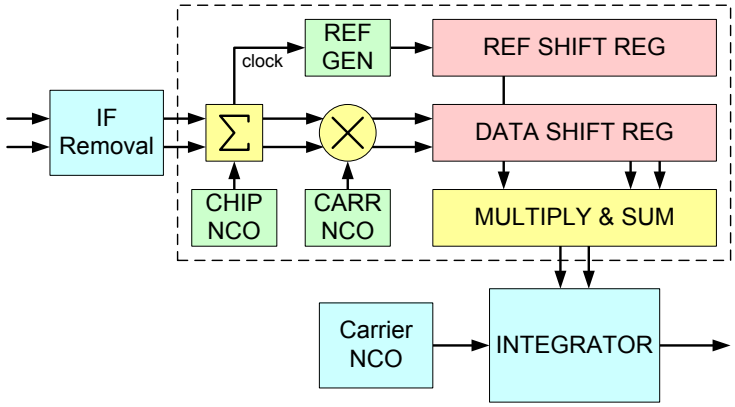


Fig. 23: *MF-based GNSS receiver architecture used for the gate count comparison (from [P6]).*

4.3.6 MF-Based Receiver for Gate Count Comparison[10]

The MF-based receiver architecture used for the gatecount comparison is illustrated in Figure 23. This architecture is based on the one originally published by the author in the patent US 6,909,739 [14]. It is using a common code generator which has the ability of freezing the code generation during MF operation after it has been loaded into the MF coefficient registers. Even if the original version was not intended for signal tracking, this can be easily implemented by using suitable receiver software and enabling the bypassing of the magnitude computation in the post-MF integrator to allow complex valued MF outputs for the tracking algorithms. The carrier mixer and the adjustable MF input rate provide means of keeping the MF timing aligned to the received signal. For multi-bit inputs, we use time-

[10] Publication [P6] contains the details of the MF architecture selected for gate count comparison.

multiplexing of the MF inputs to save computational hardware and we need multiple clock cycles to produce one full complex valued MF output. By extending the time-multiplexing, it is possible to also use multiple reference codes as explained in US 7,010,024 [15]. Since the bandwidth of the MF is limited and we only use one common input stream, it is not practical to use the code multiplexing for other use cases than the cold-start acquisition scenario. Thus, we will use this possibility in the gate count comparison only for the acquisition case.

4.4 Group Correlator

The correlator and the MF have traditionally been the two main categories of hardware for performing time-domain correlation in DSSS receivers. In the correlator, the replica code is multiplied with the incoming signal one sample at a time and the results are accumulated sequentially until a desired number of samples have been processed. If cross-correlation at different time offsets need to be computed in parallel, the correlator structure needs to be replicated accordingly. On the other hand, the MF always computes a correlation over all possible time offsets and outputs them sequentially. It is not very efficient in computing cross correlation for just a single time offset. The Group Correlator (GC) offers a method of computing cross correlation which fits in between the correlator and MF in flexibility and parallelism. The GC was developed to produce a flexible correlation processing unit for combined GPS and Galileo receivers which would offer high sensitivity acquisition while minimizing silicon area especially for assisted and standalone modes of operation.

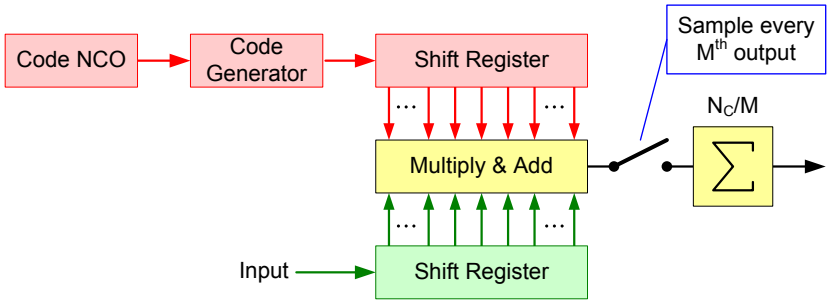


Fig. 24: *Block diagram of a basic serial-parallel correlator (from [P4]).*

4.4.1 Derivation of the GC structure[11]

It is easier to understand how the GC operates if we follow through its derivation from a traditional correlator. First, we can add Shift Registers (SRs) to the correlator to delay the incoming and replica samples. We can then see that the correlator outputs are identical as long as there are an identical number of delays in both data paths. Also, it is mathematically equivalent to multiply and add several values from corresponding SR taps together in parallel and then add them together whenever the shift registers have been updated with fresh data. Each input and reference sample will be used in the correlation only once. We call this structure *serial-parallel correlator*. It is shown in Figure 24.

It is now possible to compute more than one correlation result for differently delayed versions of the replica code in parallel by utilizing the single multiply and add block in a time-multiplexed fashion. If the shift register length is denoted by M, then this can be done up to M times using different time-offsets. We will need to perform the final accumulation of the results separately. We can use a holding register to freeze the reference code during

[11] The derivation of a working hardware structure from the GC idea [18] was first presented by the author in [P4], and also patented as [17]

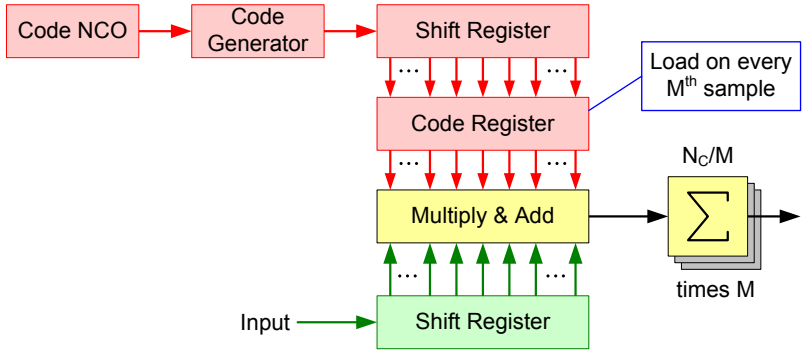


Fig. 25: *Block diagram of a modified serial-parallel correlator for multiple correlation results (from [P4]).*

the computation of the correlations while a shift register is being loaded with fresh reference code samples, and then periodically reload the holding register. If each of these results is now accumulated N_c/M times, where N_c is the code length, we have M full code length cross-correlation results covering M different consecutive time-offsets. This structure is depicted in Figure 25.

Looking Figure 25, we can see that it resembles a MF, but there are some quite significant differences. Traditionally, the length of the MF is one code period, which makes the MF bandwidth rather small. The bandwidth is approximately $1/T_{MF}$, where T_{MF} is the length of the MF in time. This requires us to remove the residual carrier (IF + Doppler) before it can be fed into the MF. The pre-correlation carrier removal usually increments the number of bits needed in the MF in order to avoid additional losses due to re-quantization, thus making the HW even more complex. On the other hand, the parallel correlation part in the serial-parallel correlator can be made reasonably short, so that it has relatively high bandwidth. The output sample rate (for each parallel channel) is lowered by the number of samples summed in this part. Thus, the residual carrier removal could

now be shared between the parallel channels while using a lower update rate than if used before the first correlation stage. The other notable difference to the MF is the continuous updating of the replica code in the serial-parallel correlator, which readily allows signal tracking. As the whole reference code is loaded into the taps of the MF, the replica code is not usually updated continuously.

We can now extend the idea of the serial-parallel correlator by constructing correlator hardware that shares much of the hardware while making it possible to operate on more than one GNSS signal. If we add shift and code registers and replica generators for each N_P GNSS signal to be received, we can cross-correlate in input with N_P code replicas at M different code phase offsets (time offsets). We can also use the same idea to increase the code phase coverage by adding more code (holding) registers that get their inputs from the previous code register. By using N_R code registers we can now cover MN_R code phase offsets with bandwidth equal to covering only M offsets. The same principle can be further extended to also utilize more than a single input shift register. The resulting GC architecture concept is illustrated in Figure 26.

4.4.2 GC Implementations[12]

While it is always possible to compute all possible cross-correlations in the GC, a big advantage in implementation can be achieved by carefully arranging the connections of the building blocks such that each replica code shift register (*Code SR*) and code holding register (*Code R*) is associated with a certain input sample rate as the code shift registers and the sample

[12]The ideas presented in this subsection were presented by the author in [P4], and also patented as [17, 19]

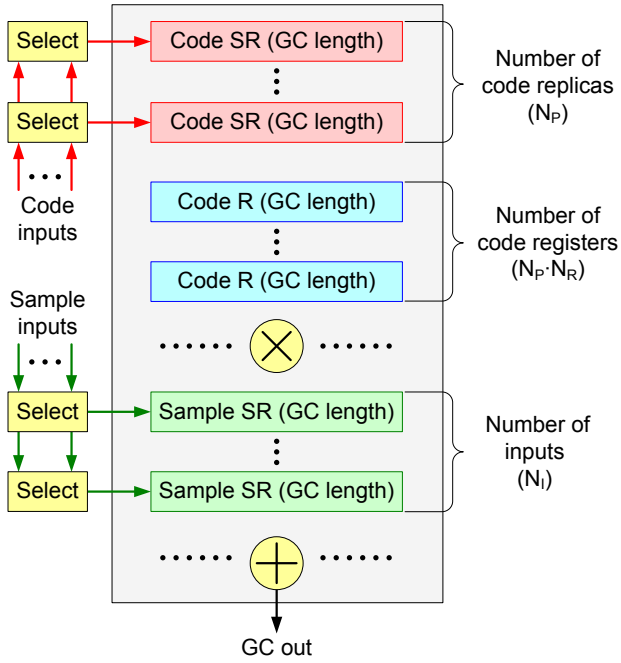


Fig. 26: *Conceptual block diagram of the group correlator(from [P4]).*

shift registers (*Sample SR*) need to be updated at the same rate to maintain the time synchronization. The clock rate of the GC should be equal to an integer multiple of the least common multiple of the input sample rates to allow deterministic clocking. In order to minimize the required clock frequency, the GC should produce outputs at every clock cycle. The GC must always be followed by an integration block, which individually sums up the partial correlation results computed by the GC-core and computes the full length correlation results.

The biggest advantage of the GC architecture can be achieved by realizing that it is possible to make the number of code replica registers and the sample rate assignment programmable and still produce correlation results at a constant rate. This will greatly simplify the design of the blocks that

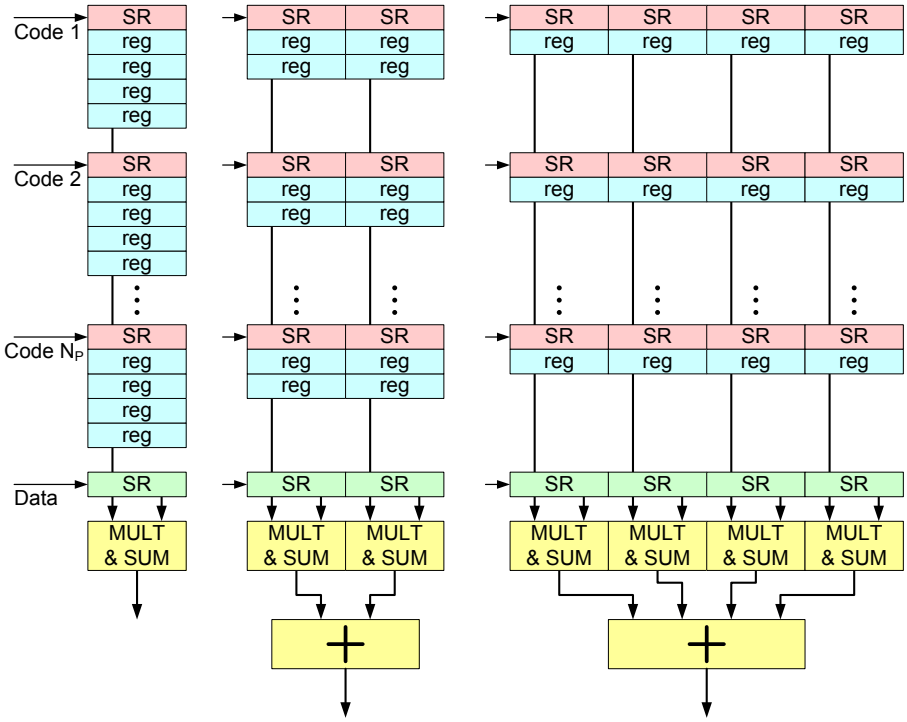


Fig. 27: *Different configurations of the programmable GC for 1x, 2x, and 4x frequency operations (adapted from [P4]).*

follow it in the signal processing path. For a single input data stream, we can make the configuration of the blocks in such a way that it would be possible to combine several blocks operating at different sample frequencies and still produce the outputs at the same rate [P4]. This is illustrated in Figure 27.

It is also possible to utilize multiple input shift registers and use them in the correlation either in a time-multiplexed way or by selecting an input shift register to use for each GC code channel. The first case requires as

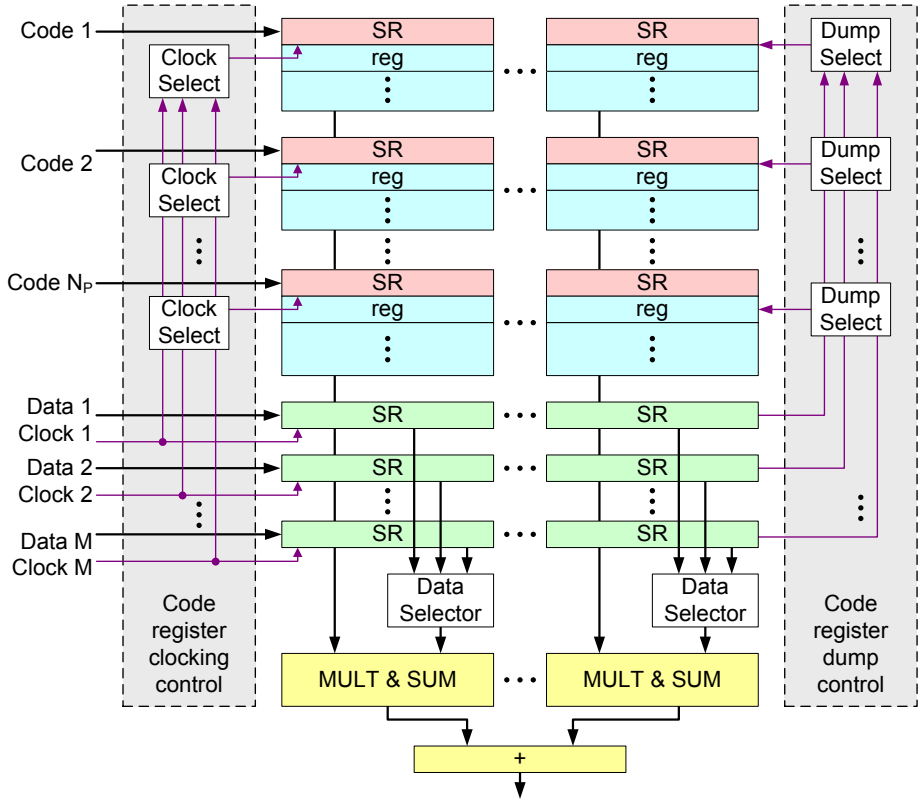


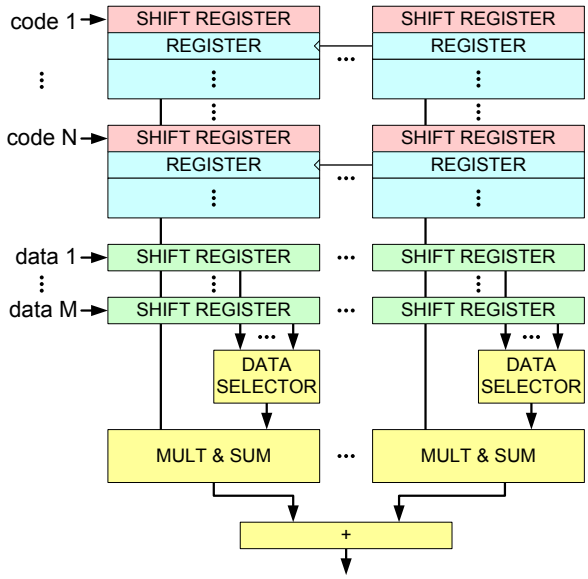
Fig. 28: *Block diagram of a programmable GC block with multiple input sample rates (from [P4]).*

many times higher clock frequency as there are inputs, but the second case can use the same clock frequency as the one with just single input shift register. If the different input channels run with different sample rates, which are not multiples of a common base frequency, then it is not so easy to correlate all input channels against all code channels. A GC with multiple inputs with their own sample rates is shown in Figure 28.

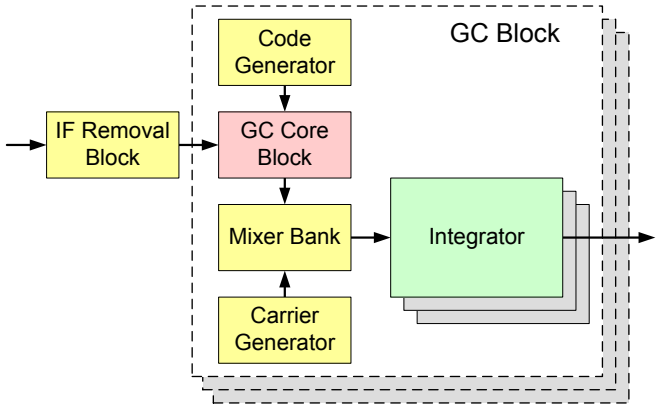
4.4.3 GC-Based Receiver for Gate Count Comparison[13]

The GC-based baseband architecture used in the gate count comparison is illustrated in Figure 29. This architecture is based on the one originally published by the author in [P4]. We have inserted a coarse carrier removal block before the GC core block to allow for a wider range of IF frequencies and Doppler frequencies than the GC bandwidth. We also have an integrator block after the GC core. By utilizing the coherent post-integration, the GC can do a full correlation over a partial range of code phases, which makes it more flexible than the MF, but controlling the unit will be more complex. The GC offers a higher bandwidth compared with the MF as the correlated code segments are shorter. Thus, it is possible to process signals from multiple satellites within one time-multiplexed block. The GC can also be made to operate on more than one input within a single block, which allows running both tracking and acquisition in parallel using only a single hardware block.

[13] Publication [P6] contains the details of the GC architecture selected for the gate count comparison.



(a) *GC core implementation (adapted from [P4]).*



(b) *Whole GNSS receiver architecture (adapted from [P6]).*

Fig. 29: *GC-based GNSS receiver architecture used for the gate count comparison.*

5. ARCHITECTURE COMPARISON[1]

In this chapter, we will show how the gate count estimation method introduced in Chapter 3 can be used on the three different GNSS receiver architectures described in Chapter 4. For the comparison, we have crafted three different test cases discussed in the next section. This is followed by the introduction of the comparison results. We will compare the architectures optimized for each test case separately. This will show how they fit for a particular use case. As each optimized receiver might not fulfill the requirements for all test cases with the same configuration, we also compare receivers that are configured to fulfil the requirements of all the test cases with a single configuration. This will show which architecture is the most flexible and best for overall use. The comparison focuses only on the gate count or silicon area of the receivers and does not take other parameters such as power consumption into account. This is a limitation which could be resolved in future research.

5.1 Test Cases

It would not be a fair comparison if all the receiver architectures would be optimized for different targets. Thus we need to start the comparison process by defining use cases for the receivers, and only after that we

[1]The contents of this chapter are based on the Publication [P6]

can perform the comparison. Also, it would not be useful to just define some arbitrary test cases since that could give wrong results for real life application. For this thesis, we have selected three different use cases and the combined test case and we will also explain how they represent real life or close to real life situations for a GNSS receiver. In order to avoid unnecessary complexity in the test case definition, we concentrate on GPS only[2], but the cases could quite easily be extended to cover other GNSS systems as well.

5.1.1 Acquisition

Signal acquisition is the most resource hungry step in GNSS receiver operation. One of the critical performance criteria for GNSS receivers is the Time-To-First-Fix (TTFF), i.e. the time it takes from starting the receiver to the moment it is able to provide the first calculated PVT output. Thus, rapid acquisition is also an important design goal for any GNSS receiver development. The acquisition problem has already been discussed in Subsection 2.4.1, so we will not go into the details here. Table 11 will summarize the important parameters for the acquisition test case.

5.1.2 Tracking

For continuous operation, the GNSS receiver needs to track the signals received from the satellites in order to constantly measure the ranges between the receiver and satellites and to decode the data message the satellites are transmitting. In the tracking phase, the target is not the speed of operation but the accuracy, which usually means that the correlator bins

[2] as we did in [P6]

Table 11: *Acquisition test case parameters (from [P6])*

Parameter	**Value**	
Time to acquire ≥ 4 satellites	10	s
Acquisition sensitivity	31	dB-Hz
Coherent integration time	3	ms
Dwell-time	1800	ms
Number of PRN codes	12	
Frequency bin size	167	Hz
Frequency search range	± 8.2	kHz
Number of frequency bins	17	
Code phase search range	1023	chips
Code phase resolution	1/2	chips
Number of code bins	2046	
Total number of search bins	417384	
Parallel search bins	83000	

Table 12: *Tracking test case parameters (from [P6])*

Parameter	**Value**	
Number of PRN codes	16	
Number of frequency bins	1	
Code phase resolution	1/8	chips
Number of code bins	5	
Total number of bins	80	

are more closely separated in the code delay-domain. In many cases we also want to receive simultaneously more signals than we need during the acquisition phase. Table 12 will summarize the important parameters for the tracking test case.

5.1.3 Assisted GPS

Assisted GNSS (A-GNSS) is usually interpreted to mean that we have some information from the satellites available before turning on the GNSS receiver [102, 103]. The information is delivered via other channels than through the signal transmitted by the satellites. As the out-of-band information does not significantly affect the tracking phase, the only difference is really in the acquisition phase. For the purpose of this comparison, we derive the Assisted GPS (A-GPS) test case parameters from the coarse-time assistance sensitivity test of the Third Generation Partnership Project (3GPP) test specification [104]. The parameters of this test case are summarized in Table 13. Note that the frequency uncertainty range for the GPS receiver is larger than what is specified in the 3GPP test case because the true frequency uncertainty for the receiver will be worse than the system uncertainty due to the inaccuracies of any frequency synchronization over the mobile phone network. In this test case, we have one strong signal and the rest are weaker.

5.1.4 Worst Case

The fourth and last test case does not define a separate receiver use case, but a combined case, for which the receivers must meet all the requirements of the preceding test cases with a single configuration. This is the closest to a real-life situation since the receiver hardware cannot usually be configured to a completely different architecture during operation. Whereas the previous cases show which architecture is the best for each test case with separately tuned configurations, this last case will show which of them is the most versatile one.

Table 13: *A-GPS test case parameters (from [P6])*

Parameter	**Value**	
Time to acquire ≥ 4 satellites	5	s
Acquisition sensitivity (weak)	27	dB-Hz
Acquisition sensitivity (strong)	32	dB-Hz
Coherent integration time	9	ms
Dwell-time	1400	ms
Number of PRN codes	9	
Frequency bin size	55	Hz
Frequency search range	$\pm$100	Hz
Number of frequency bins	4	
Code phase search range	1023	chips
Code phase resolution	1/2	chips
Number of code bins	2046	
Total number of search bins	73656	
Parallel search bins	32736	

5.2 Comparison Results

We have created a gate count estimation model for each of the GPS receiver BB architectures which were introduced in Chapter 4, and then optimized their parameters for each of the test cases separately subject to a maximum clock frequency of 128 MHz. Each receiver was split to a number of blocks (**Blks**). Each block contains a code generator capable of generating a number of different PRN sequences in parallel, these are then processed in parallel and this we call the number of channels (**Ch**). Similarly, in each block we have a carrier frequency generator, which can generate a number of parallel carrier replica signals (**Freq**). Finally, each receiver channel is capable of correlating the incoming signal with a number of differently delayed versions of the spreading code replica (**Code**).

Table 14: *Area estimates of the receivers in kilo-gates (corrected, from [P6])*

	Parameters			Area			
Arch	**Blks**	**Ch**	**Freq**	**Code**	**Total**	**/ch**	**/bin**
Acquisition							
Corr	21	8	8	64	9982	59	0.116
MF	1	12	4	2046	1205	100	0.012
GC	5	4	9	512	1925	96	0.021
Tracking							
Corr	1	16	1	5	66	4	0.826
MF	16	1	1	8184	5964	373	0.046
GC	1	16	1	128	125	8	0.061
A-GPS							
Corr	6	16	4	64	2945	31	0.120
MF	3	1	4	2046	703	234	0.029
GC	3	4	4	512	578	48	0.024
Worst-case							
Corr	21	8	8	64	9982	59	0.116
MF	16	1	4	8184	10698	669	0.020
GC	7	4	9	512	2692	96	0.021

The resulting receiver parameters and the respective gate count estimations are tabulated in Table 14, which shows both the used parameters as well as the gate count numbers. The gate count values are reported for the whole receiver (**Total**) as well as per channel (**/ch**) and per bin (**/bin**). The bins mean individual correlation results for each different PRN code, frequency, and code delay. The receiver configurations do not have the same number of blocks, channels, frequencies, and code delays, since some of the parameters cannot be selected freely.

The correlator-based receiver implemented the integrators using SRAM due to the high multiplexing ratio, which resulted in a large number of channels

sharing the computation logic. Even if the resulting memories were small in size, ranging from 0.2 to 1.5 kilobits, the size was smaller when the storage was implemented as memory. Despite the fact that the model of the gate count of the SRAM for small memories is not as accurate as for larger memories, the error is not large enough to change the interpretation of the results. The memories in the MF and GC-based receivers were considerably larger, ranging from 10 to 120 kilobits. For those architectures, the SRAM model is as accurate as the logic model. For the acquisition, A-GPS, and GC cases, the ratio between memories and logic is quite high, ranging from 55% to 90%. This is due to the large number of outputs that must be computed by the hardware, and relatively simple processing. For the tracking use case, the ratio was smaller, from 25% and 32%. This is due to the smaller number of outputs, and relatively larger number of separate receiver channels.

The main advantage of the correlator-based receiver is the flexibility. The receiver software can easily configure the receiver for different signal conditions. However, we can also see that the flexibility comes with the price of the largest area per bin. This can be seen from the fact that the worst case configuration is equal to the size of the largest individual case, whereas for the other architectures the worst case configuration is larger than any of the individual cases. The MF is in the other end of the spectrum. The flexibility is limited by the fact that the length of the shift register needs to be a multiple of the code length. This is not a problem in the acquisition case, which requires searching for all the code shifts, but causes quite a waste in the tracking case, where only a few code phases are needed for each tracked satellite, and where the delay between the different code bins needs to be a fraction of the chip period. The GC architecture is in the middle ground of these two extremes. It performs the best for the A-GPS

case, but it is still reasonably good for the acquisition and tracking cases. In the acquisition case, there is extra overhead due to the higher number of common hardware needed, and in the tracking case, there are still quite a few unnecessary correlation results produced.

6. CONCLUSIONS AND FUTURE WORK

In this thesis, the implementation and optimization of GNSS receiver baseband architectures were discussed. The GNSS receivers are becoming ubiquitous and as they are included in battery-powered, cheap consumer products, it becomes more important to ensure that the cost and power consumption of these devices are well optimized. We have shown how the optimization can be made and how the results can be measured so that the correct architecture will be chosen in each case.

We introduced a new method for high level gate count estimation that can be used already in the early phase of the development of digital integrated circuits. The method is easy to use and it requires very limited or no information about the silicon technology to be used for the implementation of the design. In publication [P5] we have shown that the new method offers very good accuracy, which is typically better than 5%. The best application of this method is early, during the architectural design phase, when different architectures are being compared. The proposed gate count comparison method provides the theoretical basis for the architecture comparison in this thesis.

We have reviewed three different time-domain, real-time architectures for performing the cross-correlation function which is the core algorithm used for signal tracking and acquisition in all GNSS receivers. For each of the

architectures, we have provided different and novel ways to optimize the implementations. This represents work over more than ten years and many of the ideas illustrated in this thesis were filed and granted as patents invented by the author, which is a testament of their novelty. These architechtures and optimizations have also been used in real GNSS receiver designs developed by the author while working in the industry.

The contribution of this thesis culminates in the comparison of the architectures presented here. We have developed test cases for the comparison based on realistic use cases for an A-GPS receiver. The test cases represent an example, as they depend on the intended use of the receivers to be compared. What we have shown, is that the winner of the comparison depends on the use case. One size does not fit all. It was seen that flexibility increases the gate count per correlator. The combined test case, where the receiver needs to fulfill all the usage scenarios with a single configuration shows that the GC represents a good compromise between flexibility and area efficiency.

In order to keep the scope of the thesis manageable, it had to be limited to real-time, digital hardware-based, time-domain architectures. It would be interesting to expand the comparison to also cover the software-based, store and process, and transform-domain implementations. This will be a nice research subject for the future. Another interesting research topic would be to find out an optimal way to construct a receiver utilizing two or more blocks based on a mixed set of architectures, as it was quite obvious based on the comparison results that different architectures are better suited for certain operations than the others.

Also, the comparison was done only for optimizing the gate count. Gate count, or silicon area, is indirectly related to the power consumption, but

no attempt was made to find out which architecture would be the most power efficient. Power estimation is a separate research topic in itself, but it would be interesting area for future research. Developing an accurate and easy-to-use method for early design phase power estimation, would be a valuable optimization tool for the design of portable, battery powered devices. It would also be interesting to apply such tool to the field of GNSS receivers.

BIBLIOGRAPHY

[1] *FCC Wireless 911 Requirements Fact Sheet WTB Policy*, FCC, Jan. 2001.

[2] H. Heusala and J. Skyttä, "How small and still effective a CMOS-SoC could ever be?" in *2015 Nordic Circuits and Systems Conference (NORCAS): NORCHIP & International Symposium on System-on-Chip (SoC)*. IEEE, oct 2015.

[3] Synopsys design compiler ultra. [Online]. Available: https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html

[4] A. A. Abidi, "The path to the software-defined radio receiver," *IEEE Journal of Solid-State Circuits*, vol. 42, no. 5, pp. 954–966, May 2007.

[5] D. M. Akos, "A software radio approach to global navigation satellite system receiver design," Ph.D. dissertation, Ohio University, 1997.

[6] M. Moeglein and N. Krasner, "An introduction to SnapTrack® wireless-assisted GPS technology^{TM}," *GPS Solutions*, vol. 4, no. 3, pp. 16–26, Jan. 2001.

[7] P. C. Ould and R. J. Van Wechel, "All-digital GPS receiver mechanization," *Navigation*, vol. 28, no. 3, pp. 178–188, Sep. 1981.

[8] D. van Nee and A. Coenen, "New fast GPS code-acquisition technique using FFT," *Electronics Letters*, vol. 27, no. 2, pp. 158–160, 1991.

[9] J. York, J. Little, D. Munton, and K. Barrientos, "A fast number-theoretic transform approach to a GPS receiver," *Navigation*, vol. 57, no. 4, pp. 297–307, 2010.

[10] R. Maher, "A comparison of multichannel, sequential and multiplex GPS receivers for air navigation," *Navigation*, vol. 31, no. 2, pp. 96–111, 1984.

[11] V. Eerola and T. Ritoniemi, "Spread spectrum receiver," U.S. Patent 6,850,558, Feb. 1, 2005, filed Oct. 13, 2000.

[12] ——, "Spread spectrum receiver," U.S. Patent 7,283,582, Oct. 13, 2000, filed Oct. 13, 2000.

[13] ——, "Spread spectrum receiver," U.S. Patent 7,471,719, Sep. 18, 2007, filed Sep. 18, 2007, Continuation of US7,283,582.

[14] ——, "Signal acquisition system for spread spectrum receiver," U.S. Patent 6,909,739, Jun. 21, 2005, filed Oct. 13, 2000.

[15] ——, "Matched filter and spread spectrum receiver," U.S. Patent 7,010,024, Mar. 7, 2006, filed Oct. 13, 2000.

[16] ——, "Matched filter and spread spectrum receiver," U.S. Patent 7,505,511, Mar. 17, 2009, filed Dec. 23, 2005, Continuation of US6,850,558.

[17] V. Eerola, "Performing a correlation in reception of a spread spectrum signal," U.S. Patent 7,852,907, Dec. 14, 2010, filed Dec. 21, 2006.

[18] H. Valio and S. Pietilä, "Reception of a spread spectrum modulated signal," U.S. Patent 7,839,915, Nov. 23, 2010, filed Apr. 21, 2005.

[19] V. Eerola, "Apparatus and method for correlation in a gps receiver," U.S. Patent 8,391,335, Mar. 5, 2013, filed Oct. 13, 2006.

[20] E. D. Kaplan, Ed., *Understanding GPS: Principles and Applications*. Norwood, MA, USA: Artech House Publishers, 1996.

[21] B. W. Parkinson, J. J. Spilker, Jr., and P. Enge, Eds., *Global positioning system: theory and applications*. Washington DC: American institute of aeronautics and astronautics, Inc., 1996, vol. 109.

[22] GPS: Space segment. [Online]. Available: https://www.gps.gov/systems/gps/space/

[23] GLONASS constellation status. [Online]. Available: https://www.glonass-iac.ru/en/GLONASS/

[24] GLONASS history. [Online]. Available: https://www.glonass-iac.ru/en/guide/index.php

[25] Galileo Programme: Operational satellites. [Online]. Available: https://www.gsa.europa.eu/galileo/programme

[26] BeiDou constellation status. [Online]. Available: http://www.csno-tarc.cn/system/basicinfo

[27] J. Shen, "Development of beidou navigation satellite system (bds)," in *57th Meeting of the Civil GPS Service Interface Committee*, Sep. 2017. [Online]. Available: https://www.gps.gov/cgsic/meetings/2017/shen.pdf

[28] *Navstar GPS Space Segment/Navigation User Interfaces. Interface Specification IS-GPS-200H*, Dec. 2015. [Online]. Available: https://www.gps.gov/technical/icwg/IRN-IS-200H-001+002+003_rollup.pdf

[29] *GLONASS ICD for frequency separated radio signal: Interface control document, Edition 5.1*, Moscow, Russia, 2008. [Online]. Available: http://russianspacesystems.ru/wp-content/uploads/2016/08/ICD_GLONASS_eng_v5.1.pdf

[30] *Galileo - Open Service - Signal In Space Interface Control Document (OS SIS ICD V1.3)*, Dec. 2016. [Online]. Available: https://www.gsc-europa.eu/system/files/galileo_documents/Galileo-OS-SIS-ICD.pdf

[31] *BeiDou Navigation Satellite System - Signal In Space - Interface Control Document - Open Service Signals (B1I and B2I) (Version 1.2)*, China Satellite Navigation Office, Nov. 2016. [Online]. Available: http://www.beidou.gov.cn/xt/gfxz/201712/P020171226741342013031.pdf

[32] *Navstar GPS Space Segment/User Segment L1C Interface. Interface Specification IS-GPS-800D*, Sep. 2013. [Online]. Available: https://www.gps.gov/technical/icwg/IS-GPS-800D.pdf

[33] *GLONASS ICD for code separated radio signal: Navigation radio signal in ranges L1 (edition 1)*, Moscow, Russia, 2016. [Online]. Available: http://russianspacesystems.ru/wp-content/uploads/2016/08/ICD-GLONASS-CDMA-L1.-Edition-1.0-2016.pdf

[34] *BeiDou Navigation Satellite System - Signal In Space - Interface Control Document - Open Service Signal B1C (Version 1.0)*, China

Satellite Navigation Office, Dec. 2017. [Online]. Available: http://www.beidou.gov.cn/xt/gfxz/201712/P020171226741342013031.pdf

[35] P. J. G. Teunissen and O. Montenbruck, Eds., *Springer Handbook of Global Navigation Satellite Systems*. Cham, Switzerland: Springer International Publishing, 2017.

[36] R. Pickholtz, D. Schilling, and L. Milstein, "Theory of spread-spectrum communications — A tutorial," *IEEE Transactions on Communications*, vol. 30, no. 5, pp. 855–884, May 1982.

[37] R. Scholtz, "The origins of spread-spectrum communications," *IEEE Transactions on Communications*, vol. 30, no. 5, pp. 822–854, May 1982.

[38] R. Price, "Further notes and anecdotes on spread-spectrum origins," *IEEE Transactions on Communications*, vol. 31, no. 1, pp. 85–97, Jan 1983.

[39] R. Scholtz, "Notes on spread-spectrum history," *IEEE Transactions on Communications*, vol. 31, no. 1, pp. 82–84, Jan. 1983.

[40] M. K. Simon, J. K. Omura, R. A. Scholtz, and B. K. Levitt, *Spread Spectrum Communications Handbook*, rev.ed. ed. New York, NY, USA: McGraw-Hill, 1994.

[41] J. K. Holmes, *Coherent spread spectrum systems*. Malabar, FL: Robert E. Krieger Publishing Company, 1990.

[42] R. L. Peterson, R. E. Ziemer, and D. E. Borth, *Introduction to Spread-spectrum Communications*. London: Prentice Hall, 1995.

[43] A. Viterbi, "Spread spectrum communications — myths and realities," *IEEE Communications Magazine*, vol. 17, no. 3, pp. 11–18, may 1979.

[44] R. C. Dixon, *Spread spectrum systems*. New York: J. Wiley, 1984.

[45] A. J. Viterbi, *CDMA: Principles of Spread Spectrum Communication*. Reading, MA: Addison-Wesley, 1995, vol. 122.

[46] R. J. Milliken and C. J. Zoller, "Principle of operation of NAVSTAR and system characteristics," *Navigation*, vol. 25, no. 2, pp. 95–106, Jun. 1978.

[47] R. B. Langley, "GPS receiver system noise," *GPS world*, vol. 8, no. 6, pp. 40–45, 1997.

[48] H. Chang, "Presampling filtering, sampling and quantization effects on the digital matched filter performance," in *International Telemetering Conference Proceedings*. International Foundation for Telemetering, Sep. 1982.

[49] T. Lim, "Non-coherent digital matched filters: Multibit quantization," *IEEE Transactions on Communications*, vol. 26, no. 4, pp. 409–419, Apr. 1978.

[50] M. Braasch and A. van Dierendonck, "GPS receiver architectures and measurements," *Proceedings of the IEEE*, vol. 87, no. 1, pp. 48–64, 1999.

[51] (2014, Jun.) Magellan celebrates 25th anniversary of first commercial handheld GPS, the NAV 1000. Press Release. [Online]. Available: http://www.magellangps.com/Newsroom/2014-Archives_3/Press-Release-June-2-2014

[52] (2017, Mar.) Retro GPS: Magellan NAV 1000. [Online]. Available: http://retro-gps.info/Magellan/Magellan-NAV-1000/index.html

[53] R. A. Maher, "A comparison of multichannel, sequential and multiplex GPS receivers for air navigation," *Navigation*, vol. 31, no. 2, pp. 96–111, Jun. 1984.

[54] J. Ashjaee, D. Bourn, R. Helkey, R. Lorenz, J. Minazio, B. Remondi, and R. Sutherland, "Ashtech XII GPS receiver-the all-in-one all-in-view," in *IEEE PLANS '88., Position Location and Navigation Symposium, Record. 'Navigation into the 21st Century'.* IEEE, 1988.

[55] *Javad OEM GNSS receiver TR-3N Datasheet*, Javad GNSS Inc., Jun. 2018. [Online]. Available: http://www.javad.com/downloads/javadgnss/sheets/TR-3N_Datasheet.pdf

[56] P. Ward, "GPS receiver search techniques," in *Proceedings of Position, Location and Navigation Symposium - PLANS '96.* IEEE, 1996.

[57] M. Lehtinen, A. Happonen, and J. Ikonen, "Accuracy and time to first fix using consumer-grade GPS receivers," in *2008 16th International Conference on Software, Telecommunications and Computer Networks.* IEEE, 2008.

[58] B. Sklar, *Digital communications — fundamentals and applications.* Upper Saddle River: Prentice-Hall PTR, 2001.

[59] J. Spilker and D. Magill, "The delay-lock discriminator-an optimum tracking device," *Proceedings of the IRE*, vol. 49, no. 9, pp. 1403–1416, Sep. 1961.

[60] J. Costas, "Synchronous communications," *Proceedings of the IRE*, vol. 44, no. 12, pp. 1713–1718, Dec. 1956.

[61] S. Riter, "An optimum phase reference detector for fully modulated phase-shift keyed signals," *IEEE Transactions on Aerospace and Electronic Systems*, vol. AES-5, no. 4, pp. 627–631, Jul. 1969.

[62] F. Natali, "AFC tracking algorithms," *IEEE Transactions on Communications*, vol. 32, no. 8, pp. 935–947, Aug. 1984.

[63] C. Cahn, D. Leimer, C. Marsh, F. Huntowski, and G. LaRue, "Software implementation of a PN spread spectrum receiver to accommodate dynamics," *IEEE Transactions on Communications*, vol. 25, no. 8, pp. 832–840, aug 1977.

[64] P. Misra and P. Enge, *Global Positioning System: Signals, Measurements and Performance*. Ganga-Jamuna Press, 2006.

[65] A. J. V. Dierendonck, "GPS receivers," in *Global positioning system: theory and applications*, B. W. Parkinson, J. J. Spilker, Jr., and P. Enge, Eds. American institute of aeronautics and astronautics, Inc., 1996, vol. I, ch. 8, pp. 329–408.

[66] E. D. Kaplan, J. L. Leva, and M. S. Pavloff, "Fundamentals of satellite navigation," in *Understanding GPS: Principles and Applications*, E. D. Kaplan, Ed. Norwood, MA, USA: Artech House Publishers, 1996, ch. 2.

[67] A. Hauschild, "Basic observation equations," in *Springer Handbook of Global Navigation Satellite Systems*, P. J. G. Teunissen and O. Montenbruck, Eds. Cham, Switzerland: Springer International Publishing, 2017, ch. 19.

[68] D. Odijk, "Positioning model," in *Springer Handbook of Global Navigation Satellite Systems*, P. J. G. Teunissen and O. Montenbruck,

Eds. Cham, Switzerland: Springer International Publishing, 2017, ch. 21.

[69] S. Verhagen and P. J. G. Teunissen, "Least-squares estimation and kalman filtering," in *Springer Handbook of Global Navigation Satellite Systems*, P. J. G. Teunissen and O. Montenbruck, Eds. Cham, Switzerland: Springer International Publishing, 2017, ch. 22.

[70] C. Shannon, "The synthesis of two–terminal switching circuits," *Bell System Technical Journal*, vol. 28, pp. 59–98, 1949.

[71] D. E. Muller, "Complexity in electronic switching circuits," *Electronic Computers, IRE Transactions on*, vol. EC-5, no. 1, pp. 15–19, Mar. 1956.

[72] E. Kellerman, "A formula for logical network cost," *Computers, IEEE Transactions on*, vol. C-17, no. 9, pp. 881–884, Sep. 1968.

[73] N. Pippenger, "Information theory and the complexity of boolean functions," *Theory of Computing Systems*, vol. 10, pp. 129–167, 1976.

[74] R. W. Cook and M. J. Flynn, "Logical network cost and entropy," *Computers, IEEE Transactions on*, vol. C-22, no. 9, pp. 823–826, Sep. 1973.

[75] K.-T. Cheng and V. Agrawal, "An entropy measure for the complexity of multi-output boolean functions," in *Proceedings of the 27th ACM/IEEE Design Automation Conference (DAC)*, Jun. 1990, pp. 302–305.

[76] M. Nemani and F. Najm, "High-level area and power estimation for vlsi circuits," *Computer-Aided Design of Integrated Circuits and*

Systems, IEEE Transactions on, vol. 18, no. 6, pp. 697–713, Jun. 1999.

[77] K. M. Büyükşahin and F. N. Najm, "High-level area estimation," in *Proceedings of the 2002 international symposium on Low power electronics and design*, ser. ISLPED '02. New York, NY, USA: ACM, 2002, pp. 271–274.

[78] S. Akers, "Binary decision diagrams," *Computers, IEEE Transactions on*, vol. C-27, no. 6, pp. 509–516, Jun. 1978.

[79] R. Bryant, "On the complexity of VLSI implementations and graph representations of boolean functions with application to integer multiplication," *Computers, IEEE Transactions on*, vol. 40, no. 2, pp. 205–213, Feb. 1991.

[80] R. Enzler, T. Jeger, D. Cottet, and G. Tröster, "High-level area and performance estimation of hardware building blocks on FPGAs," in *Proceedings of the The Roadmap to Reconfigurable Computing, 10th International Workshop on Field-Programmable Logic and Applications*, ser. FPL '00. London, UK: Springer-Verlag, 2000, pp. 525–534.

[81] M. Abdelhalim and S.-D. Habib, "Fast FPGA-based area and latency estimation for a novel hardware/software partitioning scheme," in *Electrical and Computer Engineering, 2008. CCECE 2008. Canadian Conference on*, May 2008, pp. 775–780.

[82] R. Meeuws, "A quantitative model for hardware/software partitioning," Master's thesis, Delft University of Technology, Delft, Netherlands, May 2007.

[83] *VSDSP2 Processor Core Manual*, VLSI Solution Oy, Tampere, Finland, Mar. 2001, version 2.6. [Online]. Available: http://www.vlsi.fi/fileadmin/manuals_guides/vsdsp2_um.pdf

[84] H. Singleton, "A digital electronic correlator," *Proceedings of the IRE*, vol. 38, no. 12, pp. 1422–1428, dec 1950.

[85] M. M. Chansarkar and L. Garin, "Acquisition of GPS signals at very low signal to noise ratio," in *Proceedings of the 2000 National Technical Meeting of The Institute of Navigation (ION NTM 2000)*, Anaheim, CA, Jan. 2000, pp. 731–737.

[86] F. Van Diggelen and C. Abraham, "Indoor GPS technology," *CTIA Wireless-Agenda, Dallas*, vol. 89, 2001.

[87] F. Van Diggelen, *A-GPS — Assisted GPS, GNSS, and SBAS*. Norwood: Artech House, 2009.

[88] M. Laxton and S. DeVilbiss, "GPS multipath mitigation during code tracking," in *Proceedings of the 1997 American Control Conference (Cat. No.97CH36041)*. IEEE, 1997.

[89] F. M. Schubert, J. Wendel, M. Sollner, M. Kaindl, and R. Kohl, "The astrium correlator: Unambiguous tracking of high-rate BOC signals," in *2014 IEEE/ION Position, Location and Navigation Symposium - PLANS 2014*. IEEE, May 2014.

[90] O. Julien, C. Macabiau, M. Cannon, and G. Lachapelle, "ASPeCT: Unambiguous sine-BOC(n, n) acquisition/tracking technique for navigation applications," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 43, no. 1, pp. 150–162, Jan. 2007.

[91] J. Knight, G. Turetzky, C. Norman, and D. Hilgenberg, "SiRF's low power receiver advances," in *Proceedings of the 11th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GPS 1998)*, Nashville, TN, Sep. 1998, pp. 299–305.

[92] A. Genghi, J. Brenner, F. Pisoni, and F. Spalla, "Counting equivalent correlators," *GPS World*, vol. 20, no. 1, pp. 39–65, Jan. 2009.

[93] O. Julien, M. E. Cannon, G. Lachapelle, C. Mongrédien, and C. Macabiau, "A new unambiguous BOC(n,n) signal tracking technique," in *Proceedings of The European Navigation Conference GNSS*, 2004, pp. 17–19.

[94] V. A. Veitsel, A. V. Zhdanov, and M. I. Zhodzishsky, "The mitigation of multipath errors by strobe correlators in GPS/GLONASS receivers," *GPS Solutions*, vol. 2, no. 2, pp. 38–45, 1998.

[95] A. Van Dierendonck, P. Fenton, and T. Ford, "Theory and performance of narrow correlator spacing in a GPS receiver," *Navigation*, vol. 39, no. 3, pp. 265–283, 1992.

[96] G. Turin, "An introduction to digitial matched filters," *Proceedings of the IEEE*, vol. 64, no. 7, pp. 1092–1112, 1976.

[97] D. Cartier, "Partial correlation properties of pseudonoise (PN) codes in noncoherent synchronization/detection schemes," *IEEE Transactions on Communications*, vol. 24, no. 8, pp. 898–903, Aug. 1976.

[98] T. Kim, W. Jao, and S. Tjiang, "Arithmetic optimization using carry-save-adders," in *Proceedings 1998 Design and Automation Conference. 35th DAC. (Cat. No.98CH36175).* IEEE, 1998.

[99] Z.-J. Mou and F. Jutand, "'overturned-stairs' adder trees and multiplier design," *IEEE Transactions on Computers*, vol. 41, no. 8, pp. 940–948, 1992.

[100] V. Eerola, "Rapid parallel GPS signal acquisition," in *Proc. of the 13th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GPS 2000)*, Salt Lake City, UT, Sep. 2000, pp. 810–816.

[101] D. E. Knuth, *The Art of Computer Programming: Seminumerical Algorithms*, 3rd ed. Reading, Massachusetts: Addison Wesley, 1998, vol. 2, ch. 4.1.

[102] M. Monnerat, R. Couty, N. Vincent, O. Huez, and E. Chatre, "The assisted GNSS, technology and applications," in *Proceedings of the 17th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GNSS 2004)*, Long Beach, CA, Sep. 2004, pp. 2479–2488.

[103] G. Djuknic and R. Richton, "Geolocation and assisted GPS," *Computer*, vol. 34, no. 3, pp. 123–125, mar 2001.

[104] *3GPP2 C.S0036-0 v1.0, Recommended Minimum Performance Specification for C.S0022-0 Spread Spectrum Mobile Stations*, 2011. [Online]. Available: http://www.3gpp2.org/Public_html/specs/C.S0036-0_v1.0.pdf

PUBLICATION 1

Ville Eerola and Jari Nurmi, "Correlator Design and Implementation for GNSS Receivers," in *NORCHIP, 2013*, Nov 2013.
DOI: 10.1109/NORCHIP.2013.6702037

Correlator Design and Implementation for GNSS Receivers

(Invited Paper)

Ville Eerola, Jari Nurmi
Department of Electronics and Communications Engineering
Tampere University of Technology
P.O.BOX 553, FIN-33101 Tampere, Finland
Email: ville.eerola@tut.fi

***Abstract*—This paper shows how the correlator block for GNSS receivers can be analyzed and optimized. The received signal characteristics are reviewed and used in the optimization process. The correlator is a key element in the GNSS receivers and it takes a considerable amount of the chip area and power. It is shown that a gate-count reduction of 30% or more can be achieved while also reducing the power consumption by more than 50%. The authors also discuss how advanced functionality can be added with just a minor increase of the correlator gate-count. The additional logic allows real-time configuration of the correlator hardware, which increases its usability in advanced GNSS receivers.**

I. Introduction

The number of Global Navigation Satellite System (GNSS) receivers in use has exploded with the introduction of personal navigation devices (PND) and inclusion of GNSS receivers in mobile phones. These portable devices require power and cost efficient implementations of the GNSS receivers. The most widely known GNSS system is the American GPS [1], [2], but there are other systems already existing or coming such as the Russian GLONASS [3], and European Galileo [4], to name a few. Although specific to GPS, [5] contains a nice introduction to GNSS receiver architectures and measurements.

The GNSS signals employ direct-sequence spread-spectrum (DS-SS) transmission for their signals. A good introduction to spread-spectrum communications is for example in [6]. A notable characteristic of a spread-spectrum signal is that its bandwidth is significantly larger than required for the data itself. The DS-SS signal is generated by multiplying the modulated data-bearing signal by a spreading code, which is a pseudo-random sequence. The DS-SS receivers need to de-spread the incoming signal by multiplying it with a properly aligned local replica of the spreading code, and keep the code aligned accurately while operating.

The signal from each satellite needs a separate receiver channel. Fig. 1 illustrates the data flow and processing required for tracking and receiving a signal from one GNSS satellite. In addition, many modern GNSS receivers have dedicated hardware blocks for signal acquisition, which is an important part of the operation of a GNSS receiver. The multiplication of the received signal by the local carrier and spreading code followed by the filter is termed as "correlation" and the part doing this operation is called "correlator". With increasing performance requirements and with the introduction of new GNSS systems, the demand for the number of correlator channels and their complexity is increasing. Already the correlators take a major portion of GNSS receiver chip area. Thus, their efficient implementation is very important and it will be the main subject of this paper.

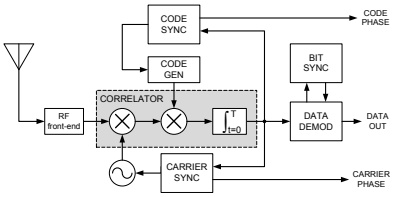


Fig. 1. Conceptual diagram of GNSS receiver channel structure.

In the next section, we will first go through some basic elements and the traditional correlator architectures. Next, we will show how to further optimize the implementation cost and power consumption of the correlator channel. Then we will discuss how to further improve the functionality and versatility of the correlator channel to improve the performance of the GNSS receiver operation. Finally, we will conclude our paper and summarize the merits of the correlator architectures presented here.

II. Traditional Correlator Design Considerations

The basic correlator is very simple in its operation and it might appear that there is little to optimize in its design. However, there are special implementation aspects that need to be taken into account when designing correlators for GNSS applications. In this section, we will first introduce the traditional GNSS correlator channel architecture, and then discuss the constraints affecting its design.

A. Corelator Functionality

Fig. 2 depicts a traditional architecture for a digital GNSS receiver correlator channel. First, the signal is multiplied by the local carrier replica, and then by several differently delayed

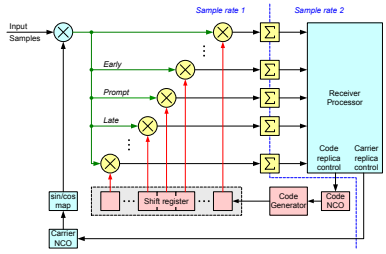


Fig. 2. Block diagram of traditional correlator channel architecture.

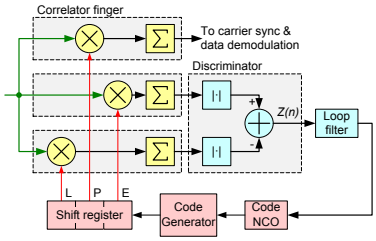


Fig. 3. Block diagram of correlator fingers for DLL based signal tracking.

versions of the local replica code. Finally, the results are accumulated for a period of time, and transferred to the receiver processor, which performs further computations for signal acquisition, tracking, and data recovery based on the accumulated data. Each correlation with a differently delayed version of the replica code is called a correlator finger. The data is usually processed as complex samples mixed to DC or very low IF. The conversion from real, bandpass to complex, lowpass data occurs either in the RF front-end or in the carrier replica mixer, but the correlator output data is almost invariably in complex, or I/Q-format.

The number of correlator fingers per channel in modern GNSS receiver varies, but it is always more than three. Two of the fingers (*early* and *late*) provide values for code tracking by a delay locked loop by correlating the incoming data with slightly advanced and delayed versions of the local replica code, and the third (*prompt*) provides values for decoding the satellite data message needed for position calculation by correlating the incoming signal with replica code that is exactly aligned with it. The tracking loops are closed in software running on the receiver processor. The particular discriminator functions ($Z(n)$ in Fig. 3) used in the software will determine how to adjust the code and carrier NCOs if it is obvious that the prompt output is out of phase. In the simplest case the discriminator is looking at the difference of the early and late outputs. This is the case shown in Fig. 3. Additional correlator fingers might be implemented to assist in signal acquisition

TABLE I
GNSS RECEIVER SNR VALUES EXAMPLES

	BW	**Weak**	**Nominal**	**Strong**
RX signal power		-155 dBm	-130 dBm	-120 dBm
RF filter	4 MHz	-50 dB	-25 dB	-15 dB
ADC output	16 MHz	-56 dB	-31 dB	-21 dB
Correlator output	1 kHz	-14 dB	11 dB	21 dB

TABLE II
GNSS RECEIVER NOISE AND SIGNAL VALUES EXAMPLES

	Noise		**Signal Amplitude**		
	σ	3σ	**Weak**	**Nominal**	**Strong**
Correlator input	1.86	5.58	0.003	0.052	0.163
Correlator output	238	714	48	845	2671

when the correct replica code phase is not yet known. In fact some correlator architectures have some reconfiguration capabilities to provide tens of fingers in acquisition mode. Another reason for increased number of correlator fingers is to provide indication of multipath propagation by correlating the incoming signal with a slightly more delayed version of the replica code than used for the DLL tracking. The multipath detection is required for example by the cellular Assisted GPS (AGPS) minimum performance standard [7] and it can also be used to improve the position computation accuracy in some cases. Furthermore, additional correlator fingers are required to support more complex modulations such as higher order Binary Offset Carrier (BOC) modulation in GALILEO receivers.

B. Wordlength considerations

One way to optimize the silicon area of the correlator is to minimize the data wordlength along the processing path. In order to do this, it is necessary to consider the characteristics of the received data stream as it comes into the correlator. The GNSS signals are in normally buried in the thermal noise, as the received signal strength is very weak. The noise power spectral density in a typical case is -171 dBm/Hz [8]. SNR values for three signal strength cases along the signal processing chain are shown in table Table I, with *BW* denoting bandwith. On most modern, commercial GNSS receivers the RF data is sampled using 2-bit signed-magnitude data format having the values of $\pm 1, \pm 3$. Following Widrow [9], it can be shown that the standard deviation (std) of the quantized thermal noise (σ_N) is 1.86 in the optimum case, which is normally maintained by the RF AGC circuitry. We can use this value and the SNR information to compute the noise and signal levels in the correlator input and output. The AGC keeps the noise at constant level, and σ_N is proportional to the square root of the number of accumulated samples in the integration. On the other hand, the signal level increases linearly with the number of accumulated samples. Noise and signal values for the three cases along the signal processing chain are shown in table Table II (represented as unitless numbers) with an

assumed ADC sampling rate of about 16 MHz. From the table we can see that 13-bit two's complement numbers would be enough to fully contain the strongest received signal, and that the largest representable value would be more than 17 times the std of the integrated noise. On the other hand, considering maximum possible value that could occur given the input values, we would need 17 bits to avoid overflow. This represents almost 25% decrease of the required bits in the integrator part of the correlator, which in most cases is the major portion of the correlator hardware.

C. Optimizing the Arithmetic

The preceding analysis makes a few simplifications. Firstly, it does not take into account the increase of wordlength in the arithmetic operations along the signal processing path. Secondly, it simply follows how the data values grow with the integration, which does not mean that the values could not be scaled down. Thirdly, if we used smaller than the maximum wordlength in the integrators, there would be a small possibility of overflow in case of large noise burst, interference, or exceptionally strong received signal, which could happen e.g. during spoofing attempts. The overflow must be avoided as it would be detrimental to the receiver operation.

We will first follow the maximal wordlength case, and then see where and how the wordlength could be minimized. Following Fig. 2, we first have to multiply the incoming samples by the carrier replica. The carrier replica is a sinusoidal represented usually with 3–5 bits. The complex multiplication will increase the wordlength by 3–5 bits. The multiplication by the local code replica is simply inversion of sign, so it does not affect the wordlength. Thus, in the worst case scaling method, we would need up to 22 bits for the correlator output. The mixers do not add information to the data stream, so we could scale the data after the carrier replica mixer back to three bits needed for the actual data. We should, however, not simply truncate the numbers after scaling, as that would introduce a bias to the data. The bias would adversely affect the receiver operation. The truncation of 2's complement numbers would add a bias of -0.5 to the data. This represents a very large DC signal, which would be 41 dB above the GNSS signal in the weak signal case acting as a very large jamming signal. The bias would also affect the operation of the acquisition and tracking algorithms which assume bias free signals. Using rounding, we can get rid of this bias, with some additional hardware cost. This cost is quite small in comparison with the saved cost on the integrators, however.

The overflow can be avoided by using saturating arithmetic. There are two different ways to handle the saturation. The first is to limit the output to the maximum representable value during each arithmetic operation without remembering the overflow. The second way is to monitor the output values, and raise a flag when an overflow is detected. The latter is called sticky overflow. Sticky overflow can be better for the receiver processing logic and can be used to detect abnormal situations, but it makes the implementation more complex. The non-sticky saturation could cause some of the signal to be lost, which would lead to erroneous signal amplitude estimations, but this is not as severe a problem as letting the output to overflow. Overflow would lead to errors which are almost twice as large as the largest value possible. In the implementation of the saturation for 2's complement numbers, it is possible to introduce bias. The saturation should also be implemented as bias free as it is very easy to implement in that way.

Taking into account the optimization of the arithmetic using scaling, rounding and saturation, it is possible to reduce the correlator output wordlength from 22 down to about 12 bits, which results in a significant saving in the hardware.

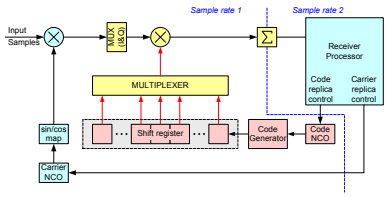


Fig. 4. Block diagram of time-multiplexed correlator channel architecture.

III. Time-Multiplexed Correlator Architecture

Time-multiplexing is a well-known method for reducing hardware area by scheduling computations to occur serially instead of performing them in parallel. Savings from time-multiplexing come from reducing the number of data processing elements. However, time-multiplexing cannot save any hardware which is used for storing the state, i.e. data storage registers.

Correlators in GNSS receivers offer very good possibilities for taking advantage of time-multiplexing since there are a large number of parallel computations required. Fig. 4 shows one possible implementation of a time-multiplexed correlator. This implementation saves area as we shall see shortly. There are other ways to approach the time-multiplexing of the correlator, which we will discuss next. Usually time-multiplexing requires increasing the clock rate to allow performing mode operations in the same amount of time, but it is possible to time-multiplex the correlators in a way which does not require increasing the clock rate since the output data rate is just a fraction of the input rate. Creating an optimized time-multiplexed correlator design requires some further analysis of the GNSS signals. As the signals are spread during transmission, the commercial GNSS signals require from 1 MHz (Glonass) to 4 MHz (Galileo) bandwidth before de-spreading. The carrier frequencies are in the L1 band (approximately 1.6 GHz) and their variation is very small, in the order of 10–30 kHz, depending on the quality of the frequency reference. This holds true for all systems utilising Code Domain Multiple Access (CDMA). Originally, Glonass system uses Frequency Domain Multiple Access (FDMA) and all satellites use the same spreading code. It is planned to add CDMA signals to

TABLE III
CORRELATOR GATE-COUNT EVALUATION RESULTS

	Traditional Correlator			**Time-Mult.**		**Time-Multiplexed & power opt.**			
Output bits	13	22		13		13			
First int N	n/a	n/a		n/a		16 / 32		64 / 128	
Fingers	gates	gates	diff	gates	diff	gates	diff	gates	diff
3	**2948**	4313	46.3%	2400	-18.6%	2933	-0.5%	3171	7.6%
5	**4364**	6639	52.1%	3267	-25.1%	4517	3.5%	4864	11.5%
8	**6488**	10128	56.1%	4569	-29.6%	6893	6.2%	7403	14.1%
16	**12152**	19432	59.9%	8038	-33.9%	13228	8.9%	14173	16.6%

TABLE IV
CORRELATOR ACTIVITY EVALUATION RESULTS IN KGATES·MHZ

	Trad.	**Time-Mult.**		**Time-Multiplexed & power optimized**							
First int N	n/a	n/a		16		32		64		128	
Fingers	act	act	diff	act	diff	act	diff	act	diff	act	diff
3	**22.7**	25.7	12.8%	12.1	-46.8%	9.8	-56.8%	10.0	-56.2%	9.3	-59.0%
5	**33.4**	43.1	29.1%	20.9	-37.5%	16.7	-49.9%	16.8	-49.8%	15.6	-53.2%
8	**49.4**	76.7	55.2%	35.2	-28.8%	27.6	-44.0%	27.3	-44.8%	25.2	-48.9%
16	**92.1**	208.8	126.8%	79.5	-13.6%	59.9	-34.9%	57.0	-38.1%	51.6	-43.9%

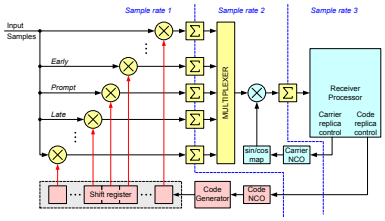


Fig. 5. Block diagram of power optimized time-multiplexed correlator channel architecture.

Glonass in future, which would make it compatible with time-multiplexed receiver structures.

In the correlator shown in Fig. 2, the carrier removal is done first, and the code removal next. This is to avoid implementing the larger carrier mixer for each correlator finger. However, the carrier mixer is operating at the input sample rate, and as a complex block also draws quite a bit of power. The code removal de-spreads the desired satellite signals to a much narrower bandwidth allowing the reduction of the sample rate for the received signals assuming that the IF frequency is low. At a lower sample rate, it is then possible to utilize time-multiplexing for the remainder of the data path. Then we could move the carrier removal mixer after the code multiplication and still implement it only once. This kind of arrangement is illustrated in Fig. 5, which shows a correlator structure which is functionally equal to that in Fig. 2. The integration is divided into two stages, because the typical output rate of the correlator, 1 kHz, is too low to represent the carrier variation without aliasing. It is possible to also time-multiplex the carrier removal mixer so that it would only need one multiplier and adder instead of four multipliers and two adders. We can use a similar data analysis process as for the traditional correlator optimization to determine the correct wordlength of the first integrators. Further optimization of the size of the remaining blocks in the data path is possible by rounding the first stage integration results, but this should be done only after a thorough analysis of the effects of dropping the lower bits.

For showing the achievable results, we have computed gate-count estimations for some design examples based on the traditional and time-multiplexed correlators. The gate-count numbers are only computed for the correlator core excluding the carrier and code replica generation. We have assumed a typical 2-bit sign-magnitude input data sampled at 16.368 MHz rate, which is sufficient for high quality tracking of GPS and Galileo signals. The results are shown in Table III. The gate-counts are compared with the optimally scaled traditional correlator, and the differences are shown relative to it. It is obvious that the word length optimization yields the biggest savings. The direct time-multiplexed architecture area is smallest, but it comes with a price of increased power consumption. The area of the power optimized time-multiplexed architecture is slightly larger than the optimized traditional correlator, but noticeably smaller than the non-optimized one. Overall, we can see that the architecture is more complicated, and results in similar area as for the traditional architecture. Next we will take a look on the power consumption of the different versions, and see how they compare to each another.

Commonly it has been thought that time-multiplexing does not save power but it saves area. This is usually true, if it is done only to re-schedule operations to be performed in series instead of performing them in parallel. Then the operations need to be performed faster, and the result is that the

switching activity level can increase. However, in the case of GNSS correlators, we can save considerable amount of power by performing time-multiplexing of the operations. Table IV shows the results of a simplistic evaluation of the switching activities in the correlators. We have included only the optimal wordlength cases in the table. The activity is calculated by assuming uniform 50% switching probability for all data bits. This is somewhat pessimistic, but as the data is more or less random by nature, it is not too far off. The activity of the registers is 100%, as they consume energy every time the clock is toggled. The clock rates are the same as discussed above. The activity numbers are given in kGates·MHz. Looking the results we see that there are great possibilities for huge power savings, but also that the results for the power optimized time-multiplexed case depend on the first integrator integration count. The activity can be seen to decrease with increasing first integration length, but that will cost us silicon area. So it is possible to select a "sweet-spot" in the parameter space to optimize the power consumption and to achieve power savings in the order of 40-60%.

IV. Advanced Correlator Functionality

As we have seen, the correlator fingers are relatively expensive to implement in hardware. However, many advanced receiver algorithms require more than simple early and late correlations. For example, in one BOC signal tracking algorithm [10] two versions of the early and late correlators are required in addition to the prompt correlator used for data demodulation. Another interesting scheme is the so called *strobe correlator* [11], which can improve tracking accuracy in presence of multipath. In a strobe-correlator, the chips of the normal replica PRN-codes are replaced with a stream of formatted pulses, called strobes. Fig. 6 shows some example strobes that can be used, and Fig. 7 shows their corresponding discriminator characteristics. In many cases, we could implement these advanced correlator functions by simply adding or subtracting the correlator outputs. For example, the simple strobe correlator can be implemented by subtracting the late correlator output from the early correlator output, given that the local replica code is suitably delayed in the shift register. Similarly, the other strobe correlators can be implemented by a combining a few correlators using differently delayed versions of the replica code. The generation of the strobes in the example cases are described in Table V. The notation $r[C \pm n]$ denotes using the replica code delayed (+) or advanced (–) by n times the code delay amount relative to the prompt correlator delay. We can also note that the simple strobe correlator is functionally equivalent to a "narrow correlator" [12], which uses a small (a fraction of a chip) delay between the early and late replicas in contrast to the normal 1 chip spacing.

The correlator is a linear system in the sense that adding the outputs of two correlators with different replicas is equivalent to adding the replicas before the multiplications. Thus, it would be possible to form more complex versions of the replica code and then use only one correlator finger to process it. This is illustrated in Fig. 8, which shows the generation of

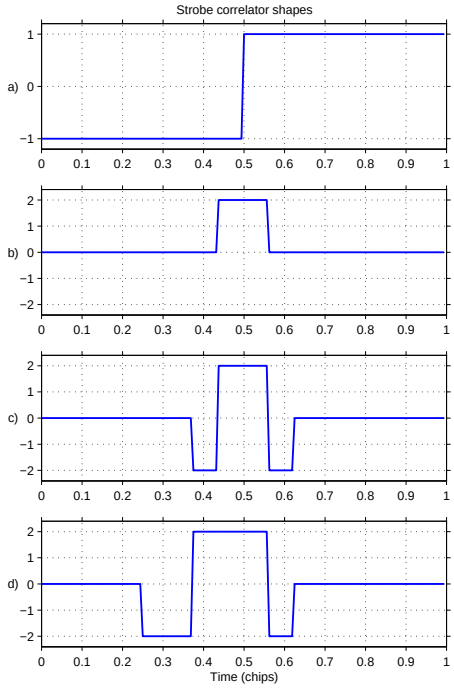


Fig. 6. Strobe correlator strobe example. a) Original PRN code, b) Simple strobe, c) Bipolar symmetrical strobe, d) Asymmetrical strobe.

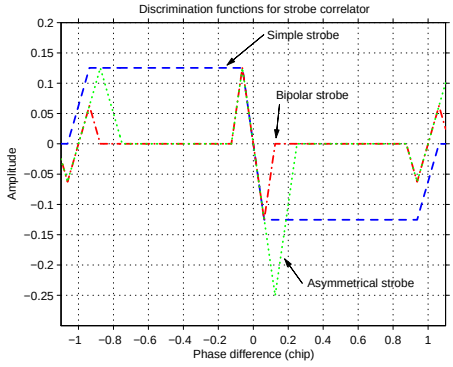
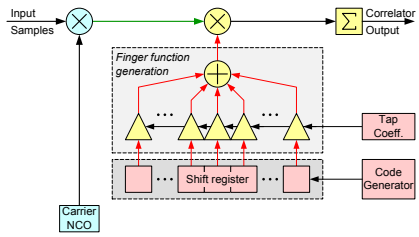
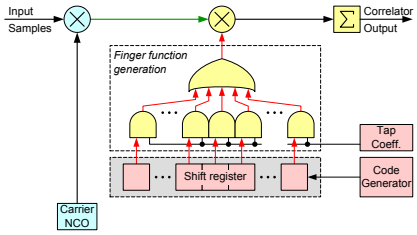


Fig. 7. Examples of strobe correlator discriminator characteritics.



(a) Full implementation



(b) Simplified implementation

Fig. 8. Correlator finger function example implementations.

TABLE V
GENERATION OF THE EXAMPLE STROBE SHAPES IN FIG. 6.

Strobe	**Generation**
Simple	$r[C-1] - r[C+1]$
Bipolar symmetrical	$r[C-2] - 2r[C-1] + 2r[C+1] - r[C+2]$
Asymmetrical	$r[C-2] - 2r[C-1] + 2r[C+2] - r[C+4]$

TABLE VI
FINGER FUNCTION IMPLEMENTATION EXAMPLES

Case	**Fingers**	**Gain sel.**	**Adder**	**gates**	**diff**
Simple Correlator	5 (0)	*n/a*	*n/a*	4364	-
Delay selection	5 (5)	0/1	OR	4674	7.1%
Simple functions	3 (2)	0, ± 1	2-bit	4074	-6.6%
Strobe functions	3 (2)	0, ± 1, ± 2	3-bit	4144	-5.0%

the replica for one correlator finger using a linear combination of differently delayed replica code versions—*a finger function*. Fig. 8a illustrates a complete case with adjustable weight for each replica code delay whereas in the case of Fig. 8b we can just enable or disable the different delays. Finger functions can also make the correlator very flexible by allowing its functionality to be configured for multiple situations via the receiver processing software. By using finger functions we can save valuable hardware in implementing the advanced algorithms. This is shown in Table VI, which shows evaluation of some example cases. The total number of fingers is shown in the table with the finger function enabled fingers shown in parentheses. The first two cases are normal correlators, and the two latter use finger functions, which enable creation of the discriminator directly in the correlator. This way fewer correlator fingers are needed for the same functionality. The table shows that we can e.g. save 5% in area when implementing the strobe correlators with finger functions.

V. CONCLUSION

The correlator is a key functional block in any GNSS receiver. In modern receivers, the complexity and amount of correlators has increased due to new and more complex signals and higher performance requirements. Even though the correlator is seemingly a simple block, its optimization requires thorough knowledge of the application and its constraints. We have shown that a gate-count reduction of 30% and power consumption reduction of more than 50% can be achieved by taking into account the data characteristics and optimising the wordlength and scheduling of the block. We have also illustrated how the correlator functionality can be made much more flexible with just a few simple additional components. This allows the GNSS receiver to operate efficiently in many different situations.

ACKNOWLEDGMENT

This work has been supported by a grant from Alfred Kordelin Foundation, Helsinki.

REFERENCES

[1] *Navstar GPS Space Segment/Navigation Users Interface. GPS Interface Specification,* IS-GPS-200F, Sep. 2011. [Online]. Available: http://www.gps.gov/technical/icwg/IS-GPS-200F.pdf

[2] E. D. Kaplan, *Understanding GPS: Principles and Applications*. Norwood, MA, USA: Artech House Publishers, 1996.

[3] *GLONASS Interface Control Document, Edition 5.1*, 2008. [Online]. Available: http://www.spacecorp.ru/upload/iblock/fc4/IKD-redakcia%205.1%20ENG.pdf

[4] *European GNSS (Galileo) Open Service: Signal In Space Interface Control Document, Issue 1 Revision 1*, Sep. 2010. [Online]. Available: http://ec.europa.eu/enterprise/policies/satnav/galileo/files/galileo-os-sis-icd-issue1-revision1_en.pdf

[5] M. S. Braasch and A. Van Dierendonck, "Gps receiver architectures and measurements," *Proceedings of the IEEE*, vol. 87, no. 1, pp. 48–64, 1999.

[6] R. Pickholtz, D. Schilling, and L. Milstein, "Theory of spread-spectrum communications–a tutorial," *Communications, IEEE Transactions on*, vol. 30, no. 5, pp. 855 – 884, May 1982.

[7] *3GPP2 C.S0036-0 v1.0, Recommended Minimum Performance Specification for C.S0022-0 Spread Spectrum Mobile Stations*, 2011. [Online]. Available: http://www.3gpp2.org/Public_html/specs/C.S0036-0_v1.0.pdf

[8] P. Misra and P. Enge, *Global Positioning System: Signals, Measurements, and Performance*. Lincoln, MA, USA: Ganga-Jamuna Press, 2001, pp. 296–298.

[9] B. Widrow, "A study of rough amplitude quantization by means of nyquist sampling theory," *Circuit Theory, IRE Transactions on*, vol. 3, no. 4, pp. 266–276, 1956.

[10] O. Julien, M. E. Cannon, G. Lachapelle, C. Mongrédien, and C. Macabiau, "A new unambiguous boc (n, n) signal tracking technique," in *Proceedings of The European Navigation Conference GNSS*, 2004, pp. 17–19.

[11] V. A. Veitsel, A. V. Zhdanov, and M. I. Zhodzishsky, "The mitigation of multipath errors by strobe correlators in gps/glonass receivers," *GPS Solutions*, vol. 2, no. 2, pp. 38–45, 1998.

[12] A. Van Dierendonck, P. Fenton, and T. Ford, "Theory and performance of narrow correlator spacing in a gps receiver," *Navigation*, vol. 39, no. 3, pp. 265–283, 1992.

PUBLICATION 2

Ville Eerola, "Rapid Parallel GPS Signal Acquisition," in *Proceedings of the 13th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GPS 2000)*, Salt Lake City, UT, Sep. 2000, pp. 810–816.

Rapid Parallel GPS Signal Acquisition

Ville Eerola, *VLSI Solution Oy*

BIOGRAPHY

Ville Eerola was born in Finland. He has received the degree of Diploma Engineer in electrical engineering From Tampere University of Technology in 1990. He has worked at VLSI Solution since 1992. He has been involved in GPS receiver technology development since 1992. His research interests include spread spectrum communications and signal processing in communication systems. He is a co-founder of VLSI Solution Oy.

ABSTRACT

A GPS acquisition system which achieves extremely short search times is presented. The system requires no prior knowledge of position or time. Using a 50 MHz clock frequency, 25 satellites can be searched in parallel. Under nominal signal conditions, two-second search times have been demonstrated. The acquisition unit performance has been verified both with live satellites as well as with a GPS simulator. With the GPS simulator the performance with weaker signal conditions was also evaluated. The acquisition threshold was found to be −134 dBm with a RF noise figure of 4 dB.

The developed acquisition system is based on parallel matched-filter implementation, which is capable of searching code offsets of a number of satellites simultaneously. The frequency domain is searched by sweeping a common local oscillator frequency. The search process is controlled by a state machine, which has a number of parallel channels, each capable of performing a search of its corresponding PRN code. This implementation is considerably more effective than performing sequential search of each satellite signal. The modest hardware complexity increase and the search speed improvement compared to the single-satellite search unit makes the parallel architecture attractive.

The advances in current CMOS manufacturing technologies make fully digital implementation of a matched filter based acquisition unit economically possible even for low-cost GPS receivers. The acquisition system has been implemented in 0.35 µm CMOS process. A sensor GPS receiver containing the acquisition unit, 12 channel correlating receiver together with the DSP and SRAM memory for running the tracking algorithm fits in a single chip and is packaged in a 64-pin LQFP package.

1. INTRODUCTION

An important design parameter for GPS receivers is the Time To First Fix (TTFF). In usual GPS applications, the receiver has some knowledge of its location, the current time, and the approximate satellite orbit parameters. It can use this information to aid in the initial signal search. If this information is not available, the carrier frequency search should cover at least the full Doppler uncertainty, the code phase search should cover all 1023 chips, and all possible satellites should be searched for. This kind of search can take several minutes to perform without aid of special hardware.

The full sky search problem has been an uninteresting area for many GPS receiver designers because of its infrequent need, as the location, the current time, and the approximate satellite orbit parameters are usually available. However, there are some applications where providing this information would be hard or uneconomical. For example, the GPS receiver can be used as a sensor for determining location and speed of an object which is destroyed or lost after each use. In such case the receiver needs to be small, self-contained, low-cost, and usually has no way of uploading any information to it prior usage. In many cases, the signal acquisition speed is critical for successful operation. This kind of sensor receivers usually operate in differential mode where a base station with a continuously operating GPS receiver performs the differential position computations. Thus, only the time-stamped receiver measurements are needed for proper operation, and the cold-start time is not restricted by time required to receive the ephemerides.

The developed acquisition unit described in this paper is for C/A-code GPS receivers aiming for rapid signal acquisition when no information on the location, time, and satellite orbits are available for the receiver, and which need acquisition of signals from at least 4 satellites within a couple of seconds from turning on the receiver. The sensitivity of the acquisition process was not critical in this application,

as the receiver is to be used in open sky environment. The acquisition unit is based on parallel matched-filter implementation, which is capable of searching code offsets of a number of satellites simultaneously. The advances in current CMOS manufacturing technologies make fully digital implementation of a matched-filter-based acquisition unit economically possible even for low-cost GPS receivers. The application for which the presented acquisition unit was developed is cost sensitive, but it still requires high signal acquisition speed, which makes the described architecture well suited for it.

The acquisition unit is a part of a full GPS receiver baseband chip with the acquisition unit, 12 channel correlating receiver together with the DSP and SRAM memory for running the tracking algorithm. The block diagram of the chip is shown in Fig. 1. The receiver is intended to be used in differential configuration where a base station with a continuously operating GPS receiver performs the differential position computations. The receiver chip does not have enough memory to run the full navigation processing. The receiver still requires a proper signal tracking to achieve the required measurement accuracy, as the matched filter can measure the code offsets with only about one-chip accuracy. The whole receiver baseband fits in a single chip and is packaged in a 64-pin LQFP package and has been implemented in a 0.35 µm CMOS process.

2. MATCHED FILTERS

Matched filters [1] are devices which continuously compute correlation between a known (reference) signal and a signal to be measured, and give maximal output when the correlation between the reference signal and the incoming signal is strongest. They can be implemented either as a continuous time or discrete time operation. By definition, they are optimum detectors for signals embedded in additive white Gaussian noise (AWGN). Thus a matched filter (MF) is a useful device to be used in the acquisition phase of spread spectrum receiver operation to search the correct timing for the replica code.

The bandpass matched filter is implemented using the low-pass equivalent format for bandpass filters (Fig. 2). The implementation of a matched filter for finite-length pseudo-random-number (PRN) waveform is most easily visualized in the form of a tapped delay line followed by a passive filter matched to a *single* PRN waveform as illustrated in Fig. 3. The output filter (P*(ω)) is matched to the basic pulse shape of the PRN spreading chips. In the case of rectangular chips it can be ignored.

3. ALGORITHM

The basic operation of the MF acquisition system is as follows: The incoming samples are multiplied by sine and cosine signals produced by a numerically controlled oscillator (NCO). The frequency of the NCO is equal to the GPS

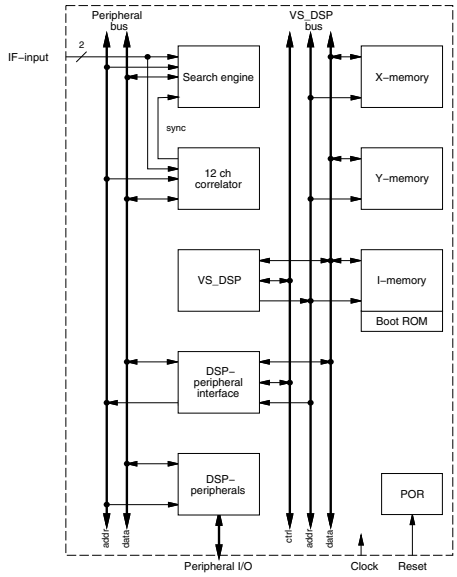


Figure 1: Block diagram of the GPS receiver baseband chip.

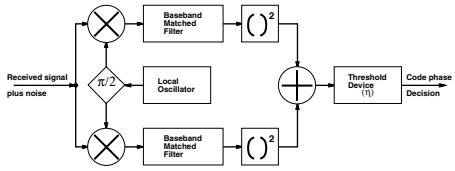


Figure 2: A band-pass version of a matched filter acquisition system using low-pass filters.

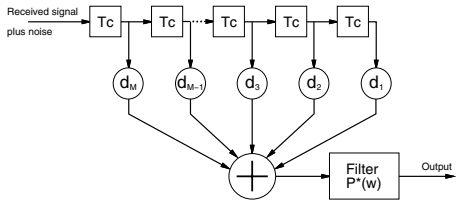


Figure 3: A tapped delay line implementation of a matched filter for an M-chip PRN sequence.

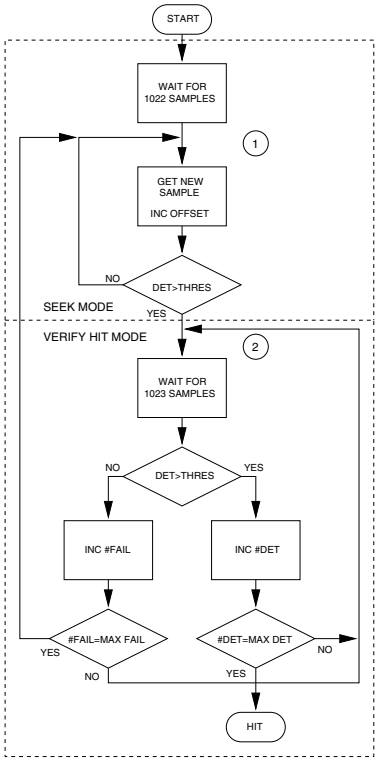


Figure 4: Flowchart of the MF search algorithm.

satellite Doppler plus the intermediate frequency (IF). The detection and verification process is performed independently for each PRN signal. The flowchart of the acquisition process is shown in Fig. 4. The used algorithm implements an M out of N majority logic decision with early termination.

In the first step the shift register is loaded with data from the input block one sample at a time. As the length of the matched filter is 1023 samples, we need to load 1022 samples at this stage. In the next step (1) one more sample is loaded into the shift register. Now, the data in the shift register is compared to the reference PRN signal in the matched filter. If the threshold is not exceeded, we go back to point (1) and load the next input sample. If the comparison value exceeds the set detection threshold, a potential match is noticed. In the simplest case exceeding the threshold would mean that the signal has been found, and its code phase is aligned with the reference signal in the matched filter. However, in most practical cases the signal to noise ratio of the received signal is too low to get a reliable detection on single comparison. To increase the detection probability and decrease the probability of false detection a number of verification comparisons are done. So, if a possible hit was found, the system enters verification mode at point (2). There the shift register is first loaded with 1023 fresh data samples to improve detection statistics and to arrive at the same code phase position. After the wait, the threshold comparison is repeated. If the threshold is not exceeded, the value of the *fail* counter is incremented. The counter is compared against a set maximum value, which sets a limit on failed verification comparisons. If the value of the counter reaches the limit, the verification is considered as failed and the search is continued in search mode by loading the next input sample at point (1). Otherwise, the verification step is repeated by going back to point (2). If the comparison value is above the threshold, the value of the *det* counter is incremented. The counter is then compared against a set maximum value, which sets a limit on successful verification comparisons. The signal is declared as found if the value of the counter equals to the limit and the search ends. Otherwise we go back to point (2) to repeat the verification.

The maximum values for the *fail* and *det* counters set the limit of total verification rounds to the sum of the limits plus one. The optimization process for the limits is described in more detail in [1]. The threshold value selection process needs to consider the desired detection and false alarm probabilities. One method for this is presented in [2].

After checking all possible code offsets for all the parallel channels, the NCO frequency is increased by a given value, and if it is not above the maximum value, the search process is repeated. When the frequency reaches the maximum value, it is set to the starting value, the set of PRN sequences to be used is changed, and the search starts from the beginning. Waiting for all channels to check all possible code offsets means that some of the channels may in fact search each frequency bin more than once, and the time between frequency sweeps depends on the slowest channel getting all of the code phases checked.

4. HARDWARE

The MF acquisition system implements a parallel matched filter with integrated search control. The user only needs to configure the main search parameters such as the frequency and detection limits, and the MF acquisition system independently searches the whole uncertainty space. The results of the search are reported to the user through the DSP interface.

The MF acquisition system implements the 25 parallel channels in a time-shared manner. The limit in the number of channels comes from the ratio of the master clock fre-

quency to the required sampling frequency of the MF. In this case the ratio is 50. The single MF core processes both in-phase and quadrature versions of each channel, reducing the number of possible channels to 25. Since the number of possible channels is smaller than the total number of possible PRN sequences to search, the PRN sequences being searched must be changed periodically. The PRN signals to be checked are stored in a ROM, and the time-multiplexing of the reference signals is done by incrementing an address counter for the ROM. The counter counts from a base address up 25 times modulo 32. The change of the set of PRN signals to be searched is done by changing the base address, which is done after sweeping through all frequencies to be searched. The base is changed by a configurable increment, which is again counted modulo 32. The configurable base increment allows for tuning the search process.

There are several events within the MF, which generate output allowing the user to be aware of the search progressing. The generated event types are:

PRN Found In this case the search was successful. The found counter exceeded the preset maximum value. The PRN number, code phase offset, frequency and hit/failure statistics are available for the user.

Not Found In this case the search was not successful. The failure counter exceeded the maximum value. The PRN number, code phase offset, frequency and hit/failure statistics are available for the user.

Next Frequency This is to inform that all 25 PRN channels have checked every code phase offset, and the frequency is incremented by the given value. The new frequency is available for the user.

Next Base This is to inform that all 25 PRN channels have checked every frequency, and the base is incremented by the given value. This event occurs at the same time than the last frequency sweep event, and they are combined into one data packet.

The events reported to the user can be configured by using the MF configuration register. The only event that is always reported is the *PRN found* event.

The architecture of the GPS acquisition system is shown in Fig. 5. The top level is composed of four blocks: the datapath, which contains the actual matched filter implementation; the control block, which generates the necessary control signals for the other blocks; the state machine controlling the search algorithm; and the I/O block interfacing the MF to the on chip bus.

4.1. Datapath

The matched-filter datapath block contains the MF datapath proper, the MF input processing block, the ROM address generation unit, and the local oscillator NCO. The

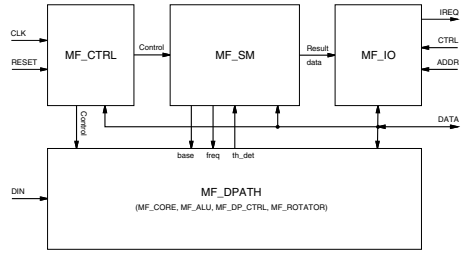


Figure 5: Block diagram of the MF unit.

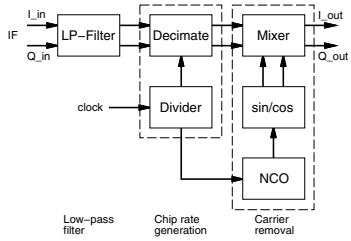


Figure 6: Block diagram of the MF input processing block.

MF input block, illustrated in Fig. 6, contains a complex low-pass filter, a decimator, an IF local oscillator, and an IF mixer. The IF mixer performs the complex multiplication of the input signal by a locally generated carrier replica signal to remove the Doppler residual. The input processing consists of decimation of the 1-bit input signals from the original sampling rate of 51.15 MHz and rotating the complex signals phase with sine and cosine local oscillator signals produced by the NCO. The NCO runs at the MF sample rate (1.023 MHz) and has 16-bit frequency resolution (≈ 15.6 Hz).

The matched-filter datapath, shown in Fig. 7, is the core of the whole matched-filter acquisition system. The MF design is of the low-pass type and the datapath arithmetic is time-multiplexed to process both the in-phase and the quadrature channels. The matched-filter length in this implementation is 1023 samples, corresponding to the full PRN code length. The incoming 1-bit I- and Q-data streams are loaded into two parallel shift registers and matched against the reference signals stored in a ROM. The ROM addresses are generated by an address generation unit, which enables the MF to process multiple PRN searches in parallel by time-multiplexing the used reference signals. The data and the reference signal values are compared by an XNOR gate whose output is one if its two inputs are the same. The single MF tap implementation has two shift-registers bits for the input data streams, and a

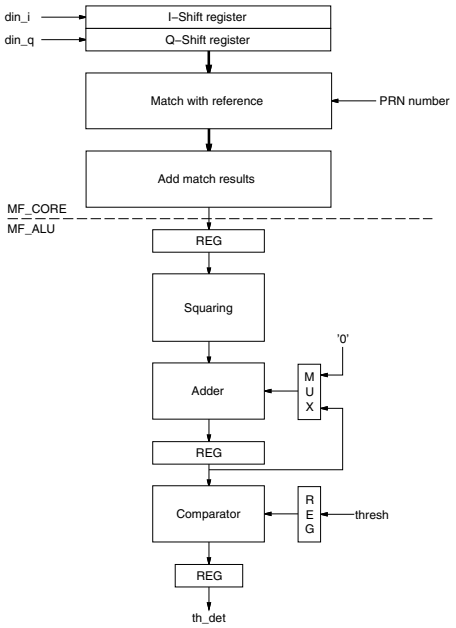


Figure 7: Block diagram of the datapath of the MF search engine.

32 by 1 ROM for holding the reference PRN signals. The data registers are compared to the reference PRN data with an XNOR gate. The comparisons are done in alternating order, i.e. first the I-channel data, and the the Q-channel data.

After the comparison, 1023 data values of 1-bits each need to be added together for every sample to produce the output of the MF. This is the most challenging part of MF implementations. Adding together the 1-bit comparison results gives the number of successful comparisons. However, it is possible to have a total match of the PRN code to the input signal, with just the sign of the input data reversed. In this case we get zero successful comparisons. In fact, the worst case match occurs when exactly half of the comparisons are wrong. To accommodate this we need to get a result that is signed and whose values are between plus and minus half of the MF length (−512 to +511).

After adding the compared bits and subtracting the offset, a pipeline register holds the result before squaring. The squared MF results are added together two at a time to combine the I- and Q-branch MF results. The combined result is finally compared by a configurable threshold value,

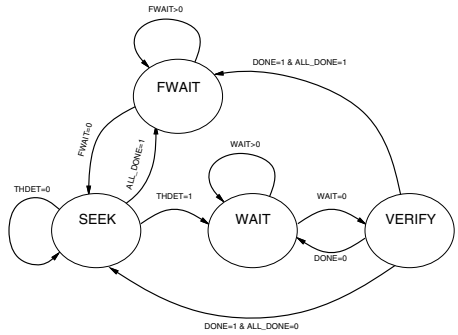


Figure 8: State diagram of the MF state machine.

and the comparison result is the output of the MF datapath.

4.2. The State Machine

The MF state machine is responsible for the high level search control of the MF. It implements 25 parallel operating state machines, each responsible for search of one PRN signal. The operation of the state machine channels is independent except for the synchronization of frequency and PRN base addresses, which happens only when each of the channels have processes all of the possible PRN code phase offsets once.

One channel of the MF state machine is illustrated in Fig. 8. There are two active states and two states used for waiting. The starting state is the **fwait** state, in which new data is clocked into the shift-registers. The wait in this state lasts for as many sample clock cycles as there are bits in the shift register. After the sweep wait the **seek** state is entered. While in this state the threshold detector output is checked on every sample, and if the threshold was exceeded the next state, **verify wait** state, is entered, the *hit* counter is set to one, and the *fail* counter is set to zero. If the last code offset was reached, the state machine sets the `done` flag for the current PRN channel. When all the state machine channels have checked every possible code phase offset, i.e. when all the `done` flags are set, the frequency is swept, optionally the PRN address base register is updated, and the **fwait** state is entered.

In the **verify wait** state the state machine waits for the data shift register to shift in totally new data in order to improve the detection statistics, and to reach the same code offset position. In order to verify the satellite detection, the threshold comparisons are repeated multiple times at the same code offset position. After waiting in the **verify wait** state for the length of the PRN sequence, the **verify** state is entered. In this state, the threshold detector value is checked, the *hit* or *fail* counter is incremented. If the num-

ber of hits and fails are still below their respective maximum values, the **verify wait** state is entered again. If the number of hits is above the maximum value, a satellite signal is declared as found. After the last verify, if the code offset is the last one, the `done` flag is set and the **fwait** state is entered if all the flags are set. Otherwise, the **seek** state is entered again, and the search continues normally.

The MF state machine sweeps the frequency between the low and high limit values by configurable steps. The frequency is the fixed IF plus the satellite Doppler frequency, so the limits should be set according to the real RF front-end IF and the largest expected Doppler shift. In addition, the *hit* and *fail* counter limits can be configured through a configuration register. There is also a mechanism for initializing the MF system to its initial state. The code offset used in reporting is relative to the local clock counter of the GPS receiver chip.

5. ADVANTAGE OF PARALLELISM

The MF-based implementation provides orders of magnitude improvement over traditional serial-search based on correlator implementations as it is equivalent to a set of correlators operating in parallel.

To evaluate the advantages of parallel MF acquisition unit, a set of simulations has been performed. The simulations were run with varying number of PRN codes searched simultaneously. The results of the simulation are shown in Fig. 9, which shows the search overall time versus the number of parallel channels for two threshold value settings. Using the higher threshold value increases acquisition speed as can be expected. It can clearly be seen that even using only two parallel channels improves the acquisition speed dramatically. The total search time is almost half of the search time for single PRN code at a time. The advantage levels off when more channels are added, but the optimum is to search all possible codes simultaneously. The two lines show acquisition times for different threshold values and it can be seen that the relative advantage is similar independent of the threshold values.

To get another view of the same simulation Fig. 10 shows the search time of a single frequency for one set of parallel PRN codes (solid line). It shows that the time to go through all possible code phases increases with the number of parallel channels. However the time required for each PRN code decreases as shown as the dashed line. This means that this implementation is less effective than a number of parallel, completely independent search channels. However, it is still considerably more effective than performing sequential search of each satellite signal. The modest hardware complexity increase and the search speed improvement compared to the single-satellite search unit makes the parallel architecture attractive.

The acquisition unit performance has been verified both

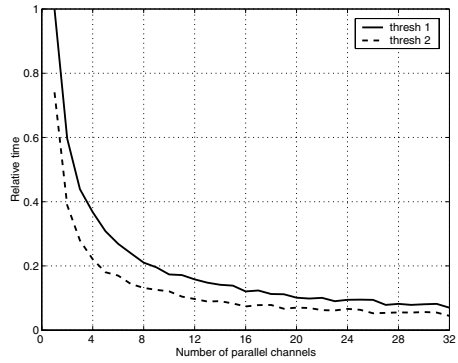


Figure 9: Relative acquisition times versus number of channels.

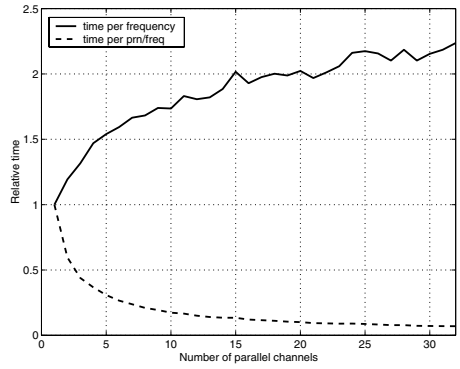


Figure 10: Relative acquisition times for one frequency sweep versus number of channels.

with live satellites as well as with a GPS simulator. Under nominal signal conditions, two-second total search times have been demonstrated. This includes two frequency sweeps with 25 PRN codes each. With the GPS simulator the performance with weaker signal conditions was also evaluated. The acquisition threshold was found to be −134 dBm with a RF noise figure of 4 dB.

6. CONCLUSIONS

A rapid GPS acquisition system requiring no prior knowledge of position, time or almanacs was presented. The system operates autonomously, and is based on a parallel matched-filter structure. The system has been implemented in CMOS and is used as a building block for a GPS-based sensor application. The acquisition speed has

been found to be excellent. The MF-based implementation provides orders of magnitude improvement over traditional serial-search based on correlator implementations. Furthermore, the parallel MF architecture offers a significant acquisition speed improvement over a single-PRN search approach.

REFERENCES

[1] Marvin K. Simon, Jim K. Omura, Robert A. Scholtz, Barry K. Levitt. *Spread Spectrum Communications Handbook, rev.ed.* McGraw-Hill, 1994. pp 815–832.

[2] A. J. Van Dierendonck "GPS Receivers." Chap. 8 in *The Global Positioning System: Theory and Applications*, Vol I, B. W. Parkinson and J. J. Spilker Jr.,eds., American Institute of Aeronautics and Astronautics, Reston, VA, 1996. pp 402–405.

PUBLICATION 3

Ville Eerola, "Optimizing Matched Filters for GNSS Receivers," in *2017 International Conference on Localization and GNSS (ICL-GNSS)*. IEEE, 2017.
DOI: 10.1109/ICL-GNSS.2017.8376238

Optimizing Matched Filters for GNSS Receivers

Ville Eerola
u-Blox Espoo Oy
Itsehallintokuja 6
FIN-02600 Espoo, Finland
Email: ville.eerola@u-blox.com

***Abstract*—This paper presents methods for optimizing matched filter based GNSS acquisition units. Matched filters become computationally complex when the reference signal length is increased. They have a fixed length in time, which also limits the achievable integration time. In this paper, we show how to design an optimal adder for a large number of single-bit values, and how this can be utilized to build matched filters for multi-bit numbers and multiple inputs. By altering the incoming sample rates, it is possible to adapt the matched filter for changing signal frequency. Employing coherent and non-coherent integration after the matched filter allows enhancing the sensitivity of signal detection with modest implementation size increase.**

I. Introduction

At the turn of the century, it became feasible especially in the economic sense to implement full code searches in GNSS hardware [1]. A further driver for this was the FCC e-Call (E911) mandate [2]. The cold start problem with rapid, full code uncertainty search had been an uninteresting problem before. The cold start Time To First Fix (TTFF) performance had not been viewed as a critical or important performance parameter for GNSS receivers, which had typically been used in continuous tracking mode. Thus, the cold start had been an infrequent event. For improving the search performance of full code uncertainty, the code search had to be parallelized. A term *massive parallel correlation* for a device which could search for the code phase with full code period uncertainty was devised and proposed first by Dr. Van Diggelen [3].

One implementation of these devices is the matched filter (MF). Matched filters [4] are devices which continuously compute the correlation between a known reference signal and the received signal and produce the maximal output when the correlation is the strongest. By definition, they are optimum detectors for signals embedded in additive white Gaussian noise (AWGN) [5]. The structure of a MF is shown in Fig. 1. The incoming signal is first captured in a tapped delay line, and the output of each tap is multiplied by the corresponding coefficient from the Pseudo Random Noise (PRN) sequence before being added together. The summing function is followed by a passive filter matched to the basic pulse shape of an individual chip. This filter can be ignored for rectangular chips.

The matched filters are attractive for GNSS receivers especially for acquiring signals with full code period uncertainty [6]. Unfortunately, matched filters become very large in silicon area wise, when the reference signal length increases, and the computational complexity tends to be high due to the large

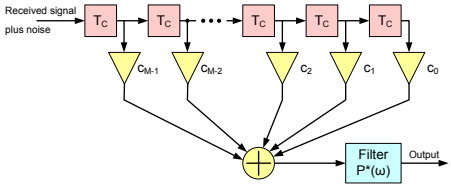


Fig. 1. A tapped delay line implementation of a matched filter for an M-chip PRN sequence.

number of operations needed to produce each output sample. To gain sensitivity, the signal integration time can be increased, but unfortunately this could mean increasing the length of the MF further. The MF is fixed to a reference signal with fixed length in time. Doppler and receiver clock frequency variations cause the apparent received signal to vary, which can lead to matching problems especially with longer integration times. This paper aims to show how these shortcomings can be overcome.

In the next section, we will explore a method of implementing the basic MF block very efficiently and how this basic optimized MF core block can be utilized to build more complex acquisition blocks. We will also explore other methods for optimizing the MF based GNSS receiver block in section III.

II. Optimizing the MF Block Implementation

Looking at the structure in Fig. 1, we can assess the difficulty of implementing the MF in hardware. Usually, the MF length is equal to a single PRN code period to offer the best acquisition performance. Otherwise we will suffer from worsening of the cross-correlation performance, which makes it more difficult to separate between signals from different satellites. We could also detect incorrect timing of the incoming signal due to the non-ideal autocorrelation properties of partial PRN codes, which would lead to failed acquisition. The PRN code lengths for civil GNSS signals intended for direct acquisition vary between 511 (GLONASS) to 4092 (Galileo). The MF must produce an output for each incoming chip at minimum, which means that the input and the output need to be updated at the chipping rate, which is between 511 kHz (GLONASS) and 2.046 MHz (BeiDou).

For GNSS applications we can typically assume that the tap multiplications are trivial to implement, and there are few bits per sample at the MF input stage. The summation element needs to add as many inputs as there are taps in the filter within the time between two output samples, which turns out to be in the range of $4 \cdot 10^9$ operations/s. There are three possible ways to solve this issue:

1) Pipeline the MF implementation by storing partial results
2) Use transposed filter structure, where the summation is fully pipelined
3) Implement the summation in a smart way

The first two options need to store intermediate MF output values, which require a large number of bits, and thus are wasteful on hardware area and power. Furthermore, using area efficient storage such as RAM is not feasible as we need parallel access of all the values, which is not possible with RAM. Trying to time-multiplex the MF calculation logic only makes the problem harder. Thus we will have a look at the third option. First we will start by developing a one-bit MF core, and then we will expand it to multi-bit input and multiple inputs.

A. One-Bit Input and Coefficients

For the MF implementation, we should consider the input and the coefficients being signed, and thus the interpretation of the input to the adder needs to be considered. For single bit values, one usual representation is:

$$x_s \in \{-1, +1\} \rightarrow x_b \in \{1, 0\} \tag{1}$$

This mapping can be implemented by the equations:

$$x_b = (1 - x_s)/2 \quad \text{and} \quad x_s = 1 - 2x_b \tag{2}$$

With this mapping, we can use the logic XOR operation ($\oplus$) to perform the multiplication between the input data and the coefficients. Usually, the output will be further processed using normal multi-bit arithmetic, and we will need to map the output back to signed numbers. This can be done easily by noting that:

$$\begin{aligned} \mathrm{MF}_s(x_s(n)) &= \sum_{k=0}^{N-1} x_s(n+k)c_{s,k} \\ &= \sum_{k=0}^{N-1} (1 - 2(x_b(n+k) \oplus c_{b,k})) \\ &= N - 2\mathrm{MF}_b(x_b(n)) \end{aligned} \tag{3}$$

To save output bits, we could also notice that we could save one bit in the word length of the following blocks by scaling the signed output by two. As the input part of the MF is easily implementable with simple logic, we only need to implement adding the bits together efficiently.

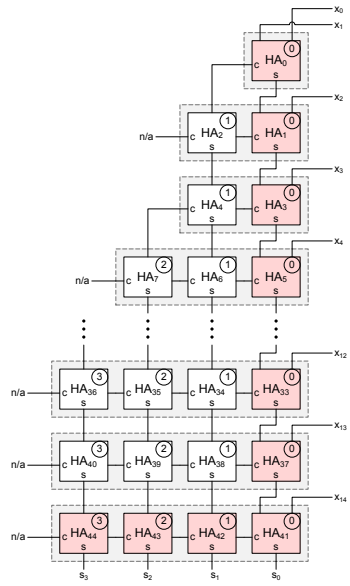


Fig. 2. 15-input linear multi-input adder with ripple carry. (HA=half adder)

B. Reduction Adder

We will start by considering the problem of adding N 1-bit numbers. This will later be expanded to multi-bit numbers. The trivial implementation is to just loop through the inputs and add them into the running sum. This is illustrated in Fig. 2 for an example case of adding 15 inputs. The figure omits the middle 7 stages of the circuit for brevity. The resulting circuit has one less adders as there are inputs, and the word length of each adder will be as big as it is needed to hold the maximum result at that stage. As we can see, it uses a lot of adder stages and creates a very long critical path (marked with red colored cells). As the inputs are only 1-bit wide, most of the circuit is just propagating the carry with the result of previous addition. A smarter way to implement the function would be to organize the addition in a binary tree with two-input adders of increasing width. This will offer a significant improvement to the length of the critical path as well as a reduction of the needed 1-bit adders. The number of levels in the tree is $\lceil \log_2 N \rceil$. If the number of values to be added at each stage is odd, then one of the values will be added at a later stage. Each level of the circuit will need an additional bit of word length compared with the previous one. This adder structure is shown in Fig. 3.

We can take the tree structure approach further by noticing that the full adder cell takes 3 inputs and provides a two-bit

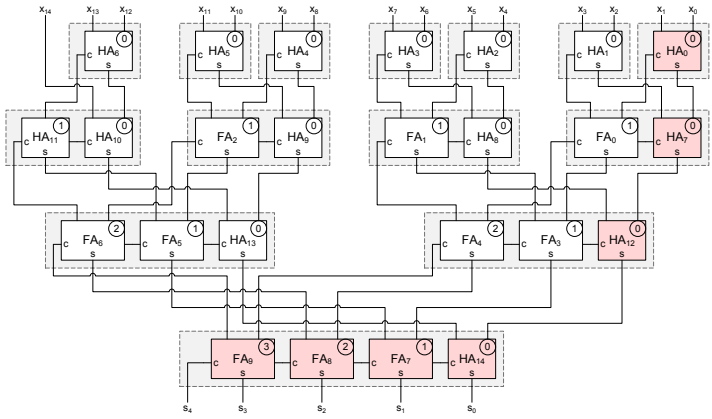


Fig. 3. 15-input binary tree reduction adder with ripple carry adders. (HA=half adder, FA=full-adder)

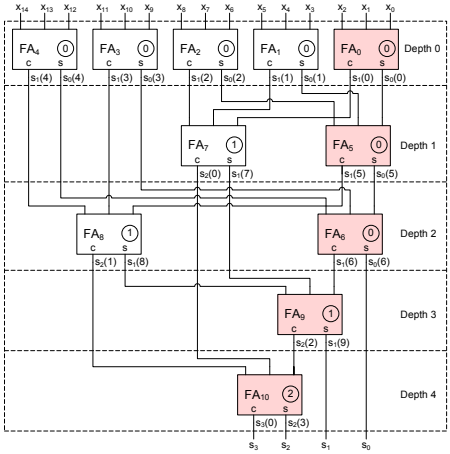


Fig. 4. 15-input binary reduction adder (bitcount).

result. We can build a network of one-bit full-adders in a tree structure which allows us to reduce the depth by $\log_2 3$ and achieve a relatively low delay in the order of $\mathrm{O}\big(\log_3 N + \log_2 N\big)$ times the delay of the single full-adder delay. The $\log_3 N$ term is due to the number of inputs that need to be added to get the LSB bit, and the $\log_2 N$ term is due to the final carry propagation chain from the final adder producing the LSB of the result to the MSB bit of the result. This number is equal to the number of bits in the result. The generation of the reduction adder network is a relatively straightforward process: To generate the LSB of the result, we first add all the input bits with full-adders which produce $N/3$ sum and carry signals. Then we add all the sum signals together again, and repeat the process until there is only a single sum signal left. This is the LSB of the result. To generate the next bit, we perform the same process to the carry signals produced by computing the LSB bit. We need to repeat the process until we have a single carry bit left. This is the MSB bit. In selecting the signals to be added a simple rule must be followed to get the shortest possible delay through the network. The rule is that one should always select the inputs that have the shortest path from the primary inputs. The resulting structure for adding 15 inputs is shown in Fig. 4. The bit weights are indicated with numbers in circles and the signal depth is indicated on the right. The intermediate signals are labeled according to the output bit position (subscript) and assignment order (in parenthesis). The sum and carry outputs of the full-adders are indicated by s and c respectively. It is possible to compute the number of elements we need to implement an N-input adder. When using normal full-adders, we take 3 inputs and produce 2 outputs with 2 different bit weights. To sum all N input bits to produce a single LSB we need to reduce the input by $N-1$ times. Each full adder reduces the number of inputs by 2. Thus we need $N_0 = (N-1)/2$ adders at each bit weight. To handle the case where N is even, we need to use one additional adder. As each adder also produces a carry bit output, there will be as many carry outputs for the next level as there are adders. For the next level, we perform the same reduction again. This time we need $N_1 = \lceil (N_0-1)/2 \rceil = \lceil (\lceil (N-1)/2 \rceil - 1)/2 \rceil$ adders.[1]

[1] $\lceil x \rceil$ is the ceiling of x, i.e. the smallest integer n such that $n \geq x$.

TABLE I
COMPARISON OF DIFFERENT MF ADDER IMPLEMENTATIONS

	Linear		Tree (ripple)		Reduction Tree	
Inputs	**size**	**carry**	**size**	**carry**	**size**	**carry**
15	23	18	18	7	11	5
31	62	35	41	9	26	7
511	2044	519	757	17	502	13
1023	4604	1032	1524	19	1013	15
2046	10230	2056	3057	21	2036	16
4092	22505	4103	6125	23	4082	18

This translates to approximately $N_i \approx (N - 2^{i+1} + 1)/2^{i+1}$ adders at each bit weight level. We can now easily compute the approximate total number of adders needed:

$$N_{\text{add}} = \sum_{i=0}^{\lceil \log_2 N \rceil} \frac{N - 2^{i+1} + 1}{2^{i+1}} \approx N - \log_2 N \qquad (4)$$

When the number of inputs becomes large, the generation of the adder network becomes a tedious process if performed by hand. To generate the large networks needed for the MF implementation, a program was developed to generate the VHDL implementation of the reduction adder.

A comparison of the reduction adder implementation against the trivial linear adder and the binary adder tree is shown in Table I. The table shows the resulting size and propagation delay with respect to a full-adder for a few of different numbers of inputs. It is clear that this is the worst possible implementation in both the size and the delay. The tree adder provides more than 3 times size reduction and cuts the propagation delay to roughly $2\log_2 N$ from $N + \log_2 N$ of the trivial implementation. The reduction adder tree offers additional reduction in size compared with the binary tree adder cutting by a factor of 1.5, and some reduction to the propagation delay due to the shallower depth of the tree ($\log_3 N$ vs. $\log_2 N$).

C. Extension to Multi-Bit Inputs

The preceding discussion was only about adding 1-bit inputs. It is possible to use the summation block explained above as a building element for a system with either larger input word length, or equally well multiple inputs. Handling multi-bit inputs can be done by either extending the above algorithm for generating the adder tree to consider multiple input bit weights when assigning the adder elements, or using time-multiplexing for processing each input bit weight at a time, and then include a post processing stage which combines the individual 1-bit sums. The latter also suggests a way of handling larger additions by smaller summation blocks. As the aim is to keep the hardware size manageable, we will explore the second solution.

We represent any non-negative valued integer n as a linear combination of $M = \lceil \log_b n \rceil$ powers of the basis b:

$$n = \sum_{i=0}^{M-1} n_i b^i, \qquad 0 \le n_i < b \qquad (5)$$

Where $\{n_0, n_1, \ldots, n_{M-1}\}$ is the M-digit representation of the number. As is typical in digital circuits, we will use binary numbers ($b = 2$, $n_i \in \{0, 1\}$) in the following discussion. To extend the representation to cover negative numbers, we have a few alternatives. To evaluate them, we need to take a deeper look at how the coefficient multiplication works. We will only consider coefficients with values ± 1, as this is the most frequent and the simplest case, and the multiplication reduces to a sign change. The simplest number format, the sign-magnitude, just means that the highest digit is interpreted as the sign of the number. Unfortunately, this does not translate to a linear combination of the digits, but the sign needs to be handled separately, which means that it is unsuitable for our MF use case. Perhaps the most used format is the two's complement, where we assign a weight of -2^{M-1} to the highest digit, and can use the linear combination principle. Unfortunately, the method for negating the number is somewhat complex requiring us to complement each digit ($\overline{n_i} = 1 - n_i$), and add one to the result. This format makes the sign reversal for the coefficient multiplication unnecessarily complicated. By using one's complement representation, we can negate a number by simply complementing all of its digits. This makes it very easy to implement the coefficient multiplication within the MF. Fortunately, one's complement numbers can also be represented as linear combination of the digits. We can easily see that we can assign a weight of $-(2^{M-1} - 1)$ to the highest digit. Complementing a binary digit means just inverting the bit. We can now write our signed numbers as:

$$x_s = \sum_{i=0}^{M} x_i w_i, \qquad w_i = \begin{cases} 2^i & \text{for } i < M, \\ -(2^{M-1} - 1) & \text{for } i = M \end{cases} \qquad (6)$$

Now considering the operation of MF using one's complement binary numbers with coefficient values $c = \pm 1$, and using the mapping from (1) we can write:

$$x_s \cdot c_s = \left(\sum_{i=0}^{M} x_i w_i \right) \cdot c_s = \sum_{i=0}^{M} (x_i \oplus c_b) w_i \qquad (7)$$

Now applying the MF to the signed M-digit numbers can be written as:

$$\begin{aligned} y_s(n) = \text{MF}_s(x_s(n)) &= \sum_{k=0}^{N-1} x_s(n+k) c_{s,k} \\ &= \sum_{i=0}^{M} \left(\sum_{k=0}^{N-1} x_i(n+k) \oplus c_{b,k} \right) w_i \\ &= \sum_{i=0}^{M} \text{MF}_b(x_i(n)) w_i \end{aligned} \qquad (8)$$

Where $\text{MF}_b(x)$ is the binary MF utilizing one bit coefficient multiplication with single-bit coefficients. We can implement this in a time-multiplexed way using the structure illustrated in Fig. 5, where the output values need to be collected from M consecutive outputs of the MF hardware block. The basic idea for this structure was first presented by the author in [7], [8]. As the output is represented using one's complement notation,

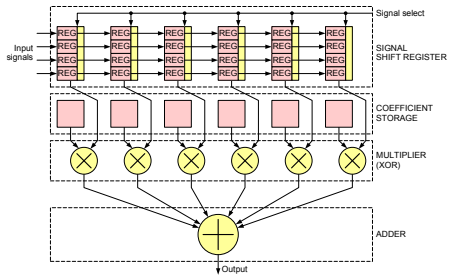


Fig. 5. Multiplexed MF input structure.

it usually needs to be transformed to two's complement format for further computations.

Using similar principles, it is possible to extend the same treatment also to handle complex valued or even separate input streams. If we first consider a matched filter with complex valued inputs ($x(n) = x_i(n) + ix_q(n)$). Its output can be represented as:

$$\begin{aligned} y(n) &= \mathrm{MF}(x(n)) \\ &= \sum_{k=0}^{N-1} \left(x_i(n+k) + ix_q(n+k)\right) c_k \\ &= \sum_{k=0}^{N-1} x_i(n+k)c_k + i \sum_{k=0}^{N-1} x_q(n+k)c_k \\ &= \mathrm{MF}\left(x_i(k)\right) + i\mathrm{MF}\left(x_q(k)\right) \end{aligned} \tag{9}$$

This can again be implemented in a time-multiplexed way with the same structure as used for the multi-bit signals, but then we need to treat the consecutive outputs as the real and imaginary parts. It is easy to see that the same idea can be extended to multiple input streams, which can be handled separately at the output. Consider for example that the MF has a limited bandwidth, end it operates on signals near DC. We could have multiple differently downshifted (in frequency) inputs to a MF to look for the signal at a wider frequency range. For this kind of use, we will use the same MF coefficients for all inputs.

III. Other Optimizations for MF Implementation

In this section, we will take a short look at a few optimizations which improve the use of the MF in GNSS applications. First, we will examine how the sensitivity can be improved by employing integration, and then we look how we can adapt the MF to changing signal frequencies.

A. Output integration

We can improve the signal detection statistics by integrating the correlated signal for longer duration [9]. The correlation can be either coherent (normal integration) or non-coherent (integrating the norm, e.g. the absolute value, of the signal). The spread spectrum modulation used in GNSS utilizes a periodic spreading code. By using a MF which is matched to the spreading code, i.e. whose length is equal to the code period, we notice that the outputs from the MF which are separated by a multiple of a code period will correspond to correlation results with identical code offset positions. We can show that adding up the MF outputs separated by one code period is equivalent to performing correlation over multiple code periods. Similarly, if we perform a norm operation (e.g. the magnitude of the complex valued outputs) we can integrate the MF outputs non-coherently for each code offset. This allows performing parallel long correlation for all searched code phases. Furthermore, as the MF output rate is equal to its input rate, we will not need to run the post-MF accumulation at a rate greater than the MF outputs become available. Also, since the MF outputs are produced sequentially, it is easy to use RAM for storing the intermediate accumulation results efficiently. The structure of such post-MF integration is illustrated in Fig. 6. Coherent and non-coherent integration is naturally performed in a time-multiplexed way and requires computational hardware for one signal. Of course, we will still need to separate storage elements for all the integration sums, but this is easily and efficiently realized as normal RAM.

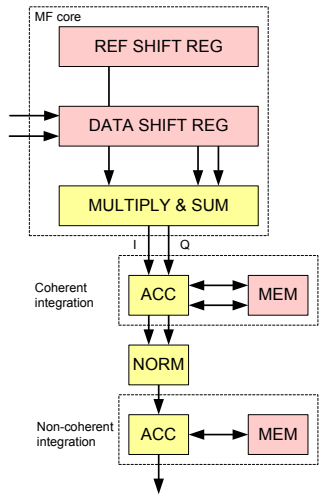


Fig. 6. Post-MF integration structure.

The post-MF integration is an efficient way to improve the acquisition sensitivity of the MF based system. By increasing the coherent integration time, the bandwidth of the signal detector becomes smaller. It is possible to implement a post-MF mixer bank to do the coherent integration at different center frequencies. This is more efficient than performing the correlation itself multiple times, as this will save the corre-

lation computation. The MF will limit the usable bandwidth to approximately reciprocal of the code period. For a wider bandwidth, we need use a mixer before the MF. Here we can utilize multiplexing of the MF inputs to save computational hardware.

B. Input Decimation to a Multiple of the Chipping Rate

The MF is matched to a fixed length signal. If the signal is sampled, then the MF will be matched to the reference signal of a fixed number of samples. The frequencies of the GNSS signal transmitted by the satellites will be subject to Doppler shift due to the relative motion between the satellites and the receiver. The effect will change the duration of the spreading code slightly from the nominal value. Also, typically the reference oscillators in the GNSS receivers are not running at exactly the designed frequency due to various nonidealities. With the post-MF integration, we can use very long integration times, and eventually it will happen that the code phase of the received signal will shift with respect to the local time base so much that the timing of the MF outputs will no longer match those in the beginning of the integration. This will ruin the post-MF integration, as the code offsets will no longer line up to the same storage elements.

Fortunately, it is possible to remedy the situation by altering the sampling frequency slightly so that it will again match to the apparent received signal code frequency. Normally a sample rate change requires decimation or interpolation filters, but in this case, the frequency change is so small that practically no aliasing or images happen, and the filters can be left out. A practical implementation has been proposed by the author in [1], [10]: The incoming signal is decimated to an integer multiple of the chipping rate before feeding it in to the MF by an integrate-dump type filter which is driven by a NCO delivering the exact chipping rate (or integer multiple of it). This method results in average an exact match of the length of the incoming chips each MF tap. This cancels out any code Doppler and it allows the MF to be kept in sync with the incoming signal. This implementation is depicted in Fig. 7. This arrangement will cause some momentary timing errors to the signal entering to the MF, but the summation of the MF will smooth them out especially if combined with post-MF integration. There are two advantages of this setup: it allows further integration of the MF outputs as the correct peak would not shift over time, and it also allows tracking the signal by selecting two outputs around the peak for the code discriminator for DLL tracking algorithm.

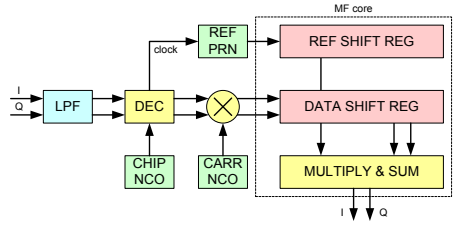


Fig. 7. Block diagram of MF input decimation to match incoming signal.

IV. Conclusions

In modern GNSS receivers the ability to acquire enough satellites for a position fix as quickly as possible has become a key performance criteria. This requires advanced massive parallel correlation methods. The matched filter is one time-domain implementation of a massively parallel correlator. We have presented two aspects of optimizing MF based GNSS receivers. By utilizing an efficient method of adding a huge number of single-bit numbers, it becomes possible to build area-efficient matched filters. Additionally, the paper presented methods to enable long coherent and non-coherent integration with MF based acquisition and tracking. These make the MF an attractive building block for high performance GNSS receivers. The presented optimization methods can be used also for building other MF-like GNSS receiver structures like the Group correlator [11].

References

[1] V. Eerola, "Rapid parallel GPS signal acquisition," in *Proc. of the 13th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GPS 2000)*, Salt Lake City, UT, Sep. 2000, pp. 810–816.

[2] *FCC Wireless 911 Requirements Fact Sheet WTB Policy*, FCC, Jan. 2001.

[3] F. van Diggelen, "Indoor GPS theory & implementation," in *Position Location and Navigation Symposium, 2002 IEEE*. IEEE, 2002, pp. 240–247.

[4] M. K. Simon, J. K. Omura, R. A. Scholtz, and B. K. Levitt, *Spread Spectrum Communications Handbook*, rev. ed. New York, NY, USA: McGraw-Hill, 1994, pp. 855 – 884.

[5] G. Turin, "An introduction to matched filters," *IRE Transactions on Information Theory*, vol. 6, no. 3, June 1960.

[6] V. Eerola and J. Nurmi, "Area estimation of time-domain gnss receiver architectures," in *Localization and GNSS (ICL-GNSS), 2014 International Conference on*. IEEE, 2014.

[7] V. Eerola, J. Takala, and T. Ritoniemi, "Rapid zero-knowledge gps signal acquisition," in *Signal Processing Conference, 2000 10th European*. IEEE, 2000, pp. 1–4.

[8] V. Eerola and T. Ritoniemi, "Matched filter and spread spectrum receiver," U.S. Patent 7,010,024, Mar. 7, 2006.

[9] F. van Diggelen, "Indoor GPS, wireless aiding and low SNR detection," Navtech Seminars, Course Notes Course 218, 2001.

[10] V. Eerola and T. Ritoniemi, "Signal acquisition system for spread spectrum receiver," U.S. Patent 6,909,739, Jun. 21, 2005.

[11] V. Eerola, S. Pietilä, and H. Valio, "A novel flexible correlator architecture for GNSS receivers," in *Proc. of the 2007 National Technical Meeting of The Institute of Navigation (ION NTM 2007)*, San Diego, CA, Jan. 2007, pp. 681–691.

PUBLICATION 4

Ville Eerola, Samuli Pietilä, and Harri Valio, "A Novel Flexible Correlator Architecture for GNSS Receivers," in *Proceedings of the 2007 National Technical Meeting of The Institute of Navigation (ION NTM 2007)*, San Diego, CA, Jan. 2007, pp. 681–691.

A Novel Flexible Correlator Architecture for GNSS Receivers

Ville Eerola, Samuli Pietilä, and Harri Valio
Nokia Corporation

BIOGRAPHY

Ville Eerola is a Senior Specialist at Nokia Corporation in Finland. He received an MSc degree in electrical engineering from Tampere University of Technology in 1990. From 1992 to 2003 he was involved in digital ASIC design and GPS receiver development at VLSI Solution and u-Nav Microelectronics. He joined Nokia in 2004. He has been involved in GPS receiver development since 1992 and has led the development of several GPS receivers. His current research interests include GNSS receiver hardware implementation. He is a co-founder of VLSI Solution Oy and u-Nav Microelectronics Corporation.

Samuli Pietilä is a Research Manager at Nokia Corporation in Finland. He received his MSc and Lic.Tech. degrees from Tampere University of Technology, Finland, in 1994 and 1996 respectively. Between 1994 and 1996 he worked for Tampere University of Technology, Control Engineering Laboratory. Since 1996 he has been with Nokia Corporation in various research and development positions. His current interests include positioning algorithms and methods for mass market devices.

Harri Valio is a Senior Research Manager at Nokia Corporation in Finland. He received an MSc degree in electrical engineering from Tampere University of Technology in 1992. He joined Nokia in 1992 and has been working in various research, development and management positions since. His current interests include positioning methods for mass market devices.

ABSTRACT

This paper describes a new architecture for the correlation processing inside a GNSS receiver. The group correlator (GC) is a code correlation device, which is optimized for parallel reception of multiple signals. It utilizes time-multiplexing to share some of the signal processing hardware amongst several independently operating channels. The GC can be used both in acquisition phase processing as well as in tracking phase processing. It can be used as the core processing block in a modern multi-system GNSS receiver due to its flexibility and efficiency.

The group correlator typically computes a number of correlation results against a reference signal for a subset of the full code period. A second stage carrier mixer and a coherent integrator can be placed after the GC for further coherent integration of the correlation results. The GC performs the correlation operation in blocks of samples, where each block represents the length of the correlation performed inside the GC. By using multiple blocks of samples the full code period can be covered. Alternatively, multiple codes can be used with a smaller correlation range. One GC contains one or more replica code inputs, each of which is split into several parts of the same size. The GC will then perform partial cross-correlations between the input data and the replica parts. Coherent integration can extend the correlation length to cover the entire replica code duration or more, even though only a part of all possible code phase offsets would be covered.

A complete GPS+Galileo receiver baseband with state-of-the-art performance based on the GC architecture is described and the parameter selection for its GC design is explained. A system simulator, which models the receiver, has been developed using SystemC and this has been used to evaluate the receiver operation in different scenarios. The performance of the designed receiver in some of the scenarios will also be shown.

INTRODUCTION

The goal of the work presented in this paper was to develop a flexible correlation processing unit that could be used in a GNSS receiver for receiving both GPS and Galileo signals simultaneously. The important system parameters for GPS and Galileo L1/OS are shown in Table 1 below.

The receiver design complexity is affected by the selection of the sample rates and signal bandwidth. For the acquisition phase, it is advantageous to use as low a sample rate as possible to maximize the code search range of the receiver with the fixed number of correlator bins.

Table 1. GNSS system parameters

Parameter	GPS	Galileo
Chip Rate (MHz)	1.023	1.023
PRN code length in chips	1023	4092
BOC code factor (samples per chip)	1	2
Minimum sample rate (MHz)	2.046	4.092
Total code search range in samples	2046	16368

For a reasonably low penalty due to the worst case mismatch of code timing, we chose to use a sample frequency that is twice the chipping frequency. For GPS this is about 2 MHz. For the Galileo L1/OS BOC(1,1) a signal of about 4 MHz would be needed for similar performance. The performance with Galileo signals with a 4 MHz sample rate would be slightly worse than the GPS performance due to the different shape of the autocorrelation function. The tracking accuracy of the receiver is a function of the pre-correlation bandwidth and the signal-to-noise ratio. It is known that a sample rate in the order of 10 MHz is a reasonable trade-off between receiver implementation cost and tracking performance. In order to simplify the design a sample rate of about 8 MHz was selected, as it is an integer multiple of the acquisition sample rates. The exact sample rates depend on the system clock selection, but as long as they are not exact multiples of the GNSS chipping rates any value reasonably close to the above rates could be chosen. In the following discussion we will use these rounded numbers when discussing the frequencies.

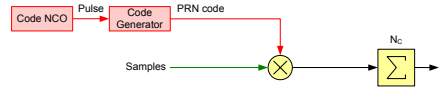
The new cellular Assisted GPS (AGPS) standards [1], [2] expect a high sensitivity with rapid signal acquisition, and the proposed design will comply with these requirements. At the same time, minimization of the silicon area is a key optimization target. The receiver could be used in both assisted as well as in standalone mode. In the AGPS mode, various levels of assistance data would be available in various situations. In the best case, microsecond-accurate time and very accurate frequency would be known, and in the worst case only the ephemeris information would be available as assistance. In order to maximize the performance in all conditions, the receiver should be able to search both the full code uncertainty of a small number of satellites as well as a small uncertainty range of more satellites. A GNSS receiver needs to perform both acquisition and tracking operations for full functionality. Additionally, in normal operation, some of the received signals are being searched for while others are being tracked. It would be desirable to be able to handle both cases with minimal hardware and control complexity.

In many older GNSS receivers correlator hardware comprising only a few code delays per channel has been used to acquire and track the signal. These kinds of receivers operate too slowly for current demands. Therefore some new approaches have been developed in which separate acquisition accelerator hardware is used for the acquisition phase and the tracking phase is implemented by traditional correlator hardware. Some solutions are also known to use correlator structures that can be configured as acquisition accelerators in one mode and as tracking correlators in another, but they are not able to perform acquisition and tracking at the same time. To solve the acquisition problem, it is necessary to implement the receiver by providing it with a large number of correlators and by using a long integration time. At present, manufacturers claim to have receivers with more than 200,000 “effective” correlators [3] or over 30,000 real-time hardware correlators [4]**.** For the sake of comparison, it should be mentioned that the first hand-portable GPS receivers only had 12 or even fewer correlators.

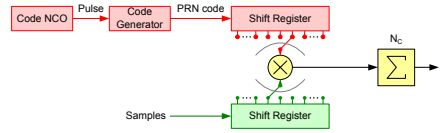
GROUP CORRELATOR

Traditionally there have been two main categories of hardware for performing time-domain correlation in direct-sequence spread spectrum (DS-SS) receivers: the correlator and the matched filter (MF). The acquisition and tracking problems in spread spectrum receivers are described e.g. in the [5],[6],[7], and the references cited therein. Correlators have been used both for acquisition and tracking while MFs are mainly used for acquisition only.

In the traditional correlator, shown in Figure 1(a), the replica code is multiplied one sample at a time and the results of the multiplication are accumulated sequentially until a desired number of samples have been processed. The numerically controlled oscillator (NCO) will give a pulse when the replica code needs to be advanced by one chip, and the code generator generates the replica code one chip at a time. This correlator structure will compute a cross-correlation between the incoming samples and the replica code at one particular time offset. If cross-correlations at different time offsets would need to be computed, the correlator structure would need to be replicated accordingly. The correlation result would be equivalent even if the incoming data and replica code would be delayed by an equal amount of time using e.g. a

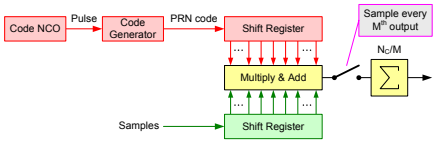


(a) Traditional correlator

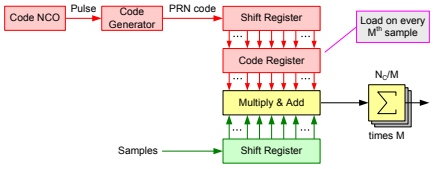


(b) Modified version of the traditional correlator

Figure 1. Block diagrams of correlator structures



(a) Basic Serial-parallel correlator



(b) Modified serial-parallel correlator for multiple correlation results

Figure 2. Block diagrams of a serial-parallel correlator

shift register (SR) as depicted in Figure 1(b). In this case we can alter the cross-correlation time offset between the replica code and the incoming samples by selecting the position where the two inputs to the multiplication are taken from. Of course, more than one multiplier and one accumulator are still needed if more than a single result is desired. The use of a shift register to delay the output of the replica code generator is widely employed in DS-SS receivers to produce different code phase offsets for correlating them with the input data stream.

We can get the same time-offset regardless of the absolute position in the shift registers of Figure 1(b) as long as the relative positions are kept constant. Thus if the shift registers are long enough, we can compute several equivalent correlations at the same time using this structure. As we can see, the multiplication results are accumulated over a time. It is mathematically equivalent to multiply and add several samples together in parallel if the final accumulation waits between summations until the shift register has been updated with fresh samples so that the same incoming samples are used only once.

Figure 2(a) illustrates how such a correlator can be constructed. Now, if the desired total correlation length is N_c samples, and the parallel multiply and add operates on M samples at a time, the final accumulation needs to be repeated N_c/M times at every M^{th} sample period. For computing more than one correlation result in parallel, it is possible to utilize the single multiply and add block in a time-multiplexed fashion up to M times using different time-offsets during this period and then perform the final accumulation of the results separately. To be able to do this for K different time-offsets the length of the shift registers should have at least K additional samples compared to the number of samples that are multiplied and added in parallel.

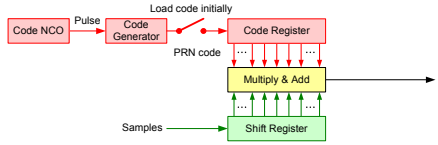


Figure 3. Block diagram of a matched filter implementation

If we would like to compute cross-correlation at M different time-offsets we would need to have $3M$ samples total in the shift registers. For example, $2M$ samples in the code replica shift register and M samples in the input data shift register. Also there should be a means of selecting the samples at different positions of the shift-registers for different time-offsets. Fortunately there is a better way of implementing the computation of M correlation results, which is shown in Figure 2(b). The data and code shift registers are both M samples long, and there is a code holding register of M samples that is loaded from the code shift register after every M^{th} sample period. While the code replica is held static, the data is shifted through the data shift register, and the parallel multiply and add circuit automatically computes the cross-correlation at M consecutive time-offsets spaced by the sample period. Every M^{th} output from the multiply and add circuit will correspond to the same time-offset and is already accumulated over M samples. If each of these results is now accumulated N_c/M times, we have M cross-correlation results covering M different consecutive time-offsets.

Comparing the modified serial-parallel correlator structure of Figure 2(b) to the matched filter displayed in Figure 3, we see some similarities, but can also note some important differences. The length of the MF is typically made equal to the replica code, and thus the code register can remain static after it is loaded with the correct code samples at the beginning of the operation. In the serial-parallel correlator, the code replica is continuously generated, and periodically loaded into the code register from the shift register. If the length of the MF is equal to or an integer multiple of the code length, then the MF computes a complete cross-correlation between the replica and the incoming data. If the MF length is shorter than the replica code, then the result is only a partial correlation, which is non-optimal from the GNSS operation point of view, as the correlation result distance between the maximum and the results at the other time-differences is not as large as it could be. This will degrade the performance of the receiver. Some solutions have been proposed to overcome this problem but the most usual way still is to make the MF length equal to the entire spreading code.

The traditional correlator based DS-SS receiver architecture as shown in Figure 4 is well suited for tracking mode operation in GNSS receivers, where only a

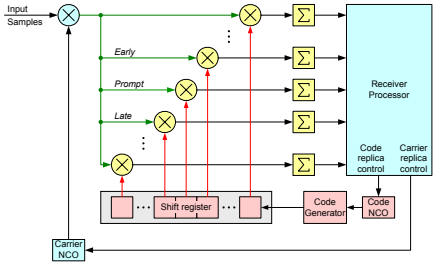


Figure 4. Traditional correlator based DS-SS receiver structure

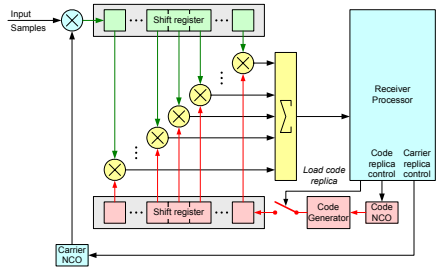


Figure 5. Traditional MF-based DS-SS receiver structure

few different code phase offsets are needed for the tracking loop. Often three are enough: one correlator positioned at the aligned code phase offset (prompt), and one earlier and one later code phase offset for the code tracking loop discriminator. For multipath detection and rejection, more correlators than just the minimum amount could be employed. However, the hardware overhead of adding more code phase offsets quickly becomes prohibitive, as a great deal of functionality would need to be replicated for each additional code phase offset to be computed. Thus, the traditional correlator is not very well suited for the GNSS acquisition process requiring rapid search over large code phase offset uncertainties.

The traditional MF-based receiver architecture as displayed in Figure 5 has long been used in the acquisition process in some DS-SS receivers [8]. The MF architecture is well suited to the acquisition process, as it can very rapidly produce correlation results for all possible code phase offset since it will produce a new correlation result for every new input sample. As input data shift register and the multiply and add arithmetic are two very large components of the MF implementation, it would be nice to time-multiplex these when trying to acquire more than one transmitted signal, which is commonly needed in GNSS applications. However, the MF length is typically very large and consequentially the bandwidth of the MF is relatively small. This makes it problematic to share the input shift register, if the center frequencies of the received signals are not very close to each other. E.g., in the GPS the duration of one C/A code period is 1 ms, and the usable bandwidth of the MF would be in the order of 500 Hz. As the Doppler frequencies for different satellites are spread over about 10 kHz, it would be unlikely that many satellites could be found from the same 500 Hz frequency range. The multiply and add computation block in the MF will need to multiply each corresponding replica code and input data sample and add all of these together in just one sample period. In a typical GPS receiver case, there would be 2046 samples to process every 0.5 us. It is difficult to economically perform this computation in time-multiplexed fashion for more than just a few channels. The MF is not very optimal for tracking mode operation as it has high overhead for producing just a few correlation results. The maximum output would also drift over time since the replica code is static, which makes the design of the tracking algorithm more challenging.

It is possible to implement the MF with length that is less than the full PRN code period. For short code lengths, the correlation properties of a partial code are far from optimum. The serial-parallel correlator structure shown in Figure 2(b) would solve the problems associated with the partial correlation by providing correlation over the full PRN code period or even more. This makes it more feasible to time-multiplex the multiply and add arithmetic block, but this does still suffer from the narrow bandwidth due to the number of samples accumulated. However, as the replica code is now updated periodically, it is possible to track the incoming signal much more easily, and maintain the code phase offset between the incoming signal and the replica code over extended periods of time. A DS-SS receiver architecture utilizing the serial-parallel correlator is illustrated in Figure 6.

The serial-parallel correlator has already many desirable properties for GNSS receiver architecture, but it is still not optimized for operation with more than a single input signal. The long coherent correlation makes the bandwidth very small, which causes a problem with the use of long

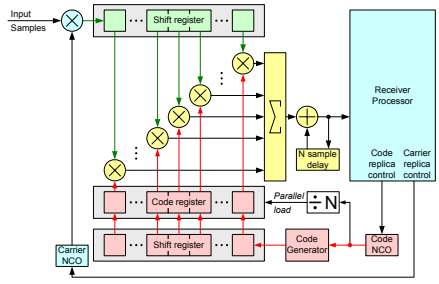


Figure 6. Serial-Parallel correlator based DS-SS receiver structure

correlation in reception of multiple GNSS signals whose carrier frequencies are separated due to the different Doppler frequencies. It is possible to divide the integration into two pieces while keeping it mathematically equivalent. Also, the mixing with the carrier replica code can be split into two parts, and the second part can be moved after the first integration, provided that the frequency of the second component of the carrier replica is low enough to be considered essentially constant over each of the shorter integration periods of the first part of the integration. Splitting the carrier replica mixing function allows sharing the largest part of serial-parallel correlation hardware, as the bandwidth of the correlator itself can pass all received signals. The second stage mixing then allows continuing the integration over the full code period or longer for each of the received signals separately. The first carrier replica should be the same for all incoming signals, and the second carrier replica needs to be adjusted for each signal separately. In the GNSS application, the first carrier replica part is used to cancel out any common frequency offsets such as residual intermediate frequency (IF), reference oscillator error etc., such that the frequency range of the incoming signals is centered on zero frequency after the first mixer. The second part of the carrier replica is then used to track the individual GNSS signals.

Now that we have found a way to overcome the problems of sharing the correlation hardware amongst channels, we can construct the correlator hardware for sharing much of the hardware while making it possible to operate on more than one GNSS signal. If we add shift and code registers and replica code generators for each GNSS signal to be received into the structure shown in Figure 2(b), and time-multiplex the multiply and add computation so that the data in the sample shift register is multiplied and added with all the code replicas within one input sample period, we can cross-correlate the input with N_P code replicas at M different code phase offsets (time-offsets). When we multiply the correlation results with the second part of the carrier replica, we need to take into account that each N_P consecutive output corresponds to a different incoming GNSS signal, and the mixing also needs to be time-multiplexed accordingly. We can also use the same idea to increase the code phase offset coverage by adding more code registers that get their input from the previous code registers associated with the same code replica generator. By using N_R code registers we can now cover $M{\cdot}N_R$ different code phase offsets for each replica code with bandwidth which is equal to covering only M offsets. This group correlator architecture allows us to efficiently compute M-samples long cross-correlations between the input samples and $N_P{\cdot}N_R$ code replica segments. The same principle can be extended to also utilize more than a single input sample SR, which is advantageous for GNSS receivers operating in both acquisition and tracking modes simultaneously. The simplified block diagram that shows the basic architecture of the group correlator is displayed in Figure 7.

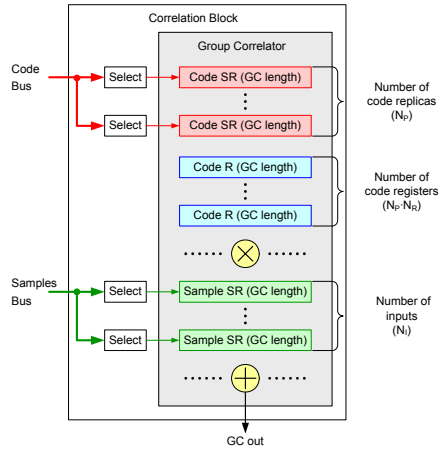


Figure 7. Conceptual block diagram of the group correlator

Hardware Implementation

While it is possible to use the group correlator architecture to always compute all possible cross-correlation combinations between the input sample SR(s) and the replica code register(s), a big advantage in implementation can be achieved by carefully arranging the connections of the building blocks. Considering the application of GNSS receivers, we note that the receiver needs to operate in several different modes, and the best input signal sample rate for the modes will not be the same. As discussed before, the group correlator architecture can cope with more than a single input sample rate for one GC block. This will require, however, that each code replica shift and code register is associated with a certain input sample rate, as the code SRs and the sample registers need to be updated at the same rate to maintain the time synchronization. This would mean that some of the code register loads would occur in different times with respect to the other registers, and it might become a challenging task to schedule the computation of the results between input samples. To allow discrete time operation, the clock rate of the GC should be equal to an integer multiple of the least common multiple of the input sample rates selected such that there will be enough clock cycles to compute all desired cross-correlations within the corresponding input samples. If the other sample rates are integer multiples of the lowest sample rate, the scheduling can be made much easier. To minimize the required clock frequency, the GC should produce outputs every clock cycle, and this will limit the usable choices even further. Fortunately, it is still possible to find suitable sample rates and number of code registers (N_R) for use with GNSS receivers. If the sample frequencies associated with the input sample SRs and their associated code replica

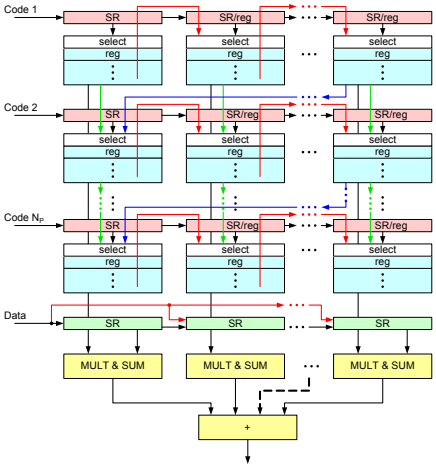


Figure 8. Generic programmable GC with single selectable input frequency

registers are $F_{S,i}$, then the number of code registers to be used for that sample frequency ($N_{R,i}$) should be selected so that it is the least common multiple (LCM) of the sample rates divided by the corresponding sample rate or an integer multiple of it, i.e.

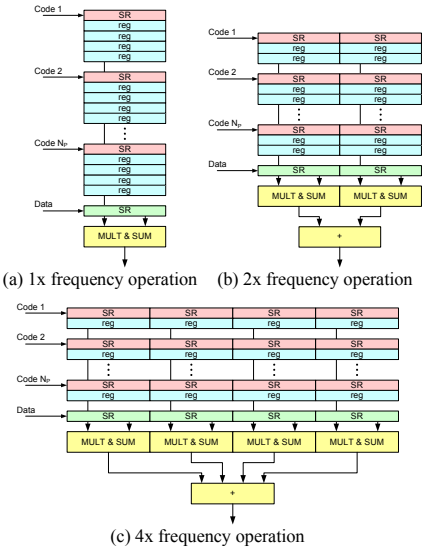
$$N_{R,i} = N_{mult} \cdot \underset{n=0}{\overset{N_P}{\mathrm{LCM}}}(F_{S,n}) / F_{S,i}$$

The required clock frequency can then be computed by

$$F_{CLK} = N_P \cdot N_{mult} \cdot N_{R,i} \cdot F_{S,i}$$

for any i.

The biggest advantage of the group correlator architecture can be achieved by realizing that it is possible to make the number of code replica registers and the sample rate assignment programmable and still produce correlation results at a constant rate. This will greatly simplify the design of the blocks that follow in the signal processing path. For the sake of simplicity we shall first describe the configuration of the GC which utilizes only a single input sample rate at a time. However, we will make the configuration in such a way that it would be possible to combine several blocks operating at different sample rates, and still produce the output samples at the same rate. This also simplifies the addition of more input sample rates later. Figure 8 shows the generic GC structure. The replica code SRs are shown in red in the figure, the code registers in blue, the input sample SRs in green, and the arithmetic units in yellow. There are basically two modes of operation for this GC structure, called narrow and wide. In the wide operation mode the code SRs that are associated with the same PRN replica code, i.e. the same group, are chained so that the reference code flows from one end of the chain all the way from left



(a) 1x frequency operation (b) 2x frequency operation

(c) 4x frequency operation

Figure 9. Different configurations of the programmable GC for 1x, 2x, and 4x frequency operation

to right. In the narrow mode only the first code SR is used, and the data from the last (lowest) code register in a group is loaded into the first (topmost) code register at the right side of it, as illustrated by the red arrows in the figure. In both modes it is possible to chain the different groups by bypassing some of the code SRs and instead taking the code register data from the group above it. The chaining data flows are shown by the blue (narrow) and green (wide) arrows. The narrow versus wide configuration can be selected separately for each horizontal pair of code registers and the pairing can be nested.

As an example, Figure 9 shows three configurations for a GC block with three possible sample rates $F_{S,1}$, $F_{S,2} = 2F_{S,1}$, and $F_{S,3} = 4F_{S,1}$. There are four code registers per group when operating at the lowest sample rate (a), two code registers when operating at the double sample rate (b), and only one code register when using the quadruple sample rate (c). However, the total number of samples in the code registers always remains the same. The correlation length in time also remains identical in all of the modes. This makes the GC very powerful architecture for use in multiple operating modes. We can note, however, that the number of samples in the code SRs increases with increasing sample rate, which increases the hardware overhead for the lower sample rate configurations. It is possible to optimize the HW size by making the correlation length smaller in the higher sample rate operation. This reduces the code offset range, which is fortunately not too detrimental to the receiver

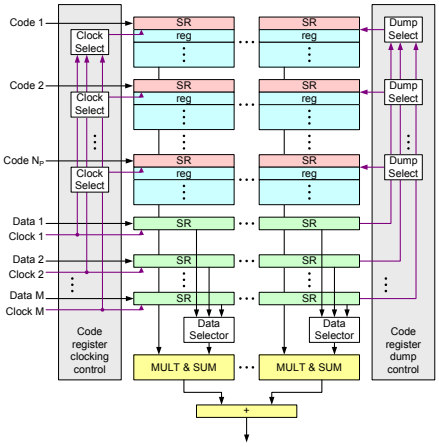


Figure 10. Block diagram of a programmable GC block with multiple input sample SRs.

performance, as the higher sample rates tend to be used in tracking mode operation, where the required code phase offset coverage is not as large as in the acquisition modes. In the example here, it would be a good optimization to leave half of the code registers unused in the highest sample rate mode.

The GC structure discussed above had only one input sample rate at a time. It is also possible to have more than one sample SR with a programmable sample rate selection for each of the reference code inputs while retaining similar code register configurability. A GC structure with several input sample SRs is illustrated in Figure 10. The figure does not show the replica code configuration part, to maintain clarity, but all of the connections shown in Figure 8 can also be made here. Some additional logic is needed to make the input selection possible. There needs to be selectable control to dump the code SRs to the code registers and to shift the code register data accordingly to the code registers below. Each group in the GC needs to be associated with one of the input sample frequencies, and the code SR should be clocked with the same clock as the corresponding input sample SR. Furthermore, the dumping of the data needs to be synchronized with the filling of fresh data into the code and input sample SRs, which should have equal lengths. For example, if the GC is like the one shown in Figure 9 but with separate input sample SRs for all of the three sample rates, the corresponding input sample SR lengths should be 1, 2, and 4 times the length of the single code register.

When combining all of the possible features presented above, the group correlator can become a very complex device. Fortunately, when the application parameters are known, the GC can be tailored to a given application and

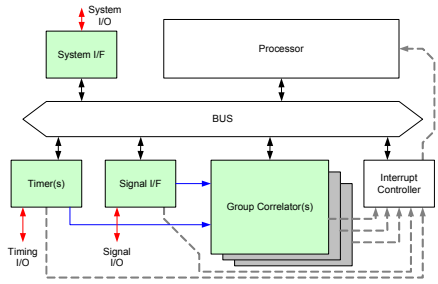


Figure 11. Block diagram of the designed receiver baseband section

many simplifications can be made while retaining much of the programmable configuration possibilities. In many cases it is possible to define a small number of operational modes for which the internal connections in the GC can be tailored. For example, in the case of a GPS receiver, only two modes might be needed: acquisition and tracking. In the acquisition mode, the sample frequency would be the lowest possible and the code phase coverage the largest. In the tracking mode the sample rate would be higher to improve the tracking accuracy, but the code phase coverage requirement would not be very large. Adding Galileo in the receiver requirements this will mean additional modes are needed. The tracking can be implemented with the same sample frequency, but the Galileo signal will have a wider bandwidth requiring a higher sample rate in the acquisition mode. The two GNSS systems should probably utilize different sample rates for acquisition to maximize the search range for GPS. For a combined GPS+Galileo receiver we might thus have three operating modes defined with a particular GC configuration associated with each:

1. GPS signal acquisition
2. Galileo signal acquisition
3. Signal tracking

Each replica code, or channel, can operate in different mode, which makes the receiver operation still very flexible, even if only three possible configurations are defined for each GC group.

GPS+GALILEO RECEIVER IMPLEMENTATION

We have designed a GNSS receiver baseband block based on the GC. A SystemC [9] implementation of it was made and tested for functionality. The GPS+Galileo receiver design is described in this section. The receiver baseband block is illustrated in Figure 11, and contains a processor subsystem along with the digital logic to implement the receiver baseband hardware. We will focus on the GNSS specific hardware, as the rest can be built from readily available components.

The group correlator based GNSS receiver signal processing path shown in Figure 12 contains several building blocks beside the GC itself. There are several important design parameters at the receiver top level:

1. Sample rates for the input signal.
2. Number of independent spreading codes to process. This determines the number of code generators needed.
3. Number of code bins for each code. This affects the size of the GC core.
4. Total number of code bins in the receiver. This affects the number of GC blocks needed.
5. Number of desired frequency bins for each separate code. This determines the size of the mixer bank.

The total number of correlator bins (often referred as the number of correlators) can be computed as a product of the number of independent codes, the number of code bins for each code, and the number of frequency bins. In the following discussion we will refer to the number of the independent codes as the number of channels in the receiver.

IF Removal Block

There is one IF removal block which is common to all of the receiver channels. This is possible since the bandwidth of the GC is relatively large compared to the possible frequency range of the incoming signals. The final Doppler removal is done later in the mixer blocks. The IF removal block removes a common frequency offset and sets the sample rates for the further processing blocks. There are three different sampling rates available for different operation modes of the channels. The input signal is mixed with a locally generated frequency in the IF removal block. Next the signal is down-sampled into the final sample rates, and re-quantized to optimize the HW requirements of the GC blocks.

GC Block

The GC Block is the basic building block of the GNSS receiver. Each GC Block contains code and carrier replica generators, a GC Core Block, and the Integrator Block, which performs coherent and non-coherent integration as well as the detection acceleration for each frequency bin. The code replica generator can be controlled by the SW and it can generate a separate code replica signal in a time-multiplexed manner independently for each channel in the GC Core Block. The receiver supports any spreading code up to a length of 4096 and it can also generate the BOC(n,n) sub-carrier to support Galileo signal reception. The carrier replica generator supports the generation of evenly spaced frequencies centered on a per-channel defined residual frequency estimate. There are separate Integrator Blocks for each of the frequency bins operating in parallel. Furthermore, the coherent and non-coherent integration use separate memory blocks, so that the processor can access the results in the non-coherent memory even while the results of the coherent integration are being continuously accumulated. The magnitude of the signal in the complex I/Q representation is computed by the absolute value unit, whose function approximates

$$\mathrm{ABS} = \sqrt{I^2 + Q^2}\ .$$

The absolute value computation can also be bypassed on a per-channel basis for the tracking mode, where the complex valued results are needed. The detection acceleration block will greatly assist the SW as the number of correlation results from the whole receiver that needs to be processed during the acquisition of the satellite signals is very large.

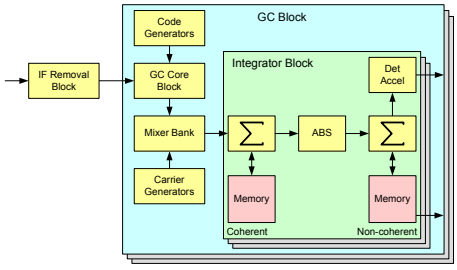


Figure 12. Structure of a GNSS receiver baseband block based on the group correlator

GC Core Block

The GC Core Block is the correlation engine in the receiver, and it contains an implementation of the group correlator as described in the previous sections. It contains the code replica shift and dump registers, the data shift registers, and the code and data multiplication and summation logic. The GC Core Block has several parameters and the most important of them are:

1. Number of independent replica code inputs for the GC
2. Correlation length for each group
3. Number of clock frequencies supported

The total number of channels in the receiver can also be adjusted by selecting how many GC Blocks are used in one particular implementation.

We next discuss the parameter selection process for the receiver. The selection of the sampling frequencies for the receiver should be done first. Fortunately it is a relatively straightforward task. We selected three different sampling frequencies – 2 MHz, 4 MHz, and 8 MHz – which are integer multiples of the lowest one, to be used as required by efficient implementation of the GC structure. Another key parameter for the receiver design is the total number of code bins for each frequency bin, which needs to be defined based on the desired receiver performance in the acquisition mode. We wanted the receiver to be able to search the full code uncertainty of four GPS satellites simultaneously. This requires us to implement at least 8184 code bins for each frequency bin. With this amount

Table 2. Receiver configurations with varying number of channels per GC Block

Channels per block	Clock frequency (MHz)	GC Blocks	Channels (total)	Code bins (total)
1	**8**	**32**	**32**	**8192**
2	**16**	**16**	**32**	**8192**
3	24	11	33	8448
4	**32**	**8**	**32**	**8192**
6	48	6	36	9216
8	**64**	**4**	**32**	**8192**
10	80	4	40	10240
12	96	3	36	9216
16	**128**	**2**	**32**	**8192**
32	**256**	**1**	**32**	**8192**

of code bins we can also search one half of a full code uncertainty of one Galileo satellite. There are some additional constraints that need to be taken into account when choosing the receiver configuration. In the tracking mode, we would like to track at least 24 satellites as to fully utilize both GNSS systems. The length of the GC should be selected in such a way that the total carrier uncertainty and Doppler range fits into the GC bandwidth with little signal-to-noise ratio (SNR) degradation. We only need to compute the GC length for the lowest sample rate, as the other cases are directly scaled accordingly. From system analysis, it has been determined that a GC bandwidth of 32 kHz is a good choice for this parameter. Rounded up to a convenient integer value this will give 64 as the GC length. As the sample rates relate to each other with ratios – 1:2:4 – the number of code registers in the 2 MHz mode needs to be a multiple of four as explained in the previous section. Thus the number of code bins per channel at the 2 MHz sample rate will be 256.

Keeping in mind the desired total number of code bins and the number of code bins per channel, we note that the minimum number of channels that will satisfy this condition is 32, which nicely satisfies our requirement for the number of channels. The group correlator architecture allows the number of channels per block to be varied easily. Table 2 shows how the other receiver parameters need to be changed with a selection of choices for the number of channels per block to satisfy the requirement for the total number of code bins. The choices that result in the minimum hardware are highlighted in green. When considering the optimum choice of the parameters it is important to note that a larger number of blocks increase the controllability of the hardware and might lead to lower power consumption, as blocks that are not currently needed might be turned off. However, the hardware overhead is the smallest when the number of blocks is also the smallest, because the amount of HW that is independent of the number of channels is also minimized. The integration memory blocks that are used in each GC block contain more memory locations when the number of

Table 3. GC-related parameters for the designed receiver

Parameter	Value
Defining Parameters	
Operating frequency	32 MHz
GC length	64
GC Multiplexing factor	4
Number of channels per GC Block	4
Number of GC Blocks	8
Derived Parameters	
Number of channels	32
Code bins per channel	256 (128)
Code bins per GC Block	1024
Total number of code bins	8192

blocks is lower, and this also results in improved silicon area efficiency. Another constraint that needs to be considered is the highest available clock frequency. As can be seen in Table 2, it can become quite high when the number of channels per block is increased. For our receiver design we selected a clock frequency of about 32 MHz, which resulted in a convenient choice for the number of channels per block. It also offered us good flexibility for design partitioning. The selected and derived GC-related parameters for the receiver implementation are summarized in Table 3.

Having now determined the GC block parameters we can still change the number of GC blocks to alter the receiver configuration. Adding blocks increases the HW size, but also makes the receiver more powerful. Removing blocks can be done if the receiver HW is too big for a particular application. This great configurability of the basic design shows the advantage of the GC-based GNSS design. It would thus be possible to make a single configurable IP block that could be used in many different applications with different performance and size requirements.

Table 4 shows how the receiver search range and number of tracking channels change when the number of GC Blocks is changed. The parameters for the GC Blocks are

Table 4. Search capability and number of tracking channels vs. GC Blocks is changed

GC blocks	Code bins	GPS range	Galileo range	Satellites tracked
1	1024	1	0.06	4
2	2048	1	0.13	8
4	4096	2	0.25	16
6	6144	3	0.38	24
8	8192	4	0.50	32
10	10240	5	0.63	40
12	12288	6	0.75	48
14	14336	7	0.88	56
16	16384	8	1.00	64
18	18432	9	1.13	72

specified in Table 3. The configurations that achieve the minimum requirements of a single full GPS code search range are shown in yellow. The ultimate configurations that can search one full Galileo code are show in blue, and the configurations which can search either four GPS satellites or half of a Galileo code are shown in green.

EVALUATION AND RESULTS

For evaluating the operation and performance of the receiver a SystemC-based system simulator was implemented. Its architecture is shown in Figure 13. The RF front end providing the digitized IF data stream for the digital baseband can be simulated or recorded data can be read from a file.

The system simulator for the designed receiver was used to perform some functionality and performance testing using a SW-based GPS signal generator. Also some testing with recorded signals from the real GPS satellites has been performed. The designed receiver was found to be functional in all aspects. The operation with the Galileo signals has not been evaluated more than by generating a simple BOC(1,1)-modulated signal and verifying that it can be detected.

Performance

The acquisition and tracking sensitivities have been tested in several situations. The receiver was configured as described in the previous section. When accurate time assistance (as in the IS-916 minimum performance test) is available the acquisition sensitivity has been found to be about 18 dB-Hz with a 2 second total search time. With coarse time assistance (as in the 3GPP minimum performance tests) the acquisition sensitivity was about 21 dB-Hz. The designed receiver can track the signal when it stays above 15 dB-Hz.

Figure 15 shows the results from simulation with accurate time assistance in the acquisition mode (2 MHz, 20 ms coherent integration, up to 2 s search time) while Figure 14 illustrates the simulated operation in signal verification mode with accurate time unknown (8 MHz, 3 ms coherent integration, up to 100 ms search time).

CONCLUSIONS

The proposed correlation device, group correlator, is a mix between the matched filter and the correlator. In the simplest form, it provides a means of computing several correlation operations in parallel just as the MF does, but it can compute the correlation over the full code period with the aid of an integrator block placed after the GC, which ensures that the correct code position has the maximum output value, even if the correlation length in the GC is less than the full code period. With a second stage carrier mixer placed after the GC and before the final integration, the proposed architecture overcomes the signal bandwidth issue related to MF-based implementations.

The group correlator design is superior compared to correlator based, matched filter-based, as well as Fast Fourier Transform (FFT)- based GNSS receiver architectures. The main advantage of the GC architecture is minimal hardware complexity compared to traditional approaches, and the possibility of software configuration of the receiver. For the hardware implementation, a single building block can be used, which simplifies the design, eases the HW configurability, improves receiver flexibility and usability, and reduces the HW resources needed.

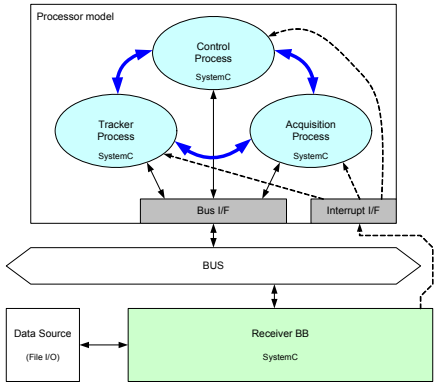


Figure 13. GNSS system simulator structure

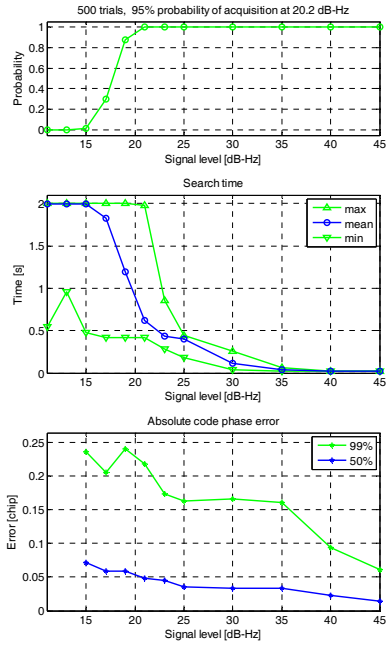


Figure 15. Simulation results in weak signal acquisition mode with accurate time assistance

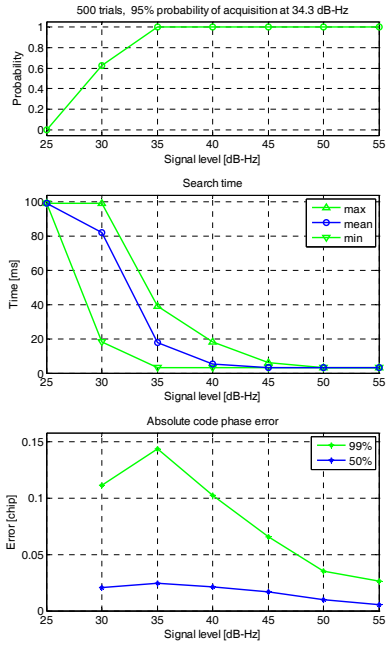


Figure 14. Simulation results in strong signal verification mode without accurate time assistance

REFERENCES

[1] 3GPP TS 25.171, *Requirements for support of Assisted Global Positioning System (A-GPS); Frequency Division Duplex (FDD)*, Available: http://www.3gpp.org/ftp/Specs/html-info/25171.htm

[2] 3GPP2 C.S0036-0 v1.0, *Recommended Minimum Performance Specification for C.S0022-0 Spread Spectrum Mobile Stations*, Available: http://www.3gpp2.org/Public_html/specs/C.S0036-0_v1.0.pdf

[3] SiRF Technology, Inc., *SiRFstarIII GSC3LTi and GSC3LTif*, Product Overview, November 2006.

[4] Global Locate, Inc., *MantaRay™ System-In-Package A-GPS Solution*, Available: http://www.globallocate.com/SEMICONDUCTORS/SEMI_MANTARAY_Frameset.htm

[5] Jack K. Holmes, *Coherent Spread Spectrum Systems*, reprint ed., Robert E. Krieger Publication Company, 1990.

[6] Marvin K. Simon, Jim K. Omura, Robert A. Scholtz, Barry K. Levitt, *Spread Spectrum Communications Handbook*, rev.ed., McGraw-Hill, 1994, Part 4, Ch. 1.

[7] Stephen S. Rappaport, Donald M. Grieco, "Spread-Spectrum Signal Acquisition: Methods and Technology," *IEEE Communication Magazine*, vol. 22, No. 6, pp. 6-21, June 1984.

[8] Ville Eerola, "Rapid Parallel GPS Signal Acquisition", *Proceeding of the ION GPS Conference*, Salt Lake City, September 2000.

[9] Open SystemC Initiative (OSCI), *Functional Specification for SystemC 2.0*, October 2001, Available: http://www.systemc.org/.

PUBLICATION 5

Ville Eerola and Jari Nurmi, "High-level parameterizable area estimation modeling for ASIC designs," *Integration, the VLSI Journal*, vol. 47, no. 4, pp. 461–475, Sep. 2014.
DOI: 10.1016/j.vlsi.2014.01.002

INTEGRATION, the VLSI journal 47 (2014) 461–475

Contents lists available at ScienceDirect

INTEGRATION, the VLSI journal

journal homepage: www.elsevier.com/locate/vlsi

High-level parameterizable area estimation modeling for ASIC designs

Ville Eerola *, Jari Nurmi

Tampere University of Technology, Department of Electronics and Communications Engineering, P.O. BOX 553, FIN-33101 Tampere, Finland

ARTICLE INFO

Article history:
Received 23 May 2013
Received in revised form
4 January 2014
Accepted 10 January 2014
Available online 23 January 2014

Keywords:
Gate-count estimation
Architecture exploration
VLSI circuits
Integrated circuit modeling
System-on-chip

ABSTRACT

Architectural design space exploration and early area budgeting for ASIC and IP block development require accurate high level gate count estimation methods without requiring the hardware being fully specified. The proposed method uses hierarchical and parameterizable models requiring minimal amount of information about the implementation technology to meet this goal. The modeling process flow is to: (1) create a block diagram of the design, (2) create a model for each block, and (3) sum up estimates of all sub-blocks by supplying the correct parameters to each sub-model. We discuss the model creation for a few parameterized library blocks as well as three communication blocks and a processor core from real IC projects ranging from 22 to 250 kgates. The average relative estimation error of the proposed method for the library blocks is 3.2% and for the real world examples 4.0%. The best application of this method is early in the design phase when different implementation architectures are compared.

1. Introduction

As both the complexity of Application Specific Integrated Circuit (ASIC) designs and demands for rapid time to market keep increasing, the importance of good estimates for the design complexity is growing. The gate count of a design affects its cost in several ways. The direct influence comes from the cost of the silicon area. The gate count also has a close relation to the device power consumption via the total capacitance accountable to the gates. The testability and test time of the device are affected by the complexity. Finally, the device yield is decreased with increasing area. In order to predict these, the device gate count or an other complexity measure is needed. In many cases the financial feasibility of an ASIC development project also depends on these estimates. It is very desirable that an accurate estimate of the device complexity could be made as early in the development as possible. Often, the first estimates would be needed even before any detailed system design has been made.

Early gate count estimation is desirable also because design optimization is more efficient at the early phases than later in the project. This would enable efficient hardware optimization at the architectural and system design levels. In many cases, the design contains several parameters which could be changed to alter its performance. However, at the same time, these alterations also have an effect on the silicon area required for the implementation. Having a gate count estimation model, which would follow the parameter changes, would allow more efficient high-level design space exploration and accurate planning of the design project.

The gate count estimation method presented in this paper has been developed to meet these demands. It can cope with rough and sketchy design specification since it does not depend on a formal and complete description of the device. It also allows having parameters to be included in the block descriptions, that will be taken into account when the gate count estimates are computed. Furthermore, the methodology requires a minimal amount of information about the implementation technology and keeps it separated from the design so that the implementation technology can easily be changed if needed. In the proposed methodology the gate count $\mathcal{G}$ for a block $\mathcal{B}$ is modeled using bottom-up parameterizable models.

The strength and differentiating factor in our methodology is to be free of requirements for strict design methodology or a formal description of the design to be estimated. This freedom allows gate count estimation to be done with incomplete designs.

The remaining of this paper is organized as follows: In Section 2 we will review some of the earlier work on the hardware area estimation field. In Section 3 we present an overview of the proposed gate count estimation methodology at a high level, describing the main steps involved. Next, in Section 4 we will explain the details involved in the gate count estimation model creation and explore some of the basic building blocks. In addition we will show the modeling process of some more complex building blocks: a multi-port register file, a CORDIC phase rotator and a complex mixer. In Section 5 we will go through four example cases from real world designs and present the results from the example cases and compare our estimates with actual gate counts. Furthermore, we will give some notes and summarizing

* Corresponding author.
E-mail addresses: Ville.Eerola@tut.fi (V. Eerola), jari.nurmi@tut.fi (J. Nurmi).

0167-9260/$ - see front matter © 2014 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.vlsi.2014.01.002

comments. Finally, Section 6 concludes the work and shortly discusses potential improvements to the methodology.

2. Related work

Several approaches to the design complexity estimation in area, gate count, or Field Programmable Gate Array (FPGA) resource usage have been proposed in the literature. The earliest approach for estimating the switch count of implementing Boolean functions was done by Shannon [1], who proposed that the *upper limit* of the complexity of Boolean functions is proportional to the exponential of the number of inputs. This was later applied to the gate count estimation for implementations of Boolean functions by Muller [2]. However, it is easy to see that these estimates cannot be applied to realistically complex circuits, due to the exponential growth of the estimate.

Kellerman [3] presented a formula for the computation of the area estimate and proposed that the area of a function is only depending on the number of conditions which must be differentiated by a one or a zero output. Other researchers, e.g. Cook and Flynn [4] and Pippinger [5], studied the relationship between the area complexity of Boolean functions and the entropy ($\mathcal{H}$). Cheng and Agrawal [6] expanded the entropy measure based estimation method to multi-output Boolean functions.

Nemani and Najm [7] pointed out problems in earlier work, which had been based on *randomly generated* Boolean functions with a small number of inputs. They showed how earlier estimates would greatly overestimate the gate count of real circuits and proposed that typical circuits are far from random in their structure. In their work, the authors developed a new *linear measure* of Boolean functions, which is dependent on the complexity of the on and off-sets of the function. Using that measure gave more realistic results when using typical circuits.

Büyükşahin and Najm [8] developed the estimation method further and used a Boolean network representation of the circuit instead of RTL level description to enable estimation at a higher level of abstraction. Still, use of these kinds of methods requires the knowledge of the accurate Boolean functions of the logic, which may not be available for all parts of the design at early phases of the design. For example, in many cases the accurate function for control logic is not known at the architecture design phase, even if the data processing would be well defined.

For the estimation of FPGA design complexity, a methodology was described by Enzler et al. [9]. Their methodology is based on capturing the features of the design on a data-flow graph (DFG) or block diagram level to a characterization vector consisting of the number and word lengths of adders, multipliers, etc. in addition to some other characteristics of each block. This is then mapped into the FPGA area and timing estimates. There is some similarity to our methodology, but our approach allows more freedom in the characterization as it is not tied to a fixed vector format containing counts of only a few characteristic functions such as adders, multipliers, and multiplexers. Our method allows freedom of selecting the functions to which the design is mapped to. This allows an optimal set of functions to be chosen for different classes of designs. It is also possible to increase the estimation accuracy by incrementally adding more basic blocks to be used in the estimation process over time.

It also seems apparent that many of the earlier efforts on the gate count or area estimation field seem to concentrate on integrating the algorithms into CAD tools for building more comprehensive frameworks like in [10].

Table 1 summarizes the above references with regard to the used design description (input) and resulting cost estimate (area) as well as shows the reported accuracy of the method, if available. Additionally, [11] provides some additional references and a good survey of the hardware characteristics estimation techniques related to hardware/software partitioning. In summary, most of the earlier work have been concentrated to finding a formula to map a small, combinatorial, accurately known logic function to a number related to the area of the silicon implementation. In contrast, our work aims to provide a more high-level tool for the area estimation, which would work on realistically complex designs early in the design process.

3. Gate count estimation methodology

We will now describe the estimation process at a high level. In the next section we will go into more detail of the process. The estimation methodology described in this paper is based on a bottom-up modeling approach. The models are built from parts at three levels:

1. Primitive library
2. Basic block library
3. Design level models

The lowest level of the model consists of a set of predefined primitives, which correspond directly to standard-cell gates of the implementation technology. The next level there is a pre-defined, but extensible, set of basic building blocks, such as adders, multipliers, and registers. The highest level consists of models that are specific to the design under estimation. The difference between the basic blocks and the design level models is that the basic blocks are re-usable and parameterized in such a way that they can be used in modeling of many different designs. On the other

Table 1
Area estimation references comparison.

Reference	Input	Cost (output)	Average accuracy	Notes
[1]	Boolean functions	Switch count	n/a	Theoretical work
[2]	Boolean functions	Gate count	n/a	Theoretical work
[3]	Boolean functions	Diode count	15% (random) / 33% (real)	
[4]	Boolean functions	Diode count	n/a	Theoretical work
[5]	Boolean functions	Gate count	n/a	Theoretical work
[6]	Boolean functions	Gate count	30%	
[7]	Boolean equations	Gate count	22%	Based on on/off sets of function
[8]	Boolean network	Gate count	24%	
[9]	Block diagram/DFG	FPGA cells	12%	Intermediate vector representation
[10]	VHDL	FPGA cells	3.5%	Intermediate CDFG representation
This work	Block diagram/DFG	Gate count	4.0%	Parameterizable/extendable models

hand, design level models are created for the design that is being estimated during the modeling process and are unique for that design.

The set of the required primitives is limited, and thus only a limited amount of information is required about the implementation technology. The required data consists of gate size information on a few key gate types in the library that will be used in the synthesis of the design. In many cases, default gate count values could be used for the primitives with only a minor loss of accuracy. This is particularly useful when developing IP blocks without the knowledge of the final implementation technology.

The algorithm for the model creation process is shown in Algorithm 1.

Algorithm 1. The model creation.

```
1: procedure CreateModel(block)
2:   Divide to sub-blocks as needed
3:   foreach sub-blocks do
4:     CreateModel(sub-block)
5:   end for
6:   Define parameters as desired
7:   Create a data-flow graph / block diagram
8:   Map remainder to basic blocks or primitives
9: end procedure
```

The process starts by creating a data-flow graph or a block diagram of the design to be estimated. In many cases, this is already available. In some cases, e.g. if a pre-existing design implemented in FPGA technology needs to be transferred to an ASIC technology, there may exist more detailed documentation and RTL-level (VHDL, Verilog, etc.) descriptions, which would allow the estimation to yield more accurate results.

When the high level description of each sub-block has been done, parameters can be defined for each block. This would allow easy exploration of the design by changing some of its parameters. The parameters should be taken into account when modeling the blocks, but their effect does not necessarily need to be taken into account in full accuracy. Consider for example the number of registers in a register file. Changing this parameter also affects the address size, but the effect is very small ($\mathcal{G} \propto \log_2(N_{\text{regs}})$) compared with the fact that the register file size is (almost) directly proportional to the number of registers ($\mathcal{G} \propto N_{\text{regs}}$).

Next, each sub-block is mapped to a predefined set of basic building blocks, and suitable parameter values for the used library blocks are calculated based on the design parameters. E.g. using the *register file* block, the number of registers and their word length as well as the number of input and output ports need to be known, and their values should be used in the mapping. If similar structures are used in many places of the design, new re-usable building blocks could be created if needed. By expanding the building block library, the effort for estimating new blocks will become smaller. Typically, designers tend to use similar structures in many of their designs and building their own estimation library will decrease the effort and improve the accuracy of the estimation. When creating new elements to the building block library, it is necessary to pay close attention to: (a) how they are parameterized, so that they will be maximally re-usable; (b) the accuracy of the models, since the modeling accuracy depends very much on the accuracy of the building block models.

The algorithm for the model evaluation is listed in Algorithm 2. This algorithm also shows the generic model structure for a block.

Algorithm 2. The model evaluation.

```
1: function EvalModel(block, params…)
2:   count ← 0
3:   foreach B ← instantiated sub−blocks do
4:     count ← count + EvalModel(B, params…
5:   end for
6:   foreach B ← instantiated basic−blocks do
7:     count ← count + EvalModel(B, params…)
8:   end for
9:   count ← count + ∑ (gate count of primitives)
10:  return count
11: end function
```

The actual modeling can be implemented in many ways. One good way is to use some interpreted computer language or program like Matlab [12] or GNU Octave [13]. This allows the easy implementation of the parameterization and evaluation of the gate counts with multiple parameter values. The modeling of the examples provided in this paper was performed by manually writing a set of Matlab functions to calculate the gate counts, but the process could easily be integrated to a block diagramming tool to combine the design documentation and the estimation model creation into a single step.

One important aspect of the presented methodology is that the routing area is not taken into account. The result of the estimation is a gate count value. If a physical area estimation is needed the routing area must be estimated separately. In the routing area estimation, the gate count and some estimates of the gate fanouts are needed. In some cases, one could also use a generic gate count per area figure available from the silicon vendors for this purpose with good accuracy.

4. Parameterized model creation

We will now explore the model creation process in more detail, and explain how it is used. In this section we also give examples how some of the basic blocks in the library can be created, as the process is essentially the same as for larger designs. The model creation process starts from the primitives, which are used to build the models of the basic building blocks. The basic building blocks are then used to build the models of the blocks of the design. The block level models can also use the primitives directly if necessary.

4.1. Primitives

The modeling methodology was initially developed for estimating the gate counts of data processing IP-blocks in Digital Signal Processing (DSP) and communication applications. It was then noticed that the majority of the area is taken by *registers*, *full-adders*, and *multiplexers*. Multiplication, which is also common, is implemented by an array of full-adders and *AND*-gates. Most DSP processing algorithms can be implemented with just these cells together with a small amount of additional controlling logic. For any reasonably sized block, counting just these few basic cells will account for the majority of the gates in the whole block.

In the proposed gate count estimation methodology, the list of primitives used is predefined. Table 2 shows the set along with their sizes as gates in a typical CMOS process library. As the gate count estimate is ultimately based on the sizes of the primitives, it would be important for its accuracy to get the sizes of the primitives right. Before starting the model creation, the primitives should be checked against the implementation library if one is available. It is also important to consider the reset strategy selected for the design as this has an impact to the size of the register bits. For the designs in this paper we use synchronous

Table 2
Primitives for estimation.

Primitive	Size	Notes
2-Input NAND (NAND2)	1 gate	1-gate area definition
Inverter (INV)	1 gate	Slightly larger than a minimum inverter
Register bit (FF)	5 gates	Synchronous reset
Full-adder (FA)	5 gates	
2-Input MUX (MUX2)	3 gates	
Half-adder (HA)	3 gates	
2-Input XOR (XOR2)	3 gates	

Table 3
Summary of adder-like building blocks.

Function	Size
Adder	$\mathcal{G}(\text{add}(N)) = N \times \mathcal{G}(\text{FA})$
Subtractor	$\mathcal{G}(\text{sub}(N)) = N \times (\mathcal{G}(\text{FA}) + \mathcal{G}(\text{INV}))$
Increment by 1	$\mathcal{G}(\text{inc}(N)) = N \times \mathcal{G}(\text{HA})$
Decrement by 1	$\mathcal{G}(\text{dec}(N)) = N \times \mathcal{G}(\text{HA})$
Absolute value	$\mathcal{G}(\text{abs}(N)) = N \times (\mathcal{G}(\text{XOR2}) + \mathcal{G}(\text{HA}))$
ADD/SUB	$\mathcal{G}(\text{add\_sub}(N)) = N \times (\mathcal{G}(\text{XOR2}) + \mathcal{G}(\text{FA}))$

reset, which requires an AND-gate at the register cell input. This has been taken into account in the register bit primitive in Table 2. A good method would be to synthesize a predefined simple block, which would map well to the set of the primitives and extract average primitive sizes from the resulting netlist. This would allow better fit than just selecting the gates to be used for the primitive sizing from the library by hand as it would take into account the varying usage of differently sized versions of the cells.

The list of the primitive cells could be extended as required, but the set shown in this paper has been sufficient for use in some relatively large real designs. Keeping the primitive list short has the added benefit that it will be simple to update for a new cell library. This is especially handy when trying to compare the size of a design in two or more different processing technologies or standard-cell libraries.

4.2. Simple basic blocks

The basic blocks are simple re-usable units that are the basis for building the estimation models. These blocks are very generic and will be used in almost all designs. The accuracy of the estimation will depend on how accurate the basic blocks are, but as the blocks are very simple in nature, the accuracy can be made good even with very moderate effort. The blocks that will now be presented have been sufficient to give very good estimation results for the examples listed in this paper.

For some (especially arithmetic) basic blocks, the chosen implementation is the one giving the smallest area. This also usually translates to the largest propagation delay. In many cases, this is still a good first estimate. It requires, however, some insight from the designer to decide, if the speed of the modeled implementation is high enough for the application. If this is not the case, then another implementation should be modeled for this block.

We will now go through the most useful basic building blocks and describe the structures used and their gate counts.

4.2.1. Addition and related operations

Table 3 summarizes the gate count models for adder-like functions for N-bit operands. The adder that we have used in our modeling is based on the simple ripple-carry architecture. This is very straightforward, but it has proved to be quite accurate a model of most of the adders in our designs. It would also be possible to extend the adder model to include other adder architectures if their use seems more frequent.

The other adder based operations, subtraction, incrementing and decrementing, absolute value, and controllable addition and subtraction (ADD/SUB) can be modeled in a similar fashion with simple additions.

4.2.2. Reduction adder

Often, it is necessary to add a large amount of numbers together in a single operation during a single clock cycle. This can be achieved by a *reduction adder*. As this was relatively often used in our designs, we created a model for it separately. It can be shown that it is possible to add N1-bit inputs together with a tree of $N - \log_2(N)$ 2-input adders. This can be easily expanded to multiple bit input cases. The model then becomes

$$\mathcal{G}(\text{radd}(\text{ibits}, N)) = \text{ibits} \times (N - \log_2(N)) \times \mathcal{G}(\text{FA}) \tag{1}$$

4.2.3. Multiplication

The multiplier is a very important element in the implementation of many DSP algorithms. There are many different, increasingly efficient and complex, architectures for multipliers (see e.g. [14]) but in most cases the most basic ripple-carry or carry-save multiplier architectures will be sufficient for the performance required. Thus, we have selected the ripple-carry array multiplier for the basis of our modeling, as it is the smallest and most used of the parallel multiplier choices. The gate count of an $N \times M$-bit unsigned multiplier is given by the following:

$$\mathcal{G}(\text{mult}_{\text{uu}}(N, M)) = N \times M \times (\mathcal{G}(\text{FA}) + \mathcal{G}(\text{AND2})) \tag{2}$$

The basic multiplier structure works only with unsigned numbers. To be able to utilize signed numbers, the absolute value of the inputs is used for the multiplication and the sign of the result will be corrected afterwards by conditional negation. The multiplier size will be reduced by one in both dimensions, as the sign bits can be dropped. The gate count of a signed multiplier can then be calculated by the following:

$$\begin{aligned}\mathcal{G}(\text{mult}_{\text{ss}}(N, M)) = {} & \mathcal{G}(\text{abs}(N-1)) \\ & + \mathcal{G}(\text{abs}(M-1)) \\ & + \mathcal{G}(\text{mult}_{\text{uu}}(N-1, M-1)) \\ & + \mathcal{G}(\text{abs}(N+M-1))\end{aligned} \tag{3}$$

4.2.4. Multiplexers

In addition to the arithmetic building blocks described above, multiplexers are often used in data processing blocks. They allow conditional data processing to happen within the data paths by altering the signal flow based on control signals. The model we are proposing here is based on using 2-input multiplexers to build structures with more than two inputs. This is not the smallest possible, as many cell-libraries contain MUX cells with more than 2 inputs that result in smaller area. However, this simplified model has proved to give good estimation results. The N-bit M-to-1 multiplexer is built as a tree structure of 2-input multiplexers and its gate count is given by the following:

$$\mathcal{G}(\text{mux}(N, M)) = N \times (M-1) \times \mathcal{G}(\text{MUX2}) \tag{4}$$

4.2.5. Registers

The last basic building block is a simple N-bit register. The model for the register that we have used assumes that the design style uses synchronous reset for all registers. The model could easily be changed for asynchronous reset as well, but that would obviously change the size of all registers. Both the primitive

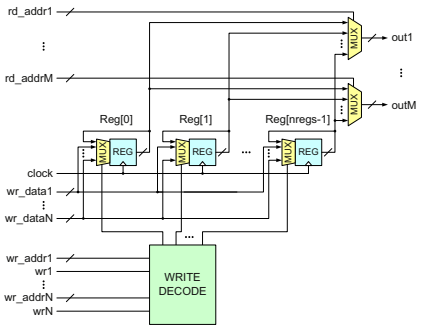


Fig. 1. Schematic picture showing multi-port register file.

register-bit size $\mathcal{G}$(FF) and the register block size should be altered if the reset style is changed. Mixing styles in one design would be bad design practice. In some specific cases there could be a need for a register without reset, which should be separately modeled on a case-by-case basis. The gate count of an N-bit single-port register can be estimated to be:

$$\mathcal{G}(\text{reg}(N)) = N \times (\mathcal{G}(\text{FF}) + \mathcal{G}(\text{MUX2})) \quad (5)$$

4.3. More complex basic blocks

The modeling of a few design blocks that will be used as a part of the basic block library will be illustrated next. These blocks act as detailed examples of the modeling process. For these blocks, we explain the model creation and then provide accuracy comparison against actual gate count, which was obtained by synthesizing the corresponding VHDL models using Synopsys DC.

4.3.1. Register files

Register files are an important part of many designs. We will now describe how to create a parameterized model for a register file with multiple input and output ports. The block diagram of the register file is shown in Fig. 1. The parameters of the register file model are:

1. *bits* – Register word length
2. *regs* – Number of registers
3. *iports* – Number of input ports
4. *oports* – Number of output ports

The output part is very simple: just *oports* multiplexers selecting the output from the register array.

The register array is slightly more complex to model, as each register bit needs to select the input from more sources than in a single port register case. The array can be implemented in a straightforward way so that the new value of each register is either selected from one of the input ports or the old value is retained. The new input selection is made in the write decode block.

The write decode block is perhaps the most complicated to model, as it needs to decode and match each address from each input port and generate select and write-enable signals to the register bit array. We also generate a write signal by ANDing the corresponding write-enable and detected address to generate a write-enable for each input port and register.

Finally, we can compute the total gate count of the register file by adding together the counts from the three different parts. The complete algorithm is listed in Algorithm 3.

Algorithm 3. Register file gate count estimation model.

```
1: function REGISTER_FILE(bits, regs, iports, oports, tech)
2:   % Output part
3:   o_mux ← oports*MUX(bits, regs, tech
4:   % Register array
5:   regs ← regs*bits*(tech.FF + iports*tech.MUX2)
6:   if iports > 1 then
7:     i_sel ← bits*regs*iports*tech.NAND2
8:     r_wr ← bits*regs*(iports − 1)*tech.NAND2
9:   else
10:    i_sel ← 0
11:    r_wr ← 0
12:  end if
13:  array ← regs + i_sel + r_wr
14:  % Write Decode part
15:  a_bits ← CEIL(LOG2(regs))
16:  a_inv ← iports*a_bits*tech.INV
17:  a_cmp ← iports*regs*a_bits*tech.NAND2
18:  i_wr ← iports*regs*tech.NAND2
19:  w_dec ← a_inv + a_cmp + i_wr
20:  return o_mux + array + w_dec
21: end function
```

To test the model we created a simple test with one and two input and output ports, which are the most common cases. The evaluation results for the register file model are shown in Table 4.

4.3.2. CORDIC phase rotator

The CORDIC [15,16] is a very versatile hardware building block, which can be used for many operations. For example, it can act as a phase rotator in communications and signal processing. Here we will show the creation of a parameterizable gate count model for an unrolled, parallel CORDIC phase rotator. Models for other CORDIC applications can be created in a similar fashion. The block diagram of a CORDIC phase rotator is shown in Fig. 2.

The parameters of the CORDIC phase rotator model are:

1. *niter* – Number of iterations
2. *ibits* – Input data word length
3. *pbits* – Input phase word length
4. *obits* – Output data word length

The pre-rotator rotates the incoming complex signal (x_{in}, y_{in}) by a multiple of 90°. This is equivalent to switching the signs and swapping the real and complex parts of the input. There are four possible combinations, which require two 2-to-1 multiplexers and a total of two shared conditional negations. The right shifters (RSH) are hard-wired and require no gates to implement. The remaining arithmetic parts are the ADD/SUB elements controlled by the look-up table outputs d_i.

The phase look-up table is generally hard to model accurately. For the CORDIC model we fitted the function

$$f(x, \alpha, \beta) = \alpha \cdot x \cdot \beta^x \quad (6)$$

to empirical data obtained by synthesizing the table for a few cases. One would expect that the parameter β would be close to 2 and α would reflect the redundancy factor. The fitting resulted in $\alpha = 0.6$ and $\beta = 1.8$. It should be noted that the input to the look-up table is two bits less than the phase input due to some bits being used in the pre-rotation.

Table 4
Register file model evaluation.

bits	regs	Inputs=1, Outputs=1			Inputs=1, Outputs=2			Inputs=2, Outputs=1			Inputs=2, Outputs=2		
		Actual gates	Model gates	Error (%)	Actual gates	Model gates	Error (%)	Actual gates	Model gates	Error (%)	Actual gates	Model gates	Error (%)
8	4	359	368	2.5	426	420	−1.4	541	548	1.3	635	601	−5.4
16	4	726	721	−0.7	864	827	−4.3	1082	1068	−1.3	1270	1174	−7.6
24	4	1083	1075	−0.7	1286	1233	−4.1	1607	1588	−1.2	1894	1746	−7.8
32	4	1437	1428	−0.6	1708	1640	−4.0	2146	2108	−1.8	2524	2319	−8.1
64	4	2860	2843	−0.6	3402	3265	−4.0	4262	4188	−1.7	4933	4610	−6.5
8	8	724	760	5.0	879	883	0.5	1114	1128	1.3	1288	1251	−2.9
16	8	1462	1485	1.6	1760	1731	−1.6	2210	2185	−1.1	2542	2432	−4.3
24	8	2141	2209	3.2	2574	2579	0.2	3274	3243	−0.9	3803	3612	−5.0
32	8	2858	2934	2.7	3454	3427	−0.8	4372	4300	−1.6	5068	4793	−5.4
64	8	5691	5833	2.5	6855	6819	−0.5	8642	8531	−1.3	9906	9516	−3.9
8	16	1522	1551	1.9	1853	1815	−2.1	2325	2301	−1.0	2647	2565	−3.1
16	16	3090	3018	−2.3	3737	3546	−5.1	4588	4434	−3.4	5207	4962	−4.7
24	16	4605	4486	−2.6	5563	5278	−5.1	6831	6566	−3.9	7780	7358	−5.4
32	16	6071	5953	−1.9	7395	7009	−5.2	9140	8699	−4.8	10 392	9755	−6.1
64	16	12 076	11 822	−2.1	14 511	13 934	−4.0	17 857	17 230	−3.5	20 287	19 342	−4.7

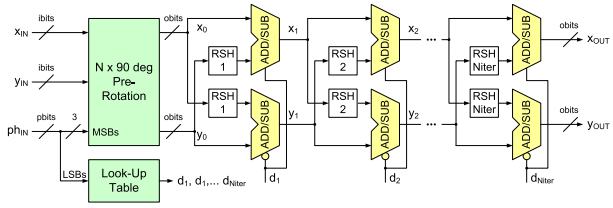


Fig. 2. Schematic picture showing CORDIC phase rotator.

The derivation of the CORDIC gate count model based on the above principles is an easy task, as long as close attention to the varying word length along the pipeline is taken into account. The algorithm of our model is listed in Algorithm 4.

Algorithm 4. CORDIC phase rotator gate count estimation model.

```
1: function CORDIC_ROT(niter, ibits, pbits, obits, tech)
2:   % Pre-rotation
3:   p_neg ← ibits*tech.XOR2
4:   p_mux ← 2*Mux(ibits, 2, tech)
5:   pre ← pre_neg + pre_mux
6:   % CORDIC iterations
7:   iter ← 0
8:   for i ← 1, niter do
9:     iter ← iter + 2*ADDSUB(obits − i)
10:  end for
11:  % Look-up table
12:  if pbits ≥ 3 then
13:    x ← pbits − 2
14:    lut ← niter*0.6**x*1.8^x*tech.NAND2
15:  else
16:    lut ← 0
17:  end if
18:  return lut + pre + iter
19: end function
```

To test the model, we created a simple test bench shown in Fig. 3. In this test bench the output of the look-up table for the phase signal is pipelined to optimize speed, as the propagation time through the look-up table can be quite large. The evaluation results for the CORDIC are shown in Table 5. It can be seen here that the model starts to fail with the smallest configuration tried. This is probably due to the smaller details of the implementation that have not been taken into account in the model.

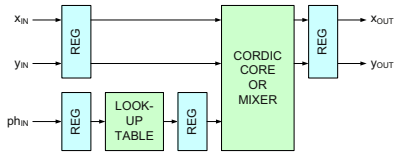


Fig. 3. Block diagram of the phase rotator test bench.

4.3.3. Complex mixer

The complex mixer is another very common block in many communications related designs. The mixer multiplies two complex numbers together, which requires four real multipliers and two adders in its simplest form. Usually the other input of the mixer is a complex sinusoidal signal, which is commonly generated by a table look-up with the phase of the signal as its input. This implementation is shown in Fig. 4. The parameters of the mixer model are:

1. *ibits* – Input data word length
2. *pbits* – Input phase word length
3. *obits* – Output data word length

Table 5
CORDIC model evaluation.

niter	ibits	pbits	obits	Actual gates	Model gates	Error (%)
3	3	3	5	465	392	−15.70
3	5	3	8	639	653	2.20
5	5	5	9	945	953	0.80
3	7	3	10	815	848	4.00
5	7	5	11	1169	1211	3.60
7	7	7	11	1634	1739	6.40
3	9	3	12	991	1042	5.10
5	9	5	13	1393	1469	5.50
7	9	7	13	2102	2061	−2.00
9	9	9	13	4156	4161	0.10

Table 6
Complex mixer model evaluation.

ibits	pbits	obits	Actual gates	Model gates	Error (%)
3	3	6	503	493	−2.0
5	3	8	705	735	4.3
5	5	10	1259	1210	−3.9
7	3	10	986	976	−1.0
7	5	12	1604	1563	−2.6
7	7	14	2551	2510	−1.6
9	3	12	1251	1217	−2.7
9	5	14	1961	1916	−2.3
9	7	16	2929	2975	1.6
9	9	18	5372	5466	1.7

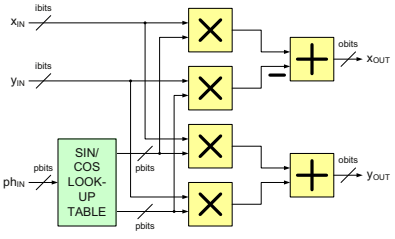


Fig. 4. Schematic picture showing the complex mixer.

The development of the gate count model for the mixer is very straightforward for the arithmetic part. The multiplier and adder models can directly be used here. We can easily count the required number of elements from the block diagram.

Again, the look-up table modeling poses a difficult problem. In this case, the table contains a full wave of $\sin(\omega)$ and $\cos(\omega)$. There is a great deal of redundancy in the table and this needs to be taken into account. The input and output of the tables are identical and the model is created by fitting a model function to the test data. We used the same function (6) as with the CORDIC look-up table for this case. Interestingly, the value obtained for β is the same for the CORDIC and the mixer cases. Fitting this function to synthesize $\sin(\omega)$ and $\cos(\omega)$ tables resulted in $\alpha=1.4$ and $\beta=1.8$.

The complete model is shown in Algorithm 5.

Algorithm 5. Complex mixer gate count estimation model.

```
1: function COMPLEX_MIX(ibits, pbits, obits, tech)
2:   % Complex multiply
3:   mult ← MULT_SS(ibits, pbits, tech)
4:   add ← ADD(obits, tech)
5:   sub ← SUB(obits, tech)
6:   mix ← 4*mult + add + sub
7:   % Look-up table
8:   lut ← 1.4*pbits*1.8^pbits*tech.NAND2
9:   return mix + lut
10: end function
```

To test the complex mixer model we used the same test bench as for the CORDIC (Fig. 3). The evaluation results of the model are shown in Table 6.

4.4. Design level modeling

At the design level the modeling process is essentially identical to the process illustrated above for the larger basic blocks. In the beginning, the design is divided into manageable sub-blocks for which models can easily be created. Normally, this is already done as a part of normal design practice. At the bottom level, the design consists of simple hardware elements, such as registers and arithmetic or logical operations. A block diagram or data-flow graph is drawn for the sub-blocks and the top level. Next, each item in the sub-block is mapped to a basic block from the library.

As an example, if the next element to be considered in the sub-block currently being modeled is an integer multiplication, we should examine its properties to decide what library block it could be mapped to. If the multiplication is parallel, and the result is signed, we should select the signed multiplier model MULT_{SS}. On the other hand, of the result is unsigned, we would use the MULT_{UU} model. The remaining task is to find suitable parameters, i.e. word lengths of the inputs, for the library model. In the case of serial implementation, we cannot directly use the multiplier models of the default block library, but must resort to creating the multiplier model from lower level components and handling it as a sub-block of the design. It would also be easy to add the serial multiplier to the basic block library after creating its model, if such multipliers are used often. In this case, suitable parameters should be used in its modeling.

If a mapping cannot be done because no such element exists in the library, the particular part must be modeled as primitives. In this case, there are a few alternatives. The first is to create a detailed model of the part using a similar process, as we did for the blocks in Section 4.3. This would be the most accurate method, and would lend itself best for the library extension. The second method, which works well for combinatorial table like structures, is to create a mathematical formula for the gate count estimation through fitting to example design data, like we did for example for the CORDIC and complex mixer examples. This works well, if the intention is to create re-usable and parameterized models. The third option is to approximate the gate count by counting the major logic elements. This is usually easier than it appears from the first thought, as the elements that cannot be mapped to the library after partitioning the design, tend to be small. Also, even though the accuracy of this approximation might not be very high, the overall effect to the accuracy of the gate count estimation of the whole design tends to be small. In the rare cases, where we encounter a element, which does not exist in the library, and is used in the design very many times, we should put more effort in its modeling. If this happens to be a part which is used in several places in the design, it will be a good candidate for adding to the basic block library, and the effort will be saved in future modeling work.

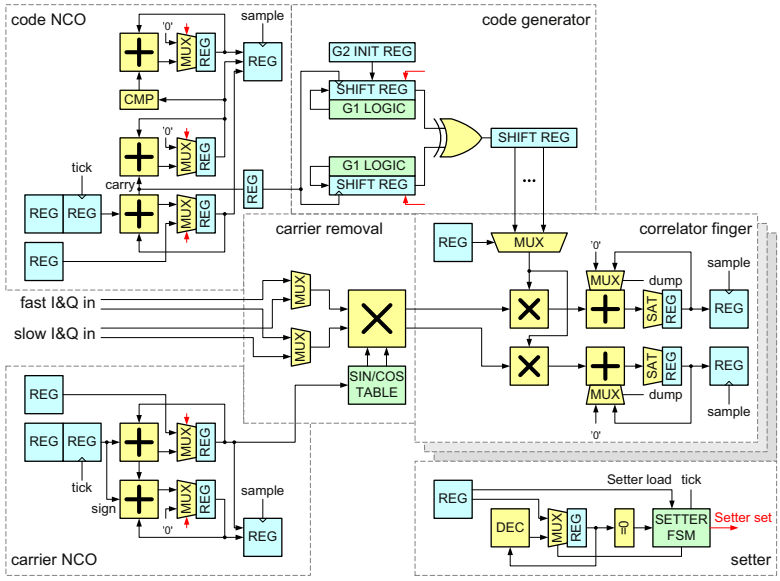


Fig. 5. Block diagram of the GPS correlator unit.

In the model creation for the real world examples presented in the next section, we could map almost all functionality to the basic block library. There were only a few specialized functions that required lower level modeling such as the code generator of the GPS receiver (Section 5.1), and even most of that could use blocks from the library. A versatile and rather general element in the basic block library is the generic *look-up table* model, which computes a gate count estimate given the number of inputs, outputs, and a redundancy factor. The model is exponential, and should not be used for large tables. It worked well for modeling the instruction decoding of the DSP processor core (Section 5.4) thus proving its suitability for general logic modeling.

5. Real world examples

All the preceding examples have been very limited and creating a detailed model for them is a very straightforward task. Furthermore, in such simple blocks it is possible to take into account even the smallest details, which can improve the modeling accuracy to a very good level. In this section, we will describe some examples from real designs and show that with a very small effort we can get very accurate models. These examples are extracted from real IC projects and their functionality has been validated in silicon or FPGA. The examples in this section include blocks from two different satellite positioning receivers and a processor core. The global navigation satellite systems (GNSS) include the American GPS [17] and the European Galileo [18]. We will not describe how to design such receivers and will not go into detail of the operation of the systems here as we will only look into the modeling of the hardware to evaluate our estimation methodology. An introduction to spread-spectrum communications and receiver design issues can be found for example in [19,20]. We will describe the hardware and discuss the gate count model creation of each example case first and then show the results at the end of this section.

5.1. GPS correlator unit

The GPS correlator unit consists of a number of *correlator channels* each with a number of *correlator fingers*. In addition there is a simple input processing block and a common control block and a simple bus-combination unit connecting all channels to the ASIC data bus. In this paper, we concentrate on the correlator channels, as the size of the other parts is negligible.

The block diagram of one correlator channel is depicted in Fig. 5 and its key parameters are listed in Table 7. The correlator channel design follows a typical GPS receiver architecture [21]. When the signal comes into the correlator, it is first multiplied by the carrier replica, which is generated by the carrier Numerically-Controlled Oscillator (NCO) and the sin/cos-table. Next it is multiplied by a delayed version of the replica code generated by the code generator driven by the code NCO. Finally, the signal is integrated for a set time before the results are copied to the output registers. There are a configurable number of *correlator fingers*, each multiplying the incoming signal by a differently delayed code and integrating it. The count of generated full cycles and the current NCO phase accumulator are sampled at the same time as the integrated data is sampled. There is also a *setter* unit, which initializes the NCOs to a desired phase and resets the code generator to start from the beginning of the code at a programmable time.

With the aid of Fig. 5 and Table 7, we can easily construct a gate count model of the correlator channel. Most of the building blocks

Table 7
GPS correlator parameters.

Parameter	Value	Notes
Number of channels	12–14	*Configurable*
Number of correlator fingers/ch	3–20	*Configurable*
Channel sample rate (MHz)	10.x/2.x	*SW selectable*
I/Q input word length	3	
NCO word length	32	
Initial NCO phase bits	16	
Carrier phase and cycles width (bits)	32	
Code phase and cycles width (bits)	42	
sin/cos table input word length	4	
sin/cos table output word length	5	
Correlator output word length	14	
Setter counter word length	16	

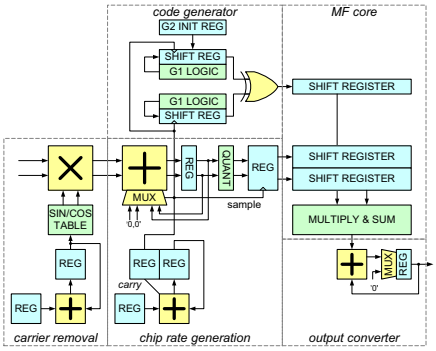


Fig. 6. Block diagram of the GPS matched filter unit.

in the block diagram are already available as basic blocks. The code generator G1 and G2 logic blocks are simply a few XOR2-gates [22]. As we know that there are a few miscellaneous register bits and the setter state requires two bits of state we can add those registers to the model. Furthermore, as we know that the FSM has 3 inputs and 2 outputs in addition to the state register, we can add an ad-hoc model of the logic using a simple look-up table model.

5.2. GPS matched filter unit

The GPS matched filter (MF) unit is used as a part of GPS receiver signal acquisition hardware. The unit presented here follows a similar architecture to that presented in [23]. The block diagram of the MF unit is shown in Fig. 6 and the key parameters are listed in Table 8.

First, the incoming signal is down-converted to DC with the carrier removal part that is similar to the one in the correlator channel. Next the signal enters into the decimator, which is implemented as an integrate and dump filter controlled by the code NCO and followed by a signal re-quantization block. The code NCO also drives the replica code generator, which generates the replica code that is initially loaded into the MF as the reference signal to be matched against. The MF core computes the matched filter output for each incoming sample. There are 1023 samples of data and reference signal in the shift registers. The data and reference shift-registers in the GPS MF unit have no reset to save their area and they need to be modeled separately from the normal register blocks. Each data and reference sample pair needs to be multiplied together and added together to produce a single output value. The MF core performs the computation in a bit-serial fashion by multiplying the 2×3 bits of data by the 1-bit, ±1 valued, reference signal one bit at a time. To simplify the operation, the data is represented in one's complement format [24], in which the negation of a number is done by inverting all of its bits. The inversion is done using a XOR2 gate. The parallel summation of 1023 single bits to a single output is handled by a reduction adder. The model of the *multiply and sum* part can be as simple as follows:

$$\mathcal{G}(\text{mul\_sum}) = 1023 \times (\mathcal{G}(\text{mux}(6,1)) + \mathcal{G}(\text{XOR2}) + \mathcal{G}(\text{radd}(1,1023)) \quad (7)$$

After the addition of all the bits, the remaining task is to assemble the three values corresponding to the I or Q part of the input to single numbers. This is done by the accumulator of the output converter part. The final output of the MF is a stream of data where the I- and Q-parts of each output sample are time-interleaved.

By observing Fig. 6 and referring to Table 8, we can proceed to create a gate count model of the GPS MF unit. Similar to the GPS correlator model, we can use the already modeled blocks to build the model except for the few parts that need a dedicated model to be created.

5.3. GNSS receiver block

We will next show the steps of creating a gate count model of a GNSS receiver based on the Group Correlator (GC) architecture [25]. Its top level block diagram is illustrated in Fig. 7 showing the major parts of the digital baseband of the receiver. In this paper, we concentrate on the *GC_block* level excluding the RAM memory blocks. The design is parameterizable for trading off its performance against silicon area. The main parameters and their values are shown in Table 9. There are also a lot of other interrelated parameters that can be derived from these and which need to be taken into account during the modeling process.

The work was divided into smaller pieces according to the design hierarchy and the model of the receiver block was built in a bottom-up fashion. We will now describe the model creation of the main sub-blocks illustrating the methods used.

5.3.1. Group correlator core

The group correlator core block diagram is shown in Fig. 8 and its parameters come directly from the top level table. It is evident that the size of this block is dominated by the large number of registers and the relatively big multiply and sum block, which multiplies and adds the parallel code and the data samples together to a single output. Here we can use the *reduction adder* model described earlier. The next task is to count the register bits in the code input shift-registers, the code holding registers, and the data shift-registers. The code and data are selected by multiplexers and depending on the GC operating mode the outputs of the parallel reduction adders are either added or interleaved by a multiplexer.

5.3.2. Carrier replica generator

Before we start to go through the carrier replica generator sub-block, we will first take a look of a more generic block diagram of a numerically controlled oscillator (NCO), which is used as a basis for the carrier and code replica generators. Fig. 9 shows a schematic diagram of a time-multiplexed implementation of an *N*-channel NCO. The NCO operates by accumulating the frequency control word to a phase accumulator register, which wraps over periodically after each complete output signal cycle. There is also

Table 8
GPS matched filter parameters.

Parameter	Value
Code spacing (chips)	1
Shift register length	1023
I/Q input word length	3
Data shift register word length	3
Carrier NCO word length	17
Code NCO word length	28
sin/cos table input word length	4
sin/cos table output word length	5
Matched filter output word length	13

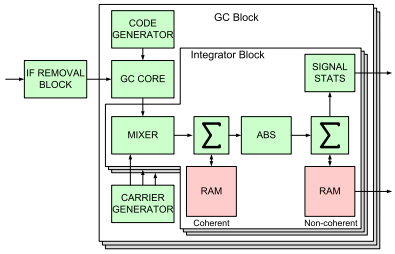


Fig. 7. Block diagram of the GNSS receiver. Adapted from [25].

Table 9
GNSS receiver parameters.

Parameter	Value	Notes
Defining parameters		
GC length	64	
GC multiplexing factor	4	
GC chaining factor	2	
Number of channels per GC block	16	
Number of GC input sample rates	3	
Number of GC blocks	2	
Number of integrator bins	5	
Input word length	3	*I* &Q
Derived parameters		
Total number of channels	32	
Code bins per channel	256	
Number of code bins per GC block	4096	
Total number of code bins	8192	

an associated cycle counter, which keeps track of the full cycles produced. The frequency of the NCO is controlled by writing the frequency registers via the write interface. The new values are first held in holding registers, whose sampling is controlled by the *sample* control signal. Similarly, the NCO phase and cycle counter registers for each channel are sampled at the same time and their values can be read over the read interface. This kind of a NCO implementation is very commonplace in GNSS applications, where multiple channels are tracking multiple input signals. GNSS receivers measure the incoming signal cycles and phase to derive the time of flight of the transmitted signals. The frequency, phase, and cycle count register values can be held in shift-register like arrays, but the accessible copies need to have register-file like structure allowing random access to the values from the SW. The phase adder and the cycle counter adder are normal adders and can be modeled as such.

Fig. 10 shows the carrier replica generator of the GNSS receiver. Two blocks on the top are similar to the multi-channel NCO discussed above. There are two additional NCO generating the running phase for creating the frequency bin spacing. The carrier replicas for the frequency bins are generated by adding an integer multiple of the frequency spacing phase to the center frequency phase or, for one bin, adding the offset frequency phase to the center phase. We can add the outputs of the two NCOs together and generate the same signal as we would do by adding the NCO control values because modulo arithmetic is used in the NCO. However, in the first case we need only two NCOs and a few adders, in the latter case there would be as many full NCOs as we have frequency bins. There is one additional feature, which is not directly visible in the figures: a desired value can be written to the NCO phase at a predetermined time via an additional initial value register. This is needed when initializing the receiver channels during the operation.

5.3.3. Replica code generator

The code replica generator for the GNSS receiver uses a pre-computed code, which is loaded into the code generator RAM before use. This is required for supporting the Galileo system, whose spreading codes are not based on the use of linear feedback shift registers, like they are in the GPS. The memory based code generator is more general and can also be used to generate the replica codes for the GPS. The block diagram of the replica code generator is shown in Fig. 11. In the bottom row, there is a NCO with some special features. The cycle counter has a per-channel programmable code length register, which is used to reset the memory pointer after each full cycle of the code. This is used to cope with the different lengths of the GPS and Galileo codes. Full code periods are counted with the epoch counter. The output of the cycle counter is used to produce the word and bit addresses for the code memory access logic on the top of the block. Above the NCOs there is the code setter, whose function is to reset the NCO and the counters to an initial value after a programmable number of clock cycles after receiving a synchronizing signal derived from a local time keeping unit. This allows resetting the phase of the generated code to a known value when initializing the receiver channels during the operation. The remaining circuitry allows using the Binary Offset Carrier (BOC) [26] modulation scheme used by the Galileo system when enabled for each channel. The bit selection logic in the code memory block is an 8-to-1 multiplexer and a single bit register.

5.3.4. Integrator bin

The integrator bin is responsible for the remaining hardware processing tasks in the GNSS receiver. Each integrator bin contains the following parts, as shown in Fig. 7:

- Frequency bin mixer
- Coherent integrator
- Absolute value computation
- Non-coherent integrator
- Statistics gathering block

Most of the sub-blocks in the integrator bin are simple. The frequency bin mixer is based on CORDIC, but it also down-scales the data to save some RAM bits for the following integration. For this reason the previous CORDIC model cannot be directly used here. However, the changes are straightforward and the earlier CORDIC model can be used as the basis for the new model. The two integrator blocks differ slightly from each other.

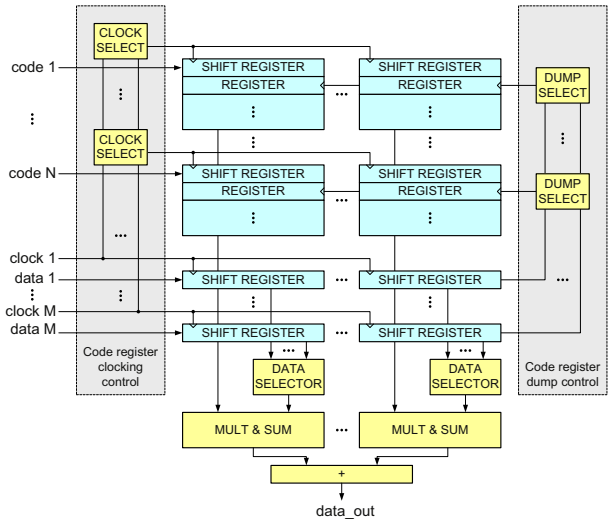


Fig. 8. Block diagram of a parameterizable Group Correlator core. Adapted from [25].

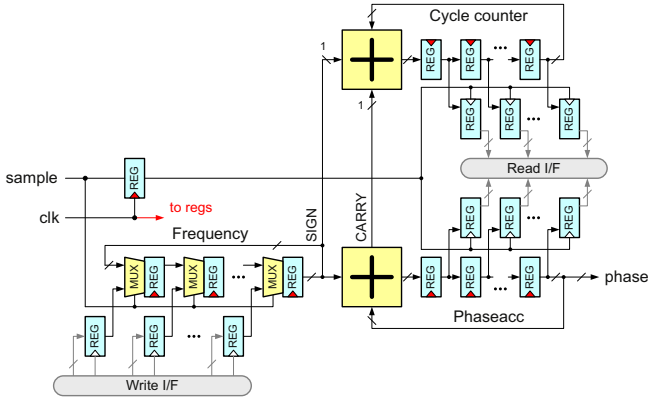


Fig. 9. Schematic diagram of a time-multiplexed generic NCO block.

The coherent integrator operates on the correlated complex samples that have been shifted to DC by the mixer. It adds a number of samples corresponding to the same code offset and channel together. These come as a time-multiplexed stream from the mixer. This means that the integrator running sum needs to be stored during processing the other streams. As there are a large number of streams to process, a RAM memory is used for the storage. For each sample processed, it has to be read from and written into the memory. It is possible to use two single-port memories by organizing the data storage in a clever way. This saves a large amount of silicon area. The integrator output and memory write data are registered. The integrator uses non-sticky saturation arithmetic to protect the integration result from overflow.

The absolute value computation logic computes an approximation of the magnitude of the complex data using the JPL algorithm [27], which yields a very good approximate of:

$$|I(t)+iQ(T)| = \sqrt{I^2(t)+Q^2(t)} \tag{8}$$

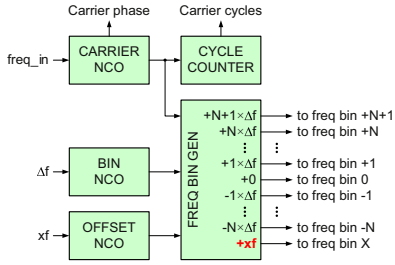


Fig. 10. Block diagram of the carrier replica generator.

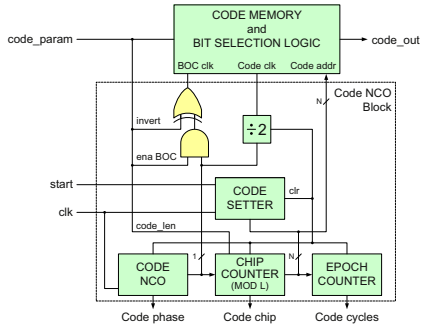


Fig. 11. Block diagram of the replica code generator.

requiring only very modest amount of hardware compared with a full implementation of the squaring and square root. The computation requires only shift and addition operations and a few multiplexers and comparators are needed for the control. This block also allows the data to be scaled down by a few bits.

The inputs of the non-coherent integrator are non-negative numbers that cause the integrator sum to be non-decreasing. In order to reduce the required word length, the unit keeps track of the minimum of the partial sums for each channel and subtracts it from all values while adding the next inputs. The subtracted values are accumulated separately and kept for each channel for adding later to the final result to restore the correct values for further processing. Due to the Doppler effect the spreading code frequency will be slightly different for each frequency bin. To compensate for this, there is a provision to add an offset to the integration memory addresses, which requires some additional logic in the memory address computation.

The last sub-block is the signal statistics computation unit. This block analyzes the results of the non-coherent integration during each processing round and provides the following data about the outputs of the non-coherent integrator for each channel: maximum value and its index, sum of all values, sum of all squared values. The sums accelerate the computing of mean and standard deviation of the output data. The main components in the statistics computation unit are the registers needed to store the data, the multiplier needed for the squaring operation and the adders for the accumulations.

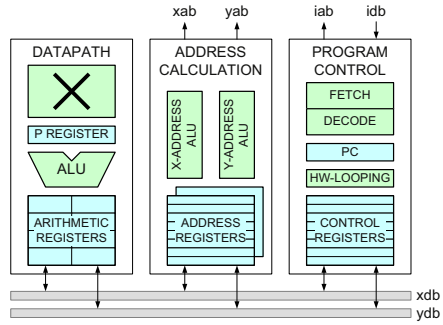


Fig. 12. Block diagram of the VS_DSP processor core. Adapted from [30].

Table 10
Main VS_DSP parameters.

Parameter	Range	Value	Notes
Data word length (n)	8–64	16	*Regs and buses*
Data address length (da)	8–23	16	$da \le n$
Program address length (pa)	11–20	16	$pa \le n$
Multiplier input width (m)	8–64	16	$m \le n$
Accumulator guard bits (g)	0–16	8	$m \le n$
# of arithmetic registers	4–12	8	*multiple of 2*
# of index registers	8, 16	8	
# of loop hardware levels	0–8	1	
Modulo addressing enable	T/F	T	
Bit-reverse addressing enable	T/F	T	

5.4. VS_DSP processor core

The VS_DSP [28,29] is a configurable, embedded, DSP processor core that can be used in many applications. The processor core is based on the modified Harvard architecture with two data buses (X and Y) and a separate program data bus. The VS_DSP user's manual [30] contains a very good description of the processor. In this paper, we will concentrate on the modeling of the core without explaining the hardware from the functional point of view. The block diagram of the processor core is shown in Fig. 12. The core consists of three functional units: datapath, data address calculation, and program control. In the following, we will now go through each of them and explain how the gate count model of the processor core was built. The main parameters of the VS_DSP core and their default values are listed in Table 10.

5.4.1. Datapath

The datapath of the VS_DSP is shown in Fig. 13. Its operation uses the register file except for the multiplier output, which is always stored into the P-register. The ALU can operate on either n or $2n+g$ long operands. The NULL/ONES is a pseudo-register, whose output is either all zeros or all ones. This can be used to form some useful operations, like increment, decrement, and negation. When the arithmetic registers are treated as $2n+g$ long concatenated words, the number of available registers is halved.

The multiplier is modeled as a normal signed multiplier, without taking into account the saturation. The fractional versus integer mode selector is a simple 2-to-1 multiplexer. The NULL/ONES register is modeled as a NAND2 array and the ALU input

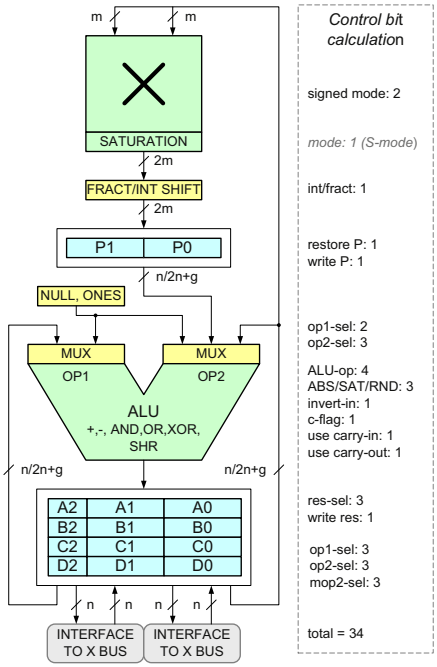


Fig. 13. Block diagram of the VS_DSP data path.

selectors are again normal multiplexers. The ALU is modeled with an adder/subtractor, a XOR2+AND2 gate array for logical functions, and a 5-to-1 multiplexer for the function selection. Notice that the ALU has to be $2n+g$ bits wide for the wide operations. The arithmetic registers are modeled as a register file. The register file has 3 input and 4 output ports. The bus interface model is simply a NAND2 array.

5.4.2. Data address calculation

The VS_DSP data address calculation unit consists of a bank of index registers and two address ALUs. The inputs to the address ALU are two index register values (base and modifier) and the addressing mode selection from the instruction word. The output of the address ALU is the updated value for the base index register. The optional bit reverse address-generation can be conceptually achieved by reversing the bits of the inputs and outputs of a normal adder. This reversion does not actually take any hardware and will be internally implemented by arranging the carry propagation of the adder. The normal index register incrementing addressing modes are implemented by a separate adder. A third adder is required to implement the modulo wraparound for optional hardware modulo addressing modes.

The modeling of the data address generation unit is fairly straightforward. The index register bank is modeled as a register file with 4 input ports and 6 output ports, which also accounts for the bus access to the two data buses. The address ALUs consist

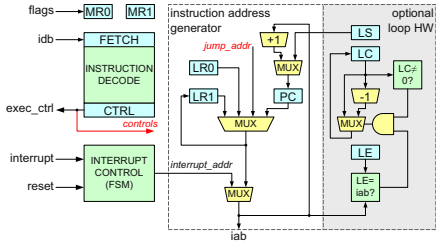


Fig. 14. Block diagram of the VS_DSP program control unit. Adapted from [30].

mainly of adders and multiplexers with word length equal to that of the index registers.

5.4.3. Program control

The VS_DSP program control unit is illustrated in Fig. 14. The control unit fetches and decodes the instructions, computes the next instruction address, and takes care of the interrupt handling. The VS_DSP is based on a three-stage fetch-decode-execute pipeline architecture, which makes the control unit quite simple. The PC is automatically incremented, except in the case of reaching the end of a hardware loop, when it is copied from the loop start register (LS). The link registers (LR1 and LR2) are used for long jumps and interrupt processing. The hardware loop is handled by comparing the instruction fetch addresses to the loop end (LE) register. If an end of a loop is reached and the loop counter (LC) is not zero, the LS register is copied to the PC causing the program to jump back at the start of the loop. In this case, the LC is decremented by one. The interrupt handler is a simple state machine, which controls the instruction fetch address and the storing of the old instruction fetch address to the LR1 register. The function of the interrupt handler is explained in detail in [30]. The fetched instruction is stored into the instruction fetch register and the decoded control signals are stored into the execution control registers at the end of each instruction decoding cycle.

The instruction address generation and loop hardware parts are modeled in a very straightforward fashion following the block diagram in Fig. 14. The modeling of the instruction decoder block is perhaps the most difficult challenge. It is also quite a large part of the processor, roughly 20% of the gate count. Its input consists of the fetched instruction and the condition and mode register (MR0). The outputs are the different control signals that go to the various parts of the processor. The first task is to determine the number of the control signals. For each unit, we computed the functional controls and bus interface controls separately. Fig. 13 illustrates the calculation of the number of functional control signals for the datapath. The other blocks are handled in a similar way. At the end, we have the total number of outputs from the instruction decoder block. There are a total of 40 inputs and 105 outputs in the instruction decoder logic. The instruction decoder block can be viewed as a PLA, which would consist of decoding each input combination ($2^{Ninputs}$ possibilities) and then for each output selecting the ones that cause the control to have a value of one. Such a model would yield unrealistically large an estimate if we would simply use a single decoder and take all inputs and outputs into account. In reality, the instruction decoder function is rather sparse and all of the input bits do not have any effect on some of the output bits. To be able to cope with this, the instruction encoding has to be taken into account and the inputs and outputs need to be considered more carefully. In the VS_DSP

Table 11
VS_DSP instruction decoder partial decode list.

Function	In	Out
General instruction selection	4	68
Control instruction selection	3	16
Condition code evaluation	14	1
ALU instruction selection	8	34
Datapath register move instruction selection	4	4
Index register move instruction selection	4	4
Control register move instruction selection	4	4
Move instruction format selection	3	3

case, the instruction decoder was modeled in smaller pieces as shown in Table 11. Additionally the model contained some multiplexers to select instruction word fields for decoding. This partitioned model yielded surprisingly good results, as we can see below. The interrupt control state machine has 6 states, 3 inputs and 6 outputs and it can again be modeled as a state register and a look-up table.

5.5. Results summary

The results of the modeling of the example cases described in the previous subsections are shown in Table 12. The actual gate count numbers used in the comparison are based on data provided by Synopsys DC runs when they were synthesized during the IC projects. The modeled gates are obtained from the developed model. The modeling errors are given in absolute gates, and as relative percentage of the actual gate count. As it can be seen from the results the modeling accuracy using the proposed method is very good and the average error of the test cases is 4.0%. The accuracy achieved for the test cases compares very well against those reported in the literature, especially considering the high abstraction level of the input data. Our example cases have been taken from real design projects that have been implemented in two different CMOS technologies from two different silicon foundries. This compares well with the overall average relative error of 3.2% calculated from all of the smaller examples shown in Tables 4–6.

For the GPS receiver blocks, there are two different configurations for the *Correlator*: configuration 1 has 7 channels with 5 correlator fingers and 7 channels with 3 correlator fingers and configuration 2 has 12 channels with 11 correlator fingers.

The *Integrator bin* in the GNSS receiver has an error of −9.6%, but most of it can be accounted to unmodeled testability and interface logic in the integration memory blocks. This can be seen in the row below, where comparison is made between only the parts which were modeled. This difference highlights the general problem of early modeling of gate count: there can be unforeseen changes during the implementation phase, which would cause differences between the estimated and actual gate count. Logic related to testability is one such source and in our example, the other is the logic which was used to emulate a two-port memory with two single-port memories. However, these kind of estimation errors are not specific to the estimation methodology. Even synthesizing early RTL code using the actual implementation technology would still have similar problems. It should be noted here that the GNSS receiver block numbers in the results table include five instances of the final integrator bin and not the partial one. This shows that modeling errors in the sub-blocks tend to be averaged out when considering the whole design.

The VS_DSP modeling results are very good especially considering the uncertainties in modeling the large instruction decode logic. The results shown in Table 12 correspond to the parameters given in Table 10. As an example of using the parameterization of the gate count models for design space explorations, some variations of the VS_DSP with different parameters are illustrated in Table 13.

Table 12
Example cases results.

Design	Actual gates	Model gates	Error gates	Error (%)
GPS receiver blocks: 0.18 μm CMOS				
GPS correlator conf. 1	96 400	99 385	2985	3.1
GPS correlator conf. 2	135 000	140 700	5700	4.2
GPS matched filter	47 972	48 248	276	0.6
GNSS receiver blocks: 65 nm CMOS				
GNSS receiver block	249 984	250 437	453	0.2
Group correlator core	65 799	74 588	8789	13.4
Carrier replica gen.	25 114	25 941	827	3.3
Replica code gen.	28 909	28 791	−118	−0.4
Integrator bin	24 375	22 023	−2352	−9.6
Integrator bin parts	21 248	22 023	775	3.6
Processor core: 0.18um CMOS				
VS_DSP (v2)	22 000	21 656	−344	−1.6

Table 13
VS_DSP parameter alteration results.

Version	Estimate gates	Δ Baseline gates
VS_DSP (v2) baseline	21 656	–
With 24-bit datapath	26 148	4492
No loop hardware	20 612	−1044
No modulo and bitrev addressing	20 555	−1101
With 4 arithmetic registers	19 498	−2158
Minimum implementation	10 966	−10 690

6. Conclusions and summary

In this paper, we have presented a methodology for gate count estimation of ASIC designs. The proposed gate count estimation methodology can be applied at a very early phase of the design process. It is based on simple bottom-up modeling of the system. Using a small set of simple, well characterized, building blocks allows good accuracy without requiring complex analysis of the design. The set of building blocks is easily extensible, which allows tailoring the methodology for particular applications and design styles. It can be used even when there is an incomplete specification available. It is also possible to apply the methodology in an incremental fashion with increasingly better estimation accuracy. The possibility of creating parameterized estimation models allows this method to be used in architectural level design exploration. It is for example a very efficient tool, when area-performance trade-offs need to be made early in an ASIC design project.

It would be advantageous for this method to improve the modeling of random logic structures like look-up tables and decoding logic, which are currently handled in an ad hoc fashion or matching a number of similar, synthesized circuits to a parameterized model. The other valuable addition would be to enable memory blocks to be included in some fashion into the area estimations. Currently, they are ignored and they need to be taken into account separately.

As we have shown with the example cases, it is possible to get very accurate estimations even with very limited effort. The smaller examples show in detail how the method is applied to smaller blocks and the large example cases show how accurate results can be achieved in realistic designs.

Acknowledgments

The authors would like to thank Seppo Turunen from Nokia for providing actual gate count data for the example cases and valuable comments. This work has been supported by a grant from the Alfred Kordelin Foundation.

References

[1] C. Shannon, The synthesis of two-terminal switching circuits, Bell Syst. Techn. J. 28 (1949) 59–98.
[2] D.E. Muller, Complexity in electronic switching circuits, IRE Trans. Electron. Comput. EC-5 (1956) 15–19.
[3] E. Kellerman, A formula for logical network cost, IEEE Trans. Comput. C-17 (1968) 881–884.
[4] R.W. Cook, M.J. Flynn, Logical network cost and entropy, IEEE Trans. Comput. C-22 (1973) 823–826.
[5] N. Pippenger, Information theory and the complexity of Boolean functions, Theory Comput. Syst. 10 (1976) 129–167.
[6] K.-T. Cheng, V. Agrawal, An entropy measure for the complexity of multi-output Boolean functions, in: Proceedings of the 27th ACM/IEEE Design Automation Conference (DAC), 1990, pp. 302–305.
[7] M. Nemani, F. Najm, High-level area and power estimation for vlsi circuits, IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 18 (1999) 697–713.
[8] K.M. Büyükşahin, F.N. Najm, High-level area estimation, in: Proceedings of the 2002 International Symposium on Low Power Electronics and Design, ISLPED '02, ACM, New York, NY, USA, 2002, pp. 271–274.
[9] R. Enzler, T. Jeger, D. Cottet, G. Tröster, High-level area and performance estimation of hardware building blocks on FPGAs, in: Proceedings of the Roadmap to Reconfigurable Computing, 10th International Workshop on Field-Programmable Logic and Applications, FPL '00, Springer-Verlag, London, UK, 2000, pp. 525–534.
[10] M. Abdelhalim, S.-D. Habib, Fast FPGA-based area and latency estimation for a novel hardware/software partitioning scheme, in: Canadian Conference on Electrical and Computer Engineering, 2008, CCECE 2008, 2008, pp. 000775–000780.
[11] R. Meeuws, A Quantitative Model for Hardware/Software Partitioning (Master's Thesis), Delft University of Technology, Delft, Netherlands, 2007.
[12] Matlab, URL: ⟨http://www.mathworks.com/Matlab⟩, 2013.
[13] J.W. Eaton, GNU Octave. URL: ⟨http://www.gnu.org/software/octave⟩, 2013.
[14] J.-P. Deschamps, G.J.A. Bioul, G.D. Sutter, Synthesis of Arithmetic Circuits: FPGA, ASIC and Embedded Systems, John Wiley & Sons, Inc., Hoboken, NJ, 2006.
[15] J.E. Volder, The cordic trigonometric computing technique, IRE Trans. Electron. Comput. EC-8 (1959) 330–334.
[16] B. Lakshmi, A.S. Dhar, Cordic architectures: a survey, VLSI Des. 2010 (2010) 2:1–2:7.
[17] Navstar GPS Space Segment/Navigation Users Interface. GPS Interface Specification, IS-GPS-200F. URL: ⟨http://www.gps.gov/technical/icwg/IS-GPS-200F.pdf⟩, 2011.
[18] European GNSS (Galileo) Open Service: Signal In Space Interface Control Document, Issue 1 Revision 1. URL: ⟨http://ec.europa.eu/enterprise/policies/satnav/galileo/files/galileo-os-sis-icd-issue1-revision1_en.pdf⟩, 2010.
[19] R. Pickholtz, D. Schilling, L. Milstein, Theory of spread-spectrum communications-a tutorial, IEEE Trans. Commun. 30 (1982) 855–884.
[20] M.K. Simon, J.K. Omura, R.A. Scholtz, B.K. Levitt, Spread Spectrum Communications Handbook, McGraw-Hill, New York, NY, USA, 1994 (revised edition).
[21] E.D. Kaplan, Understanding GPS: Principles and Applications, Artech House Publishers, Norwood, MA, USA, 1996.
[22] J.J. Spilker, Signal structure and performance characteristics space segment, NAVIGATION 25 (Summer) (1978) 121–146.
[23] V. Eerola, Rapid parallel GPS signal acquisition, in: Proceedings of the 13th International Technical Meeting of the Satellite Division of the Institute of Navigation (ION GPS 2000), Salt Lake City, UT, 2000, pp. 810–816.
[24] D.E. Knuth, The Art of Computer Programming: Seminumerical Algorithms, 3rd ed., Addison Wesley, Reading, Massachusetts, 1998.
[25] V. Eerola, S. Pietilä, H. Valio, A novel flexible correlator architecture for GNSS receivers, in: Proceedings of the 2007 National Technical Meeting of the Institute of Navigation (ION NTM 2007), San Diego, CA, 2007, pp. 681–691.
[26] J.W. Betz, The offset carrier modulation for GPS modernization, in: Proceedings of the 1999 National Technical Meeting of the Institute of Navigation (ION NTM 1999), San Diego, CA, 1999, pp. 639–648.
[27] B. Levitt, G. Morris, An improved digital algorithm for fast amplitude approximations of quadrature pairs, The Deep Space Network Prograss Report 42-40, Jet Propulsion Laboratory, Pasadena, CA, 1977.
[28] J. Nurmi, J. Takala, A new generation of parameterized and extensible DSP cores, in: IEEE Workshop on Signal Processing Systems: Design and Implementation (SIPS), 1997, pp. 320–329.
[29] J. Takala, M. Kuulusa, P. Ojala, J. Nurmi, Enhanced DSP core for embedded applications, in: IEEE Workshop on Signal Processing Systems: Design and Implementation (SIPS), 1999, pp. 271–280.
[30] VSDSP-Manual, VSDSP2 Processor Core Manual, VLSI Solution Oy, Tampere, Finland, version 2.6. URL: ⟨http://www.vlsi.fi/fileadmin/manuals_guides/vsdsp2_um.pdf⟩, 2001.

Ville Eerola received an MSc degree in electrical engineering from Tampere University of Technology in 1990. He is currently a PhD student at Tampere University of Technology. He worked from 2004 to 2011 as a Senior Specialist at Nokia Corporation in Finland. From 1992 to 2003 he was involved in digital ASIC design and GPS receiver development at VLSI Solution and u-Nav Microelectronics. He has been involved in GPS receiver development since 1992 and has led the development of several GPS receivers. His current research interests include GNSS receiver hardware implementation. He is a co-founder of VLSI Solution Oy and u-Nav Microelectronics Corporation. He is a senior member of the IEEE.

Jari Nurmi is a Professor at TUT since 1999, at the Department of Electronics and Communications Engineering. His research interests include embedded computing systems, high-level design methodology, positioning receivers, wireless localization, and software-defined radio. He held various research, education and management positions at TUT since 1987, was the Vice President of VLSI Solution Oy 1995–1998, and is a co-founder of a start-up Ekin Labs Oy since 2013. He was one of the recipients of Nokia Educational Award 2004, won the Tampere Congress Award 2005, was an Academy of Finland Research Fellow 2007–2008, was the co-recipient of IIDA Innovation Award 2011, and was selected for the Scientific Congress Award 2013. He is a steering committee member of five international conferences. He has edited two Springer books, and published over 270 international conference and journal articles and book chapters.

PUBLICATION 6

Ville Eerola and Jari Nurmi, "Area Estimation of Time-Domain GNSS Receiver Architectures," in *2014 International Conference on Localization and GNSS (ICL-GNSS)*. IEEE, 2014.
DOI: 10.1109/ICL-GNSS.2014.6934163

Area Estimation of Time-Domain GNSS Receiver Architectures

Ville Eerola, Jari Nurmi
Department of Electronics and Communications Engineering
Tampere University of Technology
P.O.BOX 553, FIN-33101 Tampere, Finland
Email: ville.eerola@tut.fi

***Abstract*—This paper presents the silicon area estimation of three different GNSS receiver architectures with the analysis of four different use cases. The receivers are based on traditional correlator, matched filter, and group correlator architectures. While introducing the selected test cases, the authors discuss their applicability for real-life receiver operations. The receiver architectures are described shortly and the implementation details explained. The comparison shows that the correlator based receiver suits best for tracking while matched filters are efficient only for pure search mode. The group correlator offers a good area-efficiency in both tracking and search modes.**

I. Introduction

GNSS receivers are becoming a common feature in a wide range of electronic devices. While their operation and implementation are well understood, there has not been a good analysis how different architectures and use-cases affect the gate count of the receivers. The gate count is an important factor affecting both the cost and power consumption of the receivers. As GNSS receivers are targeted toward cheaper and lower-power applications, good understanding of the receiver gate count becomes more important.

At the deepest level, all time-domain GNSS receivers correlate the received signal with locally generated replicas of the residual carrier and pseudorandom number (PRN) based spreading code. The receivers are typically organized as channels, where one channel is usually capable of processing the received signal from a single satellite. Such a conceptual receiver channel is illustrated in Fig. 1. The channels of the receiver architectures can be differentiated by the number of code phases and carrier frequencies as well as integration capabilities. Thus, it is possible to compare the different receiver architectures by defining common operating conditions, which can be used to calculate the number of satellites, code phases, and carrier frequencies that the receiver has to be able to process simultaneously. Additionally, some general constraints need to be defined for keeping the resulting designs realizable.

In this paper we will concentrate strictly on the GPS system [1], which simplifies the analysis, but it would be easy to generalize the treatment to include other GNSS systems such as GLONASS, Galileo, Quasi-Zenith, or Beidou as well as Satellite Based Augmentation Systems (SBAS). Furthermore, the exact definition of the test cases is exemplary and they could easily be adapted to any other situation when needed.

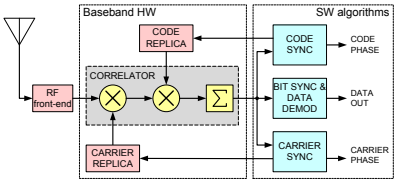


Fig. 1. Conceptual diagram of GNSS receiver channel structure.

In the next section, we will define the operating conditions and test cases for the receiver architecture evaluation. Then we describe the three architectures that will be compared and define the parameters that will need to be adjusted to comply with the test case definitions. After the architecture introductions, we will shortly describe the method of the area estimation and show the area estimation results. Finally, we will conclude the paper with a discussion of further research possibilities.

II. Defining the Test Cases

We will look into three different use cases which will be used to define the requirements for the GNSS receivers used in the area estimation. These are acquisition, tracking, and assisted acquisition. The operating conditions presented here are typical for modern GPS receivers. During the analysis of the different architectures, we will use the term *'cell'* or *'search bin'* to denote a correlation operation of a single replica PRN code, replica code phase and carrier replica frequency. We will not use the term *'correlator'* for it, as it means multiple things for different people.

A. Acquisition

Signal acquisition is the most resource demanding operational phase of any GNSS receiver. The reason is that the received satellite signals are very weak, and the search space is typically very large. At least four satellites need to be found before a position, velocity and time (PVT) solution can be computed, and often there can be more than 12 GPS satellites visible to search for.

The acquisition problem is well-understood, and discussed in the literature. For example, van Diggelen [2] presents

some of the challenges for high sensitivity acquisition. The acquisition sensitivity requirement will determine the required dwell-time, which will affect the size of a single search cell along with the desired code phase accuracy. For the unassisted GPS case, the longest coherent integration time will be limited by the data message bit duration. Usually, it will be below 10 ms, and in open-sky cold-start case it can be equal to the code period (1 ms), which minimizes the losses due to unknown data bit transitions. As the search bin width in the frequency dimension is given by [3]

$$\Delta f_{bin} < \frac{1}{2T_C}, \tag{1}$$

the frequency bin width will vary between 500 Hz and 50 Hz. The usually observed Doppler uncertainty is ± 5 kHz and a typical reference frequency uncertainty can be around 2 ppm, which equals a range of ± 3.2 kHz. Thus the total frequency uncertainty during acquisition without aiding could be ± 8.2 kHz.

In the code phase dimension, the search area is bounded by the duration of a single length of the spreading code, which is 1 ms or 1023 chips. In most cases, the search accuracy is $1/2$ chips, as this will be a good trade-off between alignment loss and required complexity.

In the PRN code or satellite dimension, the case will be limited by the visible satellites when a rough location of the receiver and time is known or the total number of satellites in the whole constellation, if no information is available. Typically, the acquisition is based on the assumption that the visible satellites can be predicted effectively halving the search space.

The acquisition uncertainty space is visualized in Fig. 2 and the acquisition test case parameters are summarized in Table I. We can use the sensitivity requirement to calculate the required integration times, which gives us the number of frequency bins and that can be used to compute the total number of search bins needed. If we wish to cover the whole uncertainty area in the given time, we repeat the search approximately five times, which will finally give us the required number of parallel search bins needed. The code resolution of the acquisition case demands a sampling rate of 2.046 MHz.

B. Tracking

Hardware-wise, the tracking case is much less demanding than acquisition, as the number of required processing cells is much less. Usually, it would be necessary to only process three to five different code phases at a single frequency for each satellite depending on the complexity of the receiver signal processing algorithms. For tracking, receivers usually employ a closer separation of the code phases to improve the code tracking accuracy. A good separation for a GPS receiver would be $1/8$ chips. Most modern GPS receivers are designed to track all available signals. This would require up to 16 channels to include a few operational spare satellites and SBAS signals.

The performance requirements such as tracking sensitivity and signal dynamics handling capability usually have little effect on the hardware requirements other than avoiding implementation losses. The tracking case hardware requirements are shown in Table II. The code resolution of the tracking case demands a sampling rate of 8.184 MHz.

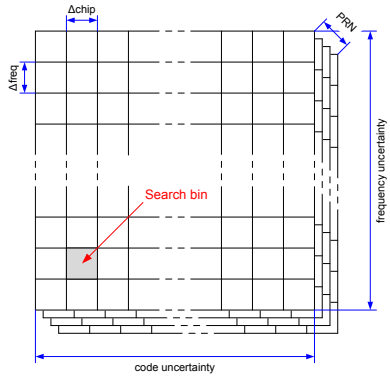


Fig. 2. Visualization of the acquisition space.

TABLE I
ACQUISITION TEST CASE PARAMETERS

Parameter	Value	
Time to acquire ≥ 4 satellites	10	s
Acquisition sensitivity	31	dB-Hz
Coherent integration time	3	ms
Dwell-time	1800	ms
Number of PRN codes	12	
Frequency bin size	167	Hz
Frequency search range	± 8.2	kHz
Number of frequency bins	17	
Code phase search range	1023	chips
Code phase resolution	$1/2$	chips
Number of code bins	2046	
Total number of search bins	417384	
Parallel search bins	83000	

TABLE II
TRACKING TEST CASE PARAMETERS

Parameter	Value	
Number of PRN codes	16	
Number of frequency bins	1	
Code phase resolution	$1/8$	chips
Number of code bins	5	
Total number of bins	80	

C. Assisted GNSS

The assisted GNSS (A-GNSS) refers to the case, where the GNSS receiver is provided with out-of-band assistance about the satellites to be received. This information usually includes predicted satellite positions in the sky and satellite data message, and may also include more accurate information about the receiver reference frequency and current time. This information can be used to reduce the necessary search range for acquisition, as well as improve positioning sensitivity through not requiring the GNSS receiver to be able to decode the satellite data message. The assistance data is standardized for different systems. One such standard is the 3rd Generation Partnership Project (3GPP) A-GNSS performance requirements specification [4]. There is an associated test specification [5] which will define test cases to ensure the performance will be met.

The 3GPP test specification for GPS case specifies a set of test conditions, where the assistance data as well as the visible satellites and their signal strength are specified. All A-GPS receivers must pass the tests and many GPS receivers today exceed the minimum requirements. Usually the tests to guarantee the performance will be modified from the standard tests by just reducing the satellite signal strengths. For most receivers, the most demanding test in the 3GPP specification will be the coarse-time assistance sensitivity test. In that test, at least 9 satellites are visible and 8 of those will be present during the test. One of the signals will have a higher signal strength and the rest will have lower power. The time uncertainty in that test is such that it will cover the whole code period. In the accurate-time assistance sensitivity test all signals will be equally weak, but the time is known within $\pm 10\,\mu$s or ± 10 chips. The availability of such an accurate timing information to the GNSS receiver will not be straightforward in real life though. The A-GPS specifications also contain some tracking tests, but usually they will not be more demanding for the hardware than in the normal tracking case as described earlier.

Table III contains the summary of the parameters for the A-GPS test case. In our case, we have set the frequency uncertainty at a higher level than strictly demanded in the A-GPS spec, as this is more typical in a real-life situation. For the acquisition time, we also set a lower limit, as some of the time allowed in the specification will be spent in the tracking mode and wasted in protocol overhead.

D. General constraints

For the purposes of this paper, we will limit the maximum operational clock frequency to approximately 128 MHz, or 128 times the C/A code frequency. This would be a suitable choice for chip implementation using a low-power 65 nm CMOS technology.

For the comparison, we will create a single design for each receiver. We will adjust the parameters of the receiver so that it will be able to fulfill the requirements of all the preceding test cases.

TABLE III
A-GPS TEST CASE PARAMETERS

Parameter	Value	
Time to acquire ≥ 4 satellites	5	s
Acquisition sensitivity (low)	27	dB-Hz
Acquisition sensitivity (high)	32	dB-Hz
Coherent integration time	9	ms
Dwell-time	1400	ms
Number of PRN codes	9	
Frequency bin size	55	Hz
Frequency search range	± 100	Hz
Number of frequency bins	4	
Code phase search range	1023	chips
Code phase resolution	$1/2$	chips
Number of code bins	2046	
Total number of search bins	73656	
Parallel search bins	32736	

All implementations will have all elements that are required for the baseband processing in the GPS receiver including replica carrier and code generators and integration blocks.

III. Three Architectures

In this section, we will show receiver implementations using the three architectures to be compared. We will first describe the architecture and then go shortly over the main aspects of the implementation. The gate count estimations will be presented in the next section and compared. The implementations will be optimized for area. Power consumption would be subject for another paper.

The replica carrier and code generators for receiver implementations of the three architectures will be the same, as the blocks are similar in all cases. The exception is the matched filter (MF), which does not need the code generator to be running all the time with variable rates. We will utilize time-multiplexing for the implementation when applicable. It will be constrained by the maximum clock frequency as previously described.

The code generator includes a numerically controlled oscillator (NCO) driving the GPS C/A-code generator implemented with on two linear-feedback shift registers (LFSR), G1 and G2. The code selection is done via setting the initial state of the G2 LFSR. Fig. 3 illustrates the replica code generator.

The carrier replica generator will have a NCO generating the carrier phase, and a second NCO driving the generation of a linearly spaced set of frequencies around it. The block diagram is shown in Fig. 4.

To aid in the processing of the huge amount of signals that need to be handled in the acquisition and A-GPS cases, we use a post-integrator block, which will take care of coherent integration, amplitude detection and non-coherent integration of samples that are correlated with replica code and carrier and pre-integrated in the receiver core processing blocks. The structure of the post-integrator is depicted in Fig. 5. The post-integrator has the possibility of bypassing the amplitude

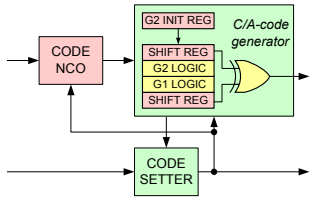


Fig. 3. Block diagram of the common replica code generator.

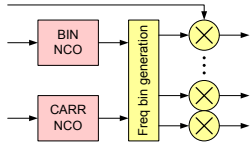


Fig. 4. Block diagram of the common carrier replica generator.

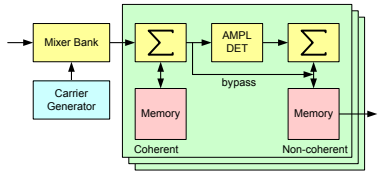


Fig. 5. Block diagram of the common post-integrator block.

detector so that the coherently integrated samples are stored into the non-coherent integration memory for use by the receiver software algorithms.

A. Correlator

The correlator is the oldest and most widely used architecture used for GNSS signal tracking. Correlator architecture optimization and implementation have been discussed by the authors in [6]. In this paper, the receiver based on the traditional correlator architecture is time-multiplexed to minimize the implementation area. The block diagram of the receiver is shown in Fig. 6. To optimize the correlator architecture, time-multiplexing is used so that multiple channels (PRN replicas) and/or carrier replicas can be processed using single datapath. The different code phase fingers are processed in parallel within the correlator block. This allows using a largish number of fingers to cover larger segments of code for the acquisition case. The carrier and code generators for the receiver are maximally time-multiplexed to optimize the receiver gate count. The code and carrier replica generators are implemented in such a way that the same number of replicas will be generated as are needed in the correlators. This may lead to an optimistic area estimation as the allocation might

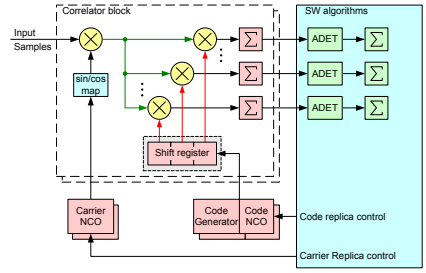


Fig. 6. Block diagram of time-multiplexed correlator based architecture.

be difficult to realize. When used in the acquisition mode, the correlator will generate an enormous amount of outputs within each coherent integration interval, which are difficult to process in the receiver software. It would be possible to aid the processing by adding a post-integrator block after the correlators to process the data already in hardware. In our paper, we have not included the post-integrator as a part of the correlator in the area evaluation.

B. Matched Filter

The matched filter is another well-known architecture for performing correlation operation especially for signal acquisition. The matched filter use in GPS signal acquisition has been presented e.g. in [7]. The architecture that we will use in this paper is based on the one documented in [8]. The block diagram of the matched filter based receiver channel is depicted in Fig. 7. We have used a custom version of the code generator in the MF, as the code code can be frozen within the MF during the operation. The original design was not designed for tracking functionality, but it can be easily implemented in the receiver software. This additional functionality would require bypassing the absolute value function after the coherent and non-coherent integrators. The way that the carrier removal is done before the matched filter core and the clever method of using adjustable decimation for sample rate adjustment allows for carrier and code tracking in a similar fashion as with the traditional correlator based design. The MF core performs the calculation of the result in bit-serial manner, meaning that each bit as well as real and imaginary part of the input data samples are multiplied by the code replica and summed one after each. The results are combined at the output so that a single complex output is generated. This will take a number of clock cycles to do, but there is still a possibility to multiplex the different replica codes in a way shown in [9]. The bandwidth of the MF is limited by its length, which limits all signals to be processed at the same time to within its pass band. This is not practical for GPS receivers except for the cold-start acquisition case, where the signals are spread over a wider frequency range. Thus, our implementation utilizes the code multiplexing possibility only for the acquisition case.

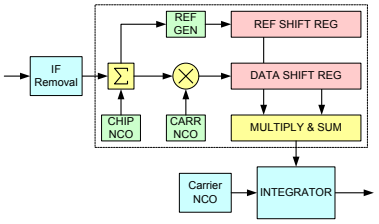


Fig. 7. Block diagram of a high sensitivity matched filter architecture.

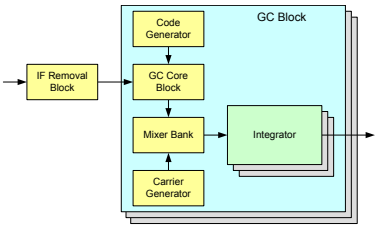


Fig. 8. Block diagram of a group correlator based receiver (based on [10]).

C. Group Correlator

The group correlator (GC) is a more recent architecture for GNSS receivers. It can be seen as a hybrid between the traditional correlator and matched filter architectures. The group correlator receiver was presented in [10]. The group correlator based receiver is illustrated in Fig. 8. A coarse carrier removal block is inserted before the GC core block as its bandwidth is limited. A post-integration processing similar to the one used in the MF architecture is also needed after the core. Combined with the coherent post-integration, the GC will perform a full correlation over range of code-phases less than the full code period. This makes it more flexible than the MF, but the flexibility comes with additional complexity. As the length of the correlated segments in the GC is much smaller than in the MF, the bandwidth of the block is larger. This allows processing signals from multiple satellites within one time-multiplexed block without issues. Also, the GC can utilize more than one input within a single block with less penalty than the MF. This allows running both tracking and acquisition in parallel.

IV. Evaluation Results

In this section, we will introduce the gate count estimation method that will be used to compare the implementations, and shortly discuss how embedded static RAM (SRAM) memories were handled. After the estimation method has been introduced, we will show the estimates for the three architectures, and present the overall winner.

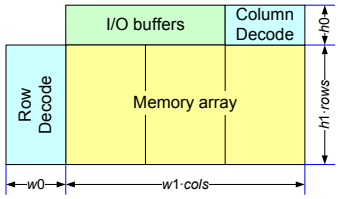


Fig. 9. Conceptual floorplan of an embedded SRAM.

TABLE IV
Memory area parameters used in the evaluation

Parameter	Value
w0	53.3 μm
w1	1.266 μm
h0	26.3 μm
h1	0.613 μm
kgates/mm^2	500

A. Area Estimation Method

The authors have presented a full description of the gate count estimation methodology in [11]. The article included gate count estimations of some GNSS receiver test cases which achieved very good accuracy when compared with actual gate counts. We will refer the reader to that article for an in-depth treatment of the method.

In the original article [11] the authors did not include the embedded SRAM memories in the gate count estimates. For this evaluation, we have included the memories in the gate counts. The area of the memory has been converted to gates according to a typical value of gates/mm^2 given by the silicon foundry. We developed a simple model for the memory area based on the conceptual SRAM floorplan illustrated in Fig. 9. The area parameters for the used low-power 65 nm CMOS technology are shown in Table IV. The "gate count" of a SRAM block is given by:

$$\text{gates} = (h0 + h1 \cdot c) \times (w0 + w1 \cdot r) \times \text{gates}/\mu\text{m}^2 \quad (2)$$

where c and r denote the rows and columns in the memory array, respectively.

B. Test Case Evaluation

The results of the gate count estimation are listed in Table V along with key implementation parameters. The columns **Blks** and **Ch** denote the number of blocks and channels, and **Freq** and **Code** mean the number of carrier frequencies and replica code phases per channel. The area is given in kilogates and is shown as a total for the whole receiver as well as divided by the number of independent channels (**Blks** $\times$ **Ch**) or by the number of total cells in the receiver. The per-channel number is an indication of how well the receiver can be configured to operate on different satellites and is an indication

TABLE V
AREA ESTIMATES OF THE RECEIVERS (IN KILO-GATES)

	Parameters				Area		
Arch	**Blks**	**Ch**	**Freq**	**Code**	**Total**	**/ch**	**/cell**
Acquisition							
Corr	21	8	8	64	4663	28	0.054
MF	1	12	4	2046	1205	100	0.012
GC	5	4	9	512	1925	96	0.021
Tracking							
Corr	1	16	1	5	53	3	0.658
MF	16	1	1	8184	5964	373	0.046
GC	1	16	1	64	125	8	0.122
A-GPS							
Corr	6	16	4	64	1425	15	0.058
MF	3	1	4	2046	703	234	0.029
GC	3	4	4	512	578	48	0.024
Worst-case							
Corr	21	8	8	64	4663	28	0.054
MF	16	1	4	8184	10698	669	0.020
GC	7	4	9	512	2692	96	0.021

of the receiver's area-efficiency in tracking modes as well as flexibility to switch between the track and acquisition modes. The per-cell number estimates the raw correlating power of the receiver and is an indication of the area-efficiency in the signal search modes. We have evaluated the receivers for four different cases. First three are the ones described earlier, and the fourth one is a worst case scenario, which defines a receiver capable of meeting all the test case criteria with a single hardware configuration. This case will test the adaptability of the different architectures for varying situations, but it is not a realistic design goal for a GPS receiver due to the overly stringent set of requirements.

As we can see from the results, the main advantage of the traditional correlator architecture is the flexibility. The receiver can be easily configured via the receiver software to adapt to varying signal conditions. The downside comes from the same flexibility and it is the relatively large area per cell. The matched filter based architecture has the best efficiency for the acquisition case. It starts to lose its efficiency when the search space can be limited such as in the A-GPS case and is the worst in the tracking case, where only a few code phases are needed. This is due to the length of the MF being equal to the spreading code length. Especially in the tracking case, when the code needs to be sampled at a higher rate, this makes the MF very large. In the worst-case example, the MF implements much more code phases than needed for the search mode. As the per-cell area was computed for all implemented code phases, it makes the MF look more effective for the search than it really is. The corrected per-cell number for the MF in the worst-case example should be four times higher as only every fourth code phase is useful. The group correlator based architecture shows great promise in the evaluation, as it achieves good overall area-efficiency in all modes. However, there are still a large number of unnecessary code phases implemented in the tracking modes.

V. CONCLUSIONS

We have presented a gate count estimate based evaluation of three GPS receiver architectures for four different use-cases. The results show that the group correlator based receiver offers good trade-off between flexibility and computational power in most of the cases. The traditional correlator is still the best choice for pure tracking mode and the matched filter offers the best raw search-power in cold-start like conditions.

In our evaluation, we did not look into the hybrid use-cases, where the receiver needs to alternate between searching for satellites and tracking them. Also, we did not look into how the receivers would work with different external aiding. Digging more deeply into these interesting real-world cases is left for future research, as the subject is too wide to be covered in this paper.

Another important dimension that is missing in this paper would be to add consideration of power consumption into the equation. The authors have scratched the subject for the correlator based receivers in [6], but there is still much more to do.

Nevertheless, we have shown in this paper that it is feasible to compare the GNSS receiver complexity at a high level mapping pre-defined use-cases to architecture realizations and comparing their suitability. This allows for the architectural level optimization of the receivers.

REFERENCES

[1] *Navstar GPS Space Segment/Navigation Users Interface. GPS Interface Specification,* IS-GPS-200F, Sep. 2011. [Online]. Available: http://www.gps.gov/technical/icwg/IS-GPS-200F.pdf

[2] F. van Diggelen and C. Abraham, "Indoor gps: The no-chip challenge," *GPS World*, vol. 12, no. 9, 2001.

[3] P. Misra and P. Enge, *Global Positioning System: Signals, Measurements and Performance*. Ganga-Jamuna Press, 2006.

[4] *3GPP2 C.S0036-0 v1.0, Recommended Minimum Performance Specification for C.S0022-0 Spread Spectrum Mobile Stations*, 2011. [Online]. Available: http://www.3gpp2.org/Public_html/specs/C.S0036-0_v1.0.pdf

[5] *3GPP TS 25.171, Requirements for support of Assisted Global Positioning System (A-GPS); Frequency Division Duplex (FDD)*, 2011. [Online]. Available: http://www.3gpp.org/ftp/Specs/html-info/25171.htm

[6] V. Eerola, "Correlator design and implementation for gnss receivers," in *NORCHIP, 2013*, Nov 2013.

[7] ——, "Rapid parallel GPS signal acquisition," in *Proc. of the 13th International Technical Meeting of the Satellite Division of The Institute of Navigation (ION GPS 2000)*, Salt Lake City, UT, Sep. 2000, pp. 810–816.

[8] V. Eerola and T. Ritoniemi, "Signal acquisition system for spread spectrum receiver," U.S. Patent 6,909,739, Jun. 21, 2005.

[9] ——, "Matched filter and spread spectrum receiver," U.S. Patent 7,010,024, Mar. 7, 2006.

[10] V. Eerola, S. Pietilä, and H. Valio, "A novel flexible correlator architecture for GNSS receivers," in *Proc. of the 2007 National Technical Meeting of The Institute of Navigation (ION NTM 2007)*, San Diego, CA, Jan. 2007, pp. 681–691.

[11] V. Eerola and J. Nurmi, "High-level parameterizable area estimation modeling for ASIC designs," *Integration, the VLSI Journal*, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S016792601400008X

This thesis was typeset in LaTeX 2_ε.

The figures were prepared with Microsoft Visio, Matlab, and Forest.

Tampereen teknillinen yliopisto
PL 527
33101 Tampere

Tampere University of Technology
P.O.B. 527
FI-33101 Tampere, Finland

ISBN 978-952-15-4217-6
ISSN 1459-2045