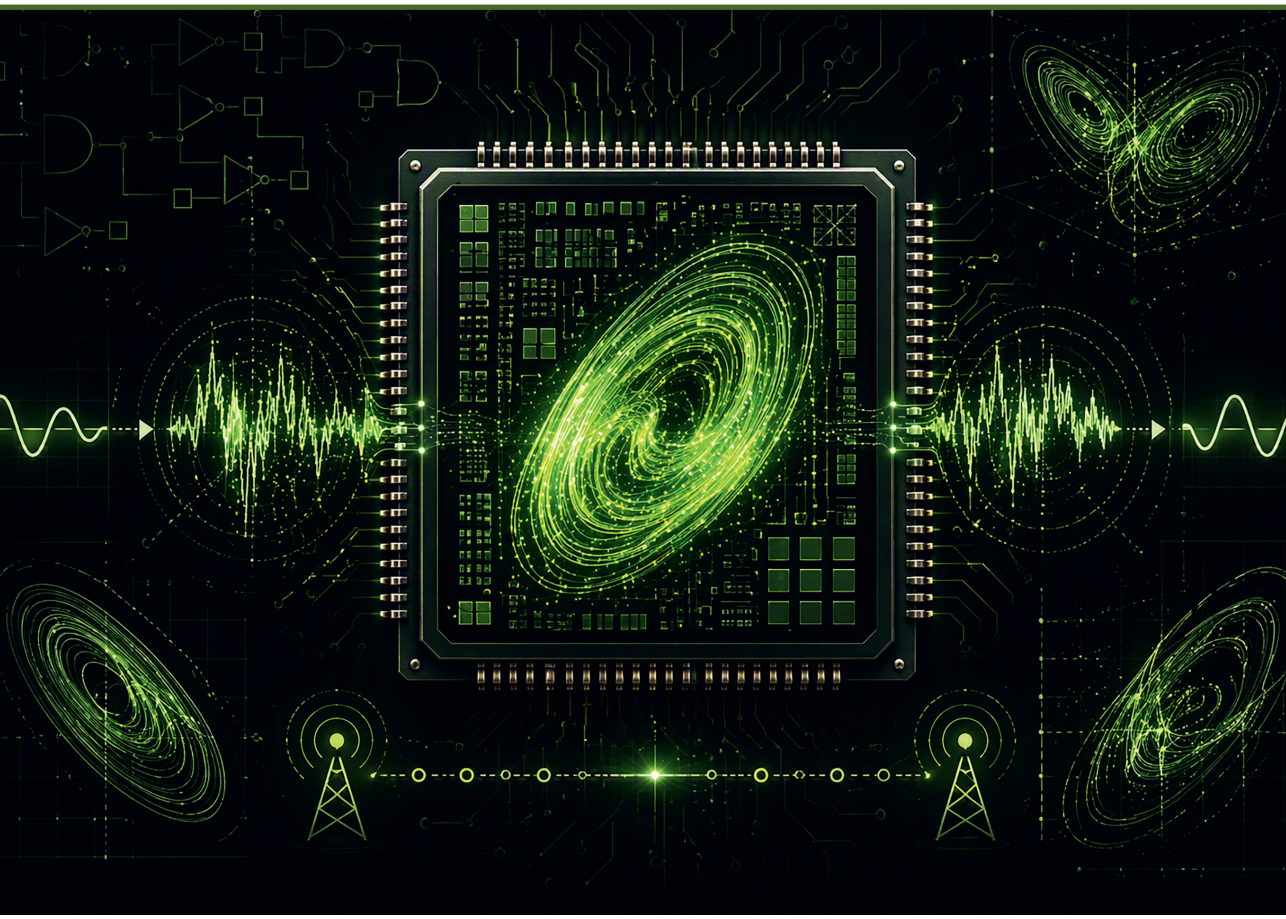
RIGA TECHNICAL UNIVERSITY

Filips Čapligins

FPGA IMPLEMENTATION OF AM-ACSK AND FM-ACSK DIGITAL CHAOTIC COMMUNICATION SYSTEMS

Doctoral Thesis



RTU Press
Riga 2026

RIGA TECHNICAL UNIVERSITY

Faculty of Computer Science, Information Technology and Energy

Institute of Photonics, Electronics and Telecommunications

Filips Čapligins

Doctoral Student of the Study Programme "Electronics"

FPGA IMPLEMENTATION OF AM-ACSK AND FM-ACSK DIGITAL CHAOTIC COMMUNICATION SYSTEMS

Doctoral Thesis

Scientific supervisors:

Professor Dr. sc. ing.

ANNA LITVIŅENKO

Associate Professor PhD

DENISS KOLOSOVS

RTU Press

Riga 2026

Čapligins, F. FPGA Implementation of AM-ACSK and FM-ACSK Digital Chaotic Communication Systems. Doctoral Thesis. – Riga: RTU Press, 2026. – 108 p.

Published in accordance with the decision of the Promotion Council “RTU P-08” of 15 May 2026, Minutes No. 52.

The research was partially supported by research and development grant No. C4835.Dok.1076 under the EU RRF project No. 5.2.1.1.i.0/2/24/I/CFLA/003.

Cover image generated using Microsoft Designer.

DOCTORAL THESIS PROPOSED TO RIGA TECHNICAL UNIVERSITY FOR PROMOTION TO THE SCIENTIFIC DEGREE OF DOCTOR OF SCIENCE

To be granted the scientific degree of Doctor of Science (PhD), the present Doctoral Thesis has been submitted for defence at the open meeting of RTU Promotion Council on 11 September 2026. at 11:00 at the Faculty of Computer Science, Information Technology and Energy of Riga Technical University, Āzenes iela 12, Room 201.

OFFICIAL REVIEWERS

Associate Professor Dr. sc. ing. Andis Supe
Riga Technical University

Professor Dr Darius Plonis
Vilnius Gediminas Technical University, Lithuania

Professor Dr. sc. ing., Dr. habil. sc. ing. Igor Kabashkin
Transport and Telecommunications Institute, Latvia

DECLARATION OF ACADEMIC INTEGRITY

I hereby declare that the Doctoral Thesis submitted for review to Riga Technical University for promotion to the scientific degree of Doctor of Science (PhD) is my own. I confirm that this Doctoral Thesis has not been submitted to any other university for promotion to a scientific degree.

Filips Čapligins _____

Date: _____

The Doctoral Thesis has been written in English. It consists of an introduction, five sections, conclusions, 56 figures, eight tables, and three appendices. The total number of pages is 96, excluding appendices. The bibliography contains 90 titles.

ABSTRACT

This Thesis explores the application of chaos theory in a digital communication systems, focusing on the design, implementation, and evaluation of two novel prototypes: amplitude-modulated antipodal chaos shift keying (AM-ACSK) and frequency-modulated antipodal chaos shift keying (FM-ACSK) to enhance physical-layer signal protection in wireless and IoT applications.

The research proposes two hybrid modulation schemes that integrate baseband antipodal chaos shift keying (ACSK) with passband amplitude modulation (AM) and frequency modulation (FM) techniques, utilizing a modified Chua chaos generator for broadband, noise-like signals and error-feedback chaotic synchronization for robust master-slave alignment. Mathematical models are formulated, including ordinary differential equation (ODE) for the chaos generator discretized via the forward Euler method with fixed-point arithmetic, ensuring chaotic behavior suitable for modulation and FPGA implementation.

The work details the field-programmable gate array (FPGA)-based transceiver prototypes, incorporating signal processing chains: ACSK modulation with 8192-sample symbol durations and coherent detection based on chaotic synchronization, NCO-driven AM/FM modulators with 10.7 MHz passband carrier, digital incoherent AM/FM demodulators, filtering to 215 kHz cutoff for noise rejection, early-late gate symbol timing recovery, and sign-based bit detection via integration.

Complementary Simulink models validate the digital components, enabling comparisons with hardware outcomes and revealing minor discrepancies attributable to fixed-point arithmetic and real-world channel effects. Performance is evaluated through bit error rate (BER) analyses in additive white Gaussian noise (AWGN) and selective fading channels across varying signal-to-noise ratio (SNR), demonstrating FM-ACSK's superior noise immunity due to constant envelope and broader spectrum bandwidth employed.

The proposed systems address challenges in chaotic communications, such as synchronization reliability and noise vulnerability, while offering innovative modulation schemes, FPGA-optimized implementations, and empirical insights into chaotic systems' practical viability, paving the way for advancing chaotic communications in resource-constrained environments.

ANOTĀCIJA

Promocijas darbā pētīta haosa teorijas lietošana digitālās sakaru sistēmās, koncentrējoties uz divu jaunu prototipu izstrādi, ieviešanu un novērtēšanu: amplitūdas modulētas antipodālās haotiskās manipulācijas (*AM-ACSK*) un frekvences modulētas antipodālās haotiskās manipulācijas (*FM-ACSK*) haotiskām sakaru sistēmām, lai uzlabotu fiziskā slāņa signāla aizsardzību bezvadu un *IoT* pielietojumos.

Pētījumā ir izvirzītas divas hibrīdas modulācijas shēmas, kas integrē pamatjoslas antipodālo haotisko manipulāciju (*ACSK*) ar *AM* un *FM* joslas pārneses tehnikām, izmantojot modificētu Čua haosa ģeneratoru platjoslas, trokšņveidīgam signālam un kļūdu atgriezeniskās saites haotisko sinhronizāciju, lai nodrošinātu stabilu vadošā un vadāmā ģeneratoru sinhronizāciju. Ir izstrādāti matemātiskie modeļi, tostarp parastie diferenciālvienādojumi (*ODE*) haosa ģeneratoram, kas diskretizēti, izmantojot Eilera metodi ar fiksēta punkta aritmētiku, nodrošinot haotisku uzvedību, piemērotu modulācijai un *FPGA* realizācijai.

Darbs detalizēti apraksta uz *FPGA* bāzes izveidotus raiduztvērēju prototipus, kas ietver signālapstrādes ķēdes: *ACSK* modulācija ar 8192 takts nolašu ilgu simbolu un koherentā detekcija, kas balstīta uz haotisku sinhronizāciju, *NCO* vadīti *AM*/*FM* modulatori ar 10,7 MHz nesējfrekvenci, digitāli nekoherenti *AM*/*FM* demodulatori, filtrēšana līdz 215 kHz trokšņu novēršanai, "*early-late gate*" simbolu laika sinhronizācija un bitu detektēšana ar integrāciju.

Papildus izstrādātie *Simulink* modeļi ļauj pārbaudīt digitālo komponentu darbību, salīdzināt tos ar aparatūras rezultātiem un atklāt nelielas neatbilstības, kas saistītas ar fiksētā punkta aritmētiku un reālos apstākļos novērojamajiem kanāla efektiem. Veiktspēja tiek novērtēta, analizējot bitu kļūdu intensitāti (*BER*) aditīvā baltajā Gausa troksnī (*AWGN*) un selektīvās vājināšanas kanālā ar dažādām signāla un trokšņa attiecībām (*SNR*), demonstrējot *FM-ACSK* pārāku trokšņu noturību, pateicoties vienmērīgai signāla apliecējai un plašākam spektra joslas platumam.

Piedāvātās sistēmas vēršas pie drošu haotisko sakaru problēmām, piemēram, sinhronizācijas noturību un jutību pret troksni, vienlaikus piedāvājot inovatīvas modulācijas shēmas, *FPGA* pielāgotas realizācijas un empīriskas atziņas par haotisku sistēmu praktisko dzīvotspēju, pavērojot ceļu haotisko sakaru uzlabojumu iespējām vidēs ar ierobežotiem resursierm.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ACSK antipodal chaos shift keying

ADC analog-to-digital converter

AGC automatic gain control

ALUT adaptive lookup table

AM amplitude modulation

AM-ACSK amplitude-modulated antipodal chaos shift keying

ASIC application-specific integrated circuit

AWGN additive white Gaussian noise

BER bit error rate

C-PSK chaotic phase shift keying

CDMA code division multiple access

CDSK correlation delay shift keying

CIC cascaded integrator-comb

CLB configurable logic block

COOK chaos on-off keying

CPSK chaos parameter shift keying

CPU central processing unit

CSK chaos shift keying

CSS chaos-based selective symbol

DAC digital-to-analog converter

DCSK differential chaos shift keying

DMA direct memory access

DSP digital signal processing

FEC forward error correction

FIFO first input first output

FIR finite impulse response

FM frequency modulation

FM-ACSK frequency-modulated antipodal chaos shift keying

FM-DCSK frequency-modulated differential chaos shift keying

FPGA field-programmable gate array

HDL hardware description language

HE-DCSK high-efficiency differential chaos shift keying

HPS hard processing system

HSMC high speed mezzanine card

IC integrated circuit

IIR infinite impulse response

IoT Internet of Things

IP intellectual property

LED light-emitting diode

LFSR linear feedback shift register

LSR linear shift register

LUT lookup table

M-DCSK M-ary differential chaos shift keying

MPU microprocessor unit

MSB most significant bit

NCO numerically controlled oscillator

NR-DCSK noise reduction differential chaos shift keying

ODE ordinary differential equation

OPCL open-plus-closed-loop

PC personal computer

PTCFZNN preset-time convergence fuzzy zeroing neural network

QCSK quadrature chaos shift keying

RCO reconfigurable chaotic oscillator

RK-4 fourth-order Runge-Kutta

RM-DCSK reference-modulated differential chaos shift keying

RMS root mean square

SDRAM synchronous dynamic random-access memory

SFDR spurious-free dynamic range

SNR signal-to-noise ratio

SOB straight offset binary

SoC system-on-chip

TED timing error detector

UART universal asynchronous receiver/transmitter

USB universal serial bus

VHDL very high speed integrated circuit hardware description language

WCDMA wideband code division multiple access

XOR exclusive OR

INTRODUCTION

Relevance

In recent decades, the necessity for communication infrastructures with increased signal protection has grown dramatically. Telecommunications play a pivotal role in fostering worldwide connectivity, and the expansion of the Internet of Things (IoT) has further augmented this need. In the context of escalating cyber vulnerabilities and privacy risks, reliable data transmission has emerged as a paramount priority. Conventional security strategies are predominantly reliant on encryption algorithms, which are susceptible to obsolescence due to advancements in computing technology, such as quantum mechanics. These methods primarily prioritize the safeguarding of upper-layer protocols, frequently overlooking the protective measures necessary for the physical transmission layer. Chaotic communication systems present a compelling alternative, taking advantage of the nonlinear and unpredictable nature of chaotic waveforms to enhance protection and deter unauthorized interception.

Research into the application of chaos generators for communication systems dates back to the 1990s, with one of their primary benefits being enhanced physical-layer signal protection [1]. Utilizing chaotic oscillations as carriers renders signal detection and information extraction highly challenging due to the inherently unpredictable signal traits. The incorporation of chaotic synchronization into coherent detection schemes serves to further amplify this protection, thereby necessitating an identical chaos generator at the receiver – with matching parameters and architecture – to achieve demodulation [2]. Furthermore, chaotic signals exhibit robust resilience to interception, demonstrating notable tolerance to multipath effects and deliberate jamming [3]. The inherent nonlinear, ergodic, and pseudorandom characteristics of these systems foster unpredictability, thereby complicating unauthorized access without exact system knowledge. When considered in junction with narrow autocorrelation, reduced cross-signal correlations, and broad spectral spread, these attributes position chaotic systems as a viable option for bolstering physical-layer defenses in wireless transmissions, rendering them a valuable subject for scholarly exploration and practical signal protection applications.

The utilization of chaos generators has historically been confined to the domain of analog circuits. However, the implementation of analog chaos generators within communication systems poses significant challenges, including the absence of initial state control and the inherent parametric instability of components. This results in the complexity of drift compensation and the unreliability of chaotic synchronization [4]. These disadvantages have prompted a shift towards the discrete digital implementations on FPGAs to eliminate drift issues and enable deterministic, high-throughput parallel signal processing via pipelining [3]. In the contemporary context, the reprogrammable nature and hardware description language support of FPGAs have led to a resurgence of interest in chaotic communication application studies [5], [6]. However, despite this growing interest, the practical

implementation of complete chaotic communication transceivers on FPGAs and their experimental performance evaluation remain comparatively limited in the existing literature. Furthermore, there is an ongoing need to optimize resource allocation in order to maintain chaos fidelity while meeting efficiency requirements.

Research Goal and Tasks

The goal of this Thesis is to research the coherent discrete chaotic communication system design options and to develop a set of approaches for the FPGA implementation of such systems. In order to accomplish the objective, the following tasks have been established.

- Analytical study on chaotic communication and FPGA implementation of chaotic systems.
- Design mathematical models for coherent discrete AM-ACSK and FM-ACSK chaotic communication systems.
- Perform the proof-of-concept verification via simulations.
- Development of the FPGA prototypes for the designed AM-ACSK and FM-ACSK chaotic communication systems with VHDL code.
- Perform the experimental verifications of the FPGA prototypes.

Scientific Novelty and Main Results

The study's primary results and scientific novelties are as follows.

- The original discrete adaptation of a modified Chua circuit chaos generator, error-feedback synchronization, and AM-ACSK alongside FM-ACSK modulated communication systems designed for FPGA hardware integration.
- The development of a Simulink mathematical model that precisely reflects the digital components of the proposed AM-ACSK and FM-ACSK communication system prototypes.
- The structural design and FPGA implementation of novel AM-ACSK and FM-ACSK discrete chaotic communication system transceivers, as well as their practical validation through operational tests, have confirmed the viability of both systems and advanced the conceptual development of chaotic communication technologies.

The following components are of particular significance.

- Transmitters: Modified Chua chaos generator (ODE discretized with forward Euler and fixed-point arithmetic), ACSK modulation (8192-sample symbols), NCO-based AM/FM 10.7 MHz passband.

- Receivers: Signal processing including incoherent digital AM/FM demodulation, noise filters, AGC (for AM-ACSK only), coherent ACSK demodulation with error-feedback chaotic synchronization and symbol timing recovery.

The implementation process involves the use of the very high speed integrated circuit hardware description language (VHDL) for prototyping on Altera Cyclone V FPGA with a clock frequency of 50 MHz. This process encompasses the following auxiliary modules, which are necessary for the performance evaluation tests:

- an LFSR is utilized for the purpose of generating pseudorandom transmitted data;
- the AWGN controller is a real-time system that adjusts noise levels;
- the BER controller is utilized for the purpose of detecting and counting bit errors;
- the BER-to-DMA module serves the purpose of transmitting BER data to the memory;
- the software program has been developed in the C programming language for the purpose of reading BER data from memory and displaying the results;
- DAC and ADC peripheral modules facilitate analog channel link interfacing between transmitter and receiver.

The performance of the system was evaluated via BER in AWGN (and selective fading for FM-ACSK) channels over a range of SNR. The emphasis was placed on noise immunity, synchronization reliability, simulation-hardware gaps (fixed-point number format and real-world effects), chaotic communication challenges, and practical viability.

The Theses to be Defended

1. A discrete **AM-ACSK** transceiver based on a modified Chua oscillator employing the forward Euler discrete ODE solving method and error linear feedback chaotic synchronization **is implementable on a single Altera/Intel Cyclone V FPGA** with the data rate of 6.1 kbps at **430 kHz channel bandwidth**.
2. A discrete **FM-ACSK** transceiver based on a modified Chua oscillator employing the forward Euler discrete ODE solving method and error linear feedback chaotic synchronization **is implementable on a single Altera/Intel Cyclone V FPGA** with the data rate of 6.1 kbps at **1.720 MHz channel bandwidth** and **FM modulation index 3.**
3. The discrete **AM-ACSK** transceiver based on a modified Chua oscillator employing the forward Euler discrete ODE solving method, error linear feedback chaotic synchronization and implemented on a single Altera/Intel Cyclone V FPGA **achieves AWGN performance of 10^{-3} BER at 8.7 dB SNR** with an uncoded data rate of 6.1 kbps at **430 kHz channel bandwidth**.

4. The discrete **FM-ACSK** transceiver based on a modified Chua oscillator employing the forward Euler discrete ODE solving method, error linear feedback chaotic synchronization and implemented on a single Altera/Intel Cyclone V FPGA **achieves AWGN performance of 10^{-3} BER at 5.6 dB SNR** with an uncoded data rate of 6.1 kbps at **1.720 MHz channel bandwidth** and **FM modulation index 3**.

Research Methodology

In order to achieve the established goals and tasks, the research methodology incorporates a multi-stage approach integrating theoretical analysis, modeling, simulation, hardware implementation, and empirical validation. This structured progression ensures a comprehensive exploration of chaotic communication systems, from conceptual design to practical deployment on FPGA platforms.

The foundation of the research begins with analytic methods, including a literature review and state-of-the-art analysis on chaotic communication systems. This involves examining historical contexts, properties of chaos, chaos oscillators, synchronization techniques, modulation schemes, and FPGA implementations as alternatives to analog circuits. The review emphasizes FPGA-based chaos generators and communication systems, identifying challenges (e.g., synchronization reliability, noise vulnerability) and metrics (e.g., ordinary differential equation (ODE) discretization, clock speeds, bit precision, resource use), drawing from scholarly sources to support the novelty of the proposed AM-ACSK and FM-ACSK schemes.

Utilizing this analysis as a foundation, mathematical modeling is employed to design the AM-ACSK and FM-ACSK systems. ODEs for the modified Chua chaos generator are formulated and discretized using the forward Euler method with fixed-point arithmetic to ensure chaotic behavior suitable for digital implementation. The models under consideration incorporate error feedback synchronization, ACSK modulation, and digital AM/FM passband techniques via numerically controlled oscillators (NCOs). Additionally, receiver signal processing modules are employed, including filtering, timing recovery using the early-late gate method, and bit detection via integration.

Numerical methods and simulations are employed to verify the proof-of-concept and assess system capabilities. Simulink models are designed to replicate the digital components of FPGA prototypes, thereby enabling performance evaluations in simulated channels. A BER analysis was conducted in an AWGN channel to compare AM-ACSK and FM-ACSK. The results demonstrated that FM-ACSK exhibits superior noise immunity. FM-ACSK's performance was further evaluated in a selective fading model.

The subsequent stage involves practical experiments, with a focus on FPGA prototype development and empirical verification. The implementation of the systems is achieved through the utilization of VHDL code compilation, facilitated by Quartus software for Altera Cyclone V FPGA hardware. This implementation incorporates transmitter and receiver signal processing chains, in

addition to auxiliary modules that are employed for experimental setup, including AWGN noise controller, BER measurement tools, and reading software. Operational trials entail the implementation of hardware-in-the-loop testing, wherein digital-to-analog converter (DAC) and analog-to-digital converter (ADC) are connected via coaxial cable to establish an analog channel with 10.7 MHz passband frequency. BER measurements are conducted for various AWGN levels and selective fading scenarios, thereby validating simulation results and confirming the prototypes' viability. This methodological approach guarantees progression from theory to practice, balancing innovation with rigorous verification.

The Object of the Research

The object of the present research is a discrete chaotic communication system. Two such novel systems have been designed with AM-ACSK and FM-ACSK modulations. The research focuses on the FPGA implementation and performance evaluation of these systems.

Practical Significance

The practical importance of this research lies in the advancement of chaotic communication technologies through the development and validation of novel AM-ACSK and FM-ACSK FPGA-implemented prototypes, addressing real-world challenges in physical-layer signal protection and noise resilience. It is noteworthy that the implementation of discrete chaotic communication systems using FPGA is a subject that has received significantly less attention than other areas of communication system research, as demonstrated by a literature analysis. Complementary Simulink models facilitate rapid design and verification, while auxiliary experimental setup modules for the FPGA prototypes support empirical testing of the system noise immunity. These advancements serve as a practical guide for building and FPGA-implementing coherent chaotic communication systems, with several approaches, algorithms, and performance metrics to aid the design process.

Potential applications include:

- secure IoT networks, where the use of noise-like chaotic signals can enhance privacy and anti-jamming in resource-constrained devices;
- wireless military communications, offering covert transmission possibility and relative robustness to interference;
- spread-spectrum systems in the healthcare and transportation infrastructure sectors, which would enhance data integrity and protect against unauthorized access;
- commercial FPGA-based modems, which would enable integration into standards-compliant devices that have low data rate demands;

- smart home local sensor networks or wide area sensor networks, where such infrastructure might benefit from enhanced data protection options.

The transceiver prototypes, developed through iterative simulations and hardware trials, demonstrate the conceptual feasibility of the FPGA implementation, which can be used for further investigation of chaotic communication systems. The proposed design can be adapted to meet the specific application requirements in terms of data rates, spectrum bandwidth, resource and energy usage, and other relevant parameters.

Approbation

The following papers are relevant to the Thesis and have been published in scientific journals (the most relevant ones that reflect the main ideas of the research are marked in bold).

1. **F. Capligins et al. "Frequency-Modulated Antipodal Chaos Shift Keying Chaotic Communication on Field Program Gate Array: Prototype Design and Performance Insights." en. In: *Applied Sciences* 15.3 (Jan. 2025), p. 1156. ISSN: 2076-3417. DOI: 10.3390/app15031156**
2. M. F. Taşdemir et al. "Performance Evaluation of FPGA-Based Design of Modified Chua Oscillator." en. In: *Chaos Theory and Applications* 6.4 (Nov. 2024), pp. 249–256. ISSN: 2687-4539. DOI: 10.51537/chaos.1466919
3. D. Cirjulina et al. "Experimental Study on Colpitts Chaotic Oscillator-Based Communication System Application for the Internet of Things." en. In: *Applied Sciences* 14.3 (Jan. 2024), p. 1180. ISSN: 2076-3417. DOI: 10.3390/app14031180
4. R. Babajans et al. "Synchronization of Analog-Discrete Chaotic Systems for Wireless Sensor Network Design." en. In: *Applied Sciences* 14.2 (Jan. 2024), p. 915. ISSN: 2076-3417. DOI: 10.3390/app14020915
5. R. Babajans et al. "Performance Analysis of Vilnius Chaos Oscillator-Based Digital Data Transmission Systems for IoT." en. In: *Electronics* 12.3 (Jan. 2023), p. 709. ISSN: 2079-9292. DOI: 10.3390/electronics12030709
6. **F. Capligins et al. "FPGA-Based Antipodal Chaotic Shift Keying Communication System." en. In: *Electronics* 11.12 (June 2022), p. 1870. ISSN: 2079-9292. DOI: 10.3390/electronics11121870**
7. F. Capligins et al. "Experimental Study of the Chaotic Jerk Circuit Application for Chaos Shift Keying." In: *Latvian Journal of Physics and Technical Sciences* 58.4 (2021), pp. 55–68. DOI: 10.2478/lpts-2021-0033

The following papers are relevant to the Thesis and have been presented in scientific conferences and published in proceedings (the most relevant ones that reflect the main ideas of the research are marked in bold).

1. **F. Capligins, A. Litvinenko, and D. Kolosovs. "Selective Fading Channel Performance Evaluation for Frequency-Modulated Antipodal Chaos Shift Keying Communication System." In: *2025 IEEE***

***12th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*. 2025, pp. 1–5. DOI:** 10.1109/AIEEE66149.2025.11050770

2. D. Cirjulina et al. "Fundamental Frequency Impact on Vilnius Chaos Oscillator Dynamics." In: *16th Chaotic Modeling and Simulation International Conference*. Ed. by C. H. Skiadas and Y. Dimotikalis. Cham: Springer Nature Switzerland, 2024, pp. 87–101. ISBN: 978-3-031-60907-7. DOI: 10.1007/978-3-031-60907-7_8
3. R. Babajans et al. "Study on Analog and Discrete Chaos Oscillators Synchronization." In: *16th Chaotic Modeling and Simulation International Conference*. Ed. by C. H. Skiadas and Y. Dimotikalis. Cham: Springer Nature Switzerland, 2024, pp. 33–46. ISBN: 978-3-031-60907-7. DOI: 10.1007/978-3-031-60907-7_4
4. **F. Capligins, A. Litvinenko, and D. Kolosovs. "Performance Evaluation of Frequency Modulated Antipodal Chaos Shift Keying Digital Communication System." en. In: *2023 Workshop on Microwave Theory and Technology in Wireless Communications (MTTW)*. Riga, Latvia: IEEE, Oct. 2023, pp. 29–33. ISBN: 9798350393491. DOI:** 10.1109/MTTW59774.2023.10320019
5. A. Aboltins et al. "Implementation of chaotic frequency modulation based spread spectrum communication system in software-defined radio." In: *2023 IEEE Wireless Communications and Networking Conference (WCNC)*. 2023, pp. 1–6. DOI: 10.1109/WCNC55385.2023.10118612
6. F. Capligins, A. Litvinenko, and D. Kolosovs. "FPGA Implementation and Study of Antipodal Chaos Shift Keying Communication System." In: *2021 IEEE Microwave Theory and Techniques in Wireless Communications (MTTW)* (2021), pp. 1–6. DOI: 10.1109/mttw53539.2021.9607226
7. F. Capligins et al. "FPGA Implementation and Study of Synchronization of Modified Chua's Circuit-Based Chaotic Oscillator for High-Speed Secure Communications." In: 2021. ISBN: 978-1-66542-538-4. DOI: 10.1109/AIEEE51419.2021.9435783

Scientific conferences, where author participated with reports and discussions about the research topics of present Thesis.

- Riga Technical University's 66th International Scientific Conference (Section "Photonics, Electronics, and Telecommunications"), Riga, Latvia, November 27, 2025.
- 2025 IEEE 12th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE), Lithuania, Vilnius, 15.-17. May, 2025.
- Riga Technical University's 65th International Scientific Conference (Section "Photonics, Electronics, and Electronic Communications"), Riga, Latvia, October 11, 2024.
- International Conference on Days of Applied NOnlinearity and Complexity (DANOC), Online, January 12-14, 2024.
- Riga Technical University's 64th International Scientific Conference, Riga, Latvia, October 6, 2023.
- 2023 Workshop on Microwave Theory and Technology in Wireless Communications (MTTW 2023): Proceedings, Latvia, Riga, 4-6 October, 2023.
- Riga Technical University's 63rd International Scientific Conference, Riga, Latvia, October 7, 2022.

- 2022 Workshop on Microwave Theory and Technology in Wireless Communications (MTTW 2022): Proceedings, Latvia, Riga, 5-7 October, 2022.
- 2021 IEEE Microwave Theory and Techniques in Wireless Communications (MTTW 2021), Latvia, Riga, 7-8 October, 2021.
- 2020 IEEE 8th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE 2020), Lithuania, Vilnius, 22-24 April, 2021.

Structure of the Thesis

The present Thesis is structured into five main sections, followed by overall conclusions.

Section 1 provides an overview of chaos applications in communication systems, covering historical background, properties of chaos, oscillators, synchronization methods, modulation schemes (non-coherent and coherent), advantages of FPGA over analog implementations, a review of existing FPGA-based chaotic systems, key challenges, and an introduction to the AM-ACSK/FM-ACSK research.

Section 2 provides a comprehensive overview of the AM-ACSK system, including its mathematical model, FPGA implementation considerations, transmitter/receiver signal processing, and Simulink model setup.

Section 3 elaborates on the FM-ACSK system, emphasizing architectural modifications from AM-ACSK, design principles, and FPGA implementation.

Section 4 evaluates the performance of both systems via Simulink simulations, by reviewing BER results for AM-ACSK (AWGN) and FM-ACSK (AWGN and fading).

Section 5 assesses the performance of FPGA prototypes, including experimental setups with auxiliary modules, BER analyses in AWGN (both systems) and selective fading (FM-ACSK).

Finally, the conclusions offer a summary of the key findings, innovations, and implications for chaotic communications.

1 CHAOS APPLICATIONS IN COMMUNICATION SYSTEMS

This section provides an overview of using chaos in communication systems and offers some general examples. It also explores the state of the art for FPGA implementation of chaos generators and chaotic communication systems.

1.1 General Context and Historical Background

Chaos manifests as aperiodic long-term behavior in a deterministic system, exhibiting a sensitive dependence on the initial conditions [3]. The use of chaotic oscillations in communication systems is not a novel technique, and it possesses certain advantages. A chaotic signal is very difficult to intercept, and it has good resistance to multipath propagation and purposefully created interference [2].

Chaotic systems possess certain characteristics that make them particularly well-suited for use in communication systems.

- **Reduced energy utilization.** Circuits capable of chaotic signal generation can function efficiently with minimal power consumption.
- **Noise-like appearance.** The output signal of a chaotic system is aperiodic and, in certain cases, can be indistinguishable from noise, which complicates the detection of a chaotic signal.
- **Broadband spectrum.** The autocorrelation function of a chaotic system demonstrates a rapid decrease in value as the time interval increases. This property is intrinsic to the nature of a broadband spectrum signal. Furthermore, the cross-correlation function for chaotic signals of the same system from different initial conditions is found to be minimal due to the system's sensitive dependence on the initial conditions. These properties render chaotic generators highly suitable for implementation in spread spectrum communication systems.
- **Self-synchronization.** This property, which will be explained further, allows the receiver to synchronize with the transmitter's state. This can enhance the communication system's resilience to signal interception by utilizing coherent detection techniques.

Chaotic signals, characterized by their inherent wide-band nature, are well-suited for the propagation of narrowband information. Consequently, the utilization of chaotic signals for information encoding results in the generation of spread-spectrum signals characterized by increased bandwidths and reduced power spectral densities. The advantages associated with spread-spectrum signals, including the complexity of uninformed detection, the mitigation of multipath fading, and anti-jamming capabilities, are all features that are inherent to this technology. Therefore, chaos emerges as a cost-effective and adaptable medium for spread-spectrum communications.

The concept of chaos had been recognized by the 1890s, a period during which Poincaré endeavored to resolve the three-body problem. He demonstrated that the orbits of bodies are aperiodic, do not increase infinitely, and do not tend to a fixed point or orbit. The complexity inherent in resolving the three-body problem stems from its sensitivity to even minor alterations in the initial conditions, thereby precluding the capacity for reliable long-term predictions. Therefore, Poincaré can be regarded as one of the pioneers in the field of chaos theory [3].

In 1963, E. Lorenz of the Massachusetts Institute of Technology developed a system of three differential equations (hereafter referred to as Lorenz's equations) for modeling atmospheric convection [21]. Lorenz utilized computer simulations to solve these equations. Lorenz employed 6-digit precision for calculations and 3-digit precision for the output of results and the input of modeling data. The rounding of the initial conditions in the modeling process resulted in substantial deviations in the long-term predictions. The simulation results did not converge to a fixed point or a periodic orbit but rather fluctuated irregularly. Lorenz demonstrated the results as three-dimensional images of state variables in phase space, where the set of states of the system formed a butterfly-shaped trajectory, now known as a strange attractor (see Fig. 1.1). Lorenz's findings led to the conclusion that the weather can be regarded as a chaotic system, thereby making long-term accurate predictions impossible.

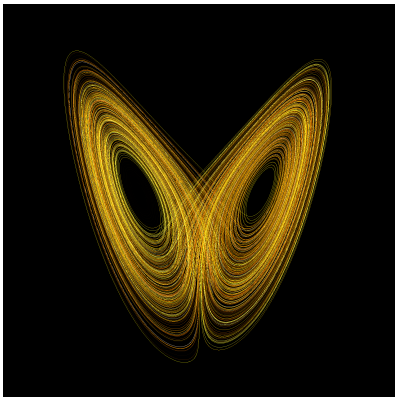


Fig. 1.1. Lorenz system strange attractor.

1.2 Properties of Chaos and Chaotic Communication Systems

Given the variation in quality, safety level, and application possibilities, communication systems can be developed for both analog and digital transmission based on chaotic generators. The study and utilization of such systems is driven by numerous factors, including the following advantages inherent to chaos-based communications.

Firstly, it is important to note that chaos and noise can be difficult to distinguish from one another. It is possible to generate a chaotic sequence that exhibits a spectrum, statistical characteristics, and

autocorrelation function that are very similar to the corresponding parameters of a stochastic noise. This, in turn, facilitates the transmission of a specified chaotic signal, the presence of which is challenging to discern by an external observer. In such cases, covert transmission of messages becomes a viable option, which can be of significant importance, particularly in military applications.

Secondly, if chaotic synchronization is employed, specific hardware is required, with the right parameters, to be able to decode the transmitted message. This complicates the unauthorized interception of the message, increasing the level of transmission protection.

Thirdly, the potential for information obfuscation is a notable consideration. In the conventional communication systems, encryption protocols are utilized, where both the transmitter and the receiver maintain a shared secret key. This key is essential for the decoding of transmitted messages. In the context of chaotic communication systems, the transmitting system itself functions as a dynamic key. The receiver must be capable of synchronizing to the dynamics of the transmitter to successfully retrieve the message.

Finally, numerous chaotic communication systems exhibit notable robustness against interference caused by multipath propagation. The importance of transmission privacy and protection in communications infrastructure is critical, as unauthorized access to such systems could compromise public order and security.

1.3 Chaos Oscillators, Chaotic Synchronization and Modulation

This subsection provides a brief overview of several examples of chaos oscillators, chaotic synchronization techniques, and modulation methods employed in chaotic communication systems with both non-coherent and coherent detection.

1.3.1 Chaos Oscillators

Chaos generators are nonlinear dynamic systems that exhibit chaotic behavior at specific parameters and initial conditions. Such systems can be described by a mathematical model using differential equations of state variables, and such systems can also be realized in a physical model (mechanical, electronic, or other). Chaotic oscillations are a possibility in a nonlinear dynamic autonomous system, defined as one that functions without external excitation and contains a minimum of three state variables [3].

A paradigmatic example of an electronic chaotic generator is the Chua circuit [22], a simple and widely studied autonomous chaotic nonlinear dynamical system for which both a physical and mathematical model are known. The concept was introduced in 1983 by Professor Leon Chua of the University of California.

As illustrated in Fig. 1.2, Chua's circuit comprises two capacitors, one coil, one resistor, and a non-linear element known as a Chua's diode. A Chua's diode is an active element characterized by

negative resistance (see Fig. 1.3).

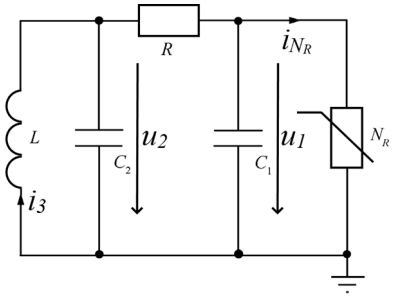


Fig. 1.2. Chua's circuit.

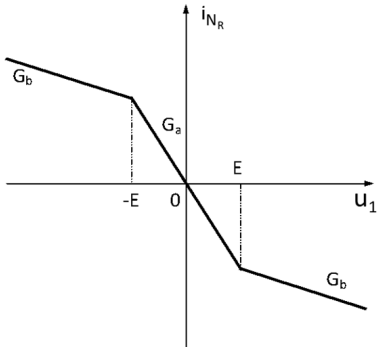


Fig. 1.3. Characteristic curve of Chua's diode.

Using Kirchhoff's and Ohm's laws, the state variable equations for the Chua's circuit can be obtained:

$$\begin{aligned} C_1\frac{du_1}{dt} &= \frac{1}{R}(u_2 - u_1) - f(u_1) \\ C_2\frac{du_2}{dt} &= \frac{1}{R}(u_1 - u_2) + i_3 \\ L\frac{di_3}{dt} &= -u_2 \end{aligned} \tag{1.1}$$

where C_1, C_2 – capacitance of the capacitors, R – resistance of the resistor, L – inductance of the coil, u_1 – voltage on the capacitor C_1, u_2 – voltage on the capacitor C_2, i_3 – current through the coil L, $f(u_1)$ – nonlinear function that describes Chua's diode:

$$f(u_1) = i_{N_R} = G_b u_1 + \frac{1}{2}(G_a - G_b)[|u_1 + E| - |u_1 - E|] \tag{1.2}$$

where i_{N_R} – current through a Chua's diode, G_a and G_b – growth factors of the linear segments, $u_1 = E$ and $u_1 = -E$ – characteristic curve break points. System (1.1) can be conveniently written with dimensionless variables:

$$\begin{aligned}\frac{dx}{dt} &= \alpha(y - x - f(x)) \\ \frac{dy}{dt} &= x - y + z, \\ \frac{dz}{dt} &= \beta y\end{aligned} \tag{1.3}$$

where the nonlinear characteristic curve function is:

$$f(x) = bx + \frac{1}{2}(a - b)(|x + 1| - |x - 1|) \tag{1.4}$$

and dimensionless quantities are:

$$\begin{aligned} x &= \frac{u_1}{E}, \quad y = \frac{u_2}{E}, \quad z = i_3\frac{R}{E} \\ \alpha &= \frac{C_2}{C_1}, \quad \beta = \frac{R^2 C_2}{L} \\ a &= RG_a, \quad b = RG_b, \quad \tau = \frac{t}{[RC_2]} \end{aligned} \tag{1.5}$$

Furthermore, the operation of the chaos generator at defined parameters can be illustrated by an attractor – a compact subset of the phase space of the dynamical system, to which the trajectories of the system tend from a wide range of initial conditions as time tends to infinity. In other words, an attractor representation can be obtained if the axes of the graph represent the values of the system variables and their changes over time. As an example, Fig. 1.4 shows a Chua's circuit attractor with a characteristic double spiral shape.

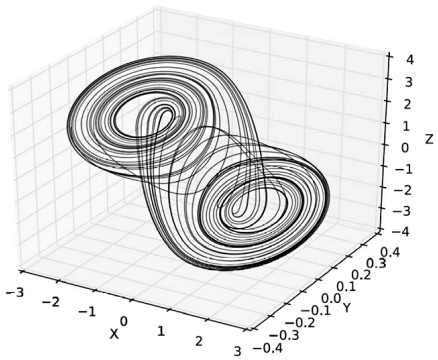


Fig. 1.4. Chua's circuit attractor.

Other well-known continuous-time chaos oscillators include Rossler [23], Colpitt [24], Chen [25], Lu [26], and others. Often, novel chaos oscillators are proposed, designed for specific applications [27]–[32].

1.3.2 Chaotic Synchronization Methods

A fundamental property of chaotic nonlinear systems is their high sensitivity to initial conditions. This sensitivity implies that two identical chaos generators, even with minor variations in starting conditions, will generate distinct chaotic outputs. However, when these generators are linked in a specific configuration, their state variables align along identical paths, resulting in chaotic synchronization. The outcome of this process is the alignment of the state variables of the two chaos generators, despite their initial conditions being distinct.

Synchronization is only possible when systems are coupled in a specific manner. Intensive research in the field of chaos communication systems commenced in 1990, when a robust solution for synchronization of chaos generators was proposed [33]. Theoretical and experimental evidence has demonstrated the feasibility of synchronizing two chaotic signals through the implementation of a slave-master configuration within the chaotic system [34].

The prevailing approach to chaotic synchronization is the Pecora-Carroll method [33], [35], which involves substituting the slave system's state variable with a variable received from the master system. Another prevalent technique is error feedback synchronization [25]. The instantaneous difference between the signal received from the master system and the signal from the slave system (which is composed using the same function as in the transmitter) produces an error signal. Through the feedback loop, this error signal modifies the state of the receiver to minimize the error. As demonstrated in [36], linear error feedback synchronization has been shown to yield superior performance in comparison to Pecora-Carroll synchronization. In instances where such methods are deemed inadequate, there exists the option of implementing more advanced and adaptive synchronization techniques. These techniques involve the real-time estimation of synchronization error and the subsequent application of that error as controlled feedback [28], [29], [37]–[41].

1.3.3 Chaotic Modulation Scheme Types

Chaotic communication systems operate on principles analogous to spread-spectrum techniques, wherein the transmitter expands the information signal's bandwidth into a broader transmission range, and the receiver compresses it during detection and demodulation. Despite the increased intricacy of these systems relative to their narrowband counterparts, they exhibit superior tolerance to multipath effects and allow for efficient multi-user frequency sharing. A critical constraint that must be addressed is the necessity for precise synchronization between identical chaotic oscillators in the transmitter and receiver. This synchronization requirement necessitates a high degree of precision to ensure optimal performance.

In essence, these systems can be categorized into two distinct classes: coherent and non-coherent. Each of these categories exhibits its own unique set of trade-offs [42]. Coherent systems, exemplified by chaos shift keying (CSK) [43], leverage chaotic synchronization to replicate the transmitter's oscillator state at the receiver, enhancing data safety through this master-slave cou-

pling. However, this is achieved at the expense of reduced noise immunity due to sensitivity to parameter mismatches and channel distortions. Conversely, non-coherent systems, exemplified by differential chaos shift keying (DCSK) [44], manipulate chaotic signal attributes without synchronization, thereby simplifying receiver architecture and enhancing robustness to noise. However, this approach compromises physical-layer signal protection. In practice, non-coherent variants predominate, owing to their straightforward deployment and adequate noise resistance for applications where advanced signal protection is not crucial. Both types exploit the nonlinear, broadband nature of chaotic signals for effective modulation. The succeeding subsections provide an overview of the fundamental mechanisms employed by prominent coherent and non-coherent schemes.

1.3.4 Non-Coherent Chaotic Modulation Schemes

COOK

The chaos on-off keying (COOK)) method is based on bit-energy estimation. In this scheme, only one chaotic signal is required. The binary digits "1" and "0" are represented by transmission of the signal and null transmission, respectively. The modulation process can thus be considered similar to the on-and-off state of a chaos generator. The bit energy associated with the transmitted signal assumes either a positive value or a null value, depending on the symbol transmitted. The demodulation process can be executed in a noncoherent manner by employing a straightforward bit energy estimator. This approach represents a minimalist non-coherent chaotic modulation method, leveraging the advantages of chaos's broad spectrum while omitting channel protection features.

DCSK

In DCSK (see Fig. 1.5), each transmitted symbol consists of two consecutive parts of a chaotic signal. The initial segment of the signal is designated as the support signal, while the subsequent segment contains the information. To illustrate, when a "1" bit is transmitted, both parts are identical; however, when a "0" bit is transmitted, the other part of the signal is an inverted support signal. In the receiver, the correlation of the support part and the information part in the duration of the transmitted half-symbol is determined. The method's drawbacks include its reduced transmission speed, its sensitivity to channel noise, and its lack of communication protection [42].

The reference and information-bearing signals are subject to corruption by channel noise. This results in a correlated noise reference signal and a noisy information-bearing signal at the receiver. Consequently, the performance of the DCSK system is deteriorated. A growing number of studies are underway to address the aforementioned limitations of the DCSK scheme, leading to the development and introduction of numerous modified DCSK schemes. One such scheme, frequency-modulated differential chaos shift keying (FM-DCSK) [45], is essentially a combination of FM and DCSK modulations. The result is a wideband chaotic signal with constant power and better noise immunity.

The following are additional examples of DCSK modifications.

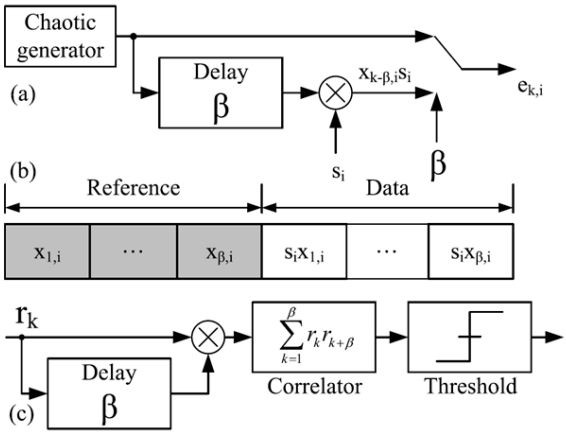


Fig. 1.5. DCSK general structure with (a) transmitter, (b) frame, and (c) receiver [42].

- The M-ary differential chaos shift keying (M-DCSK) [46] model involves the conversion of each n bits to a symbol.
- The high-efficiency differential chaos shift keying (HE-DCSK) [47] is a data modulation technique that allows for the transmission of two bits of data within a single data-modulated sample sequence.
- Reference-modulated differential chaos shift keying (RM-DCSK) [47] is an enhanced version of HE-DCSK. In this system, the chaotic signal is transmitted in either the first or second slot of each frame. In addition to carrying a single bit of data, the chaotic signal functions as the bearer of the data bit that is subsequently transmitted in its subsequent time slot.
- The noise reduction differential chaos shift keying (NR-DCSK) method [48] involves the generation of chaotic samples, which are then duplicated rather than being used as a reference sequence. At the receiver, identical samples are averaged, and the resultant filtered signal is correlated to its time-delayed replica to recover the transmitted bit.

CDSK

In the case of correlation delay shift keying (CDSK) [49], the transmitted signal is the sum of the chaotic signal and its delayed copy, multiplied by the information signal ("1" or "0"). In comparison with the DCSK transmission method, there is no necessity to alternate between a chaotic signal and its delayed replica at the transmitter side. In the CDSK scheme, the delay may vary from the half-symbol duration value. This transmission method offers enhanced confidentiality but reduced noise immunity.

QCSK

In quadrature chaos shift keying (QCSK) [50], the transmitted symbol consists of two bits of information. The system can be regarded as a four-level version of the DCSK, which exhibits double the spectral efficiency of the DCSK scheme.

As illustrated in Fig. 1.6, the block diagram of a QCSK communication system is depicted, and its operational functionality is described as follows. A chaotic reference signal $c(t)$ is delayed by half the symbol duration to produce $d(t)$. Furthermore, the phase of all frequency components is shifted by $\pi/2$, thereby forming a complementary signal, $e(t)$. Due to the $\pi/2$-phase shift, the two signals $d(t)$ and $e(t)$ are orthogonal to each other. In the QCSK modulation scheme, the chaotic reference signal $c(t)$ is transmitted during the first half symbol period, while in the second half symbol period, two orthogonal data-bearing signals are generated. These signals are produced by multiplying the two information bits q_c and q_s to be sent with $d(t)$ and $e(t)$, respectively. Then, these signals are added together before transmission. At the receiving side, the initial estimation of $d(t)$ and $e(t)$ is derived from the corrupted reference signal. Then, the correlation of the signal received in the second half symbol period with the estimated signals is performed. The symbol (two bits of information) can be further decoded by an appropriate decision-making algorithm based on the correlator outputs.

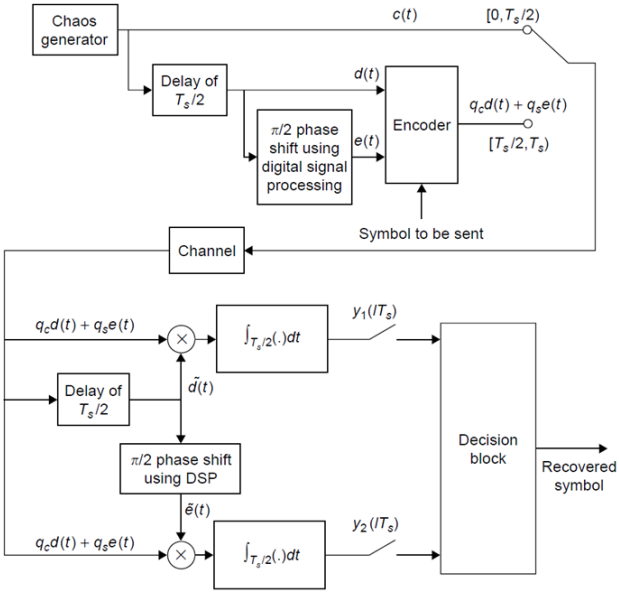


Fig. 1.6. QSCK scheme [3].

1.3.5 Coherent Chaotic Modulation Schemes

Chaotic masking

A straightforward and widely employed chaotic communication technology is information signal masking with a chaotic signal (see Fig. 1.7) [51]. In the transmitter, an intense chaotic signal is added to the low-level information signal. The receiver then synchronizes the chaos generator using the received signal, resulting in the receiver's chaos generator producing a chaos signal that is identical to the transmitter's generator. The detected signal is derived as the difference between the

input signal and the signals of the local synchronized chaos generator. The proposed transmission technology functions effectively in a noise-free channel when the power of the chaotic signal is significantly higher than the power of the information signal [3]. Another constraining factor pertains to the dispersion of the control parameters of the identical chaotic generators of the transmitter and the receiver in real systems, which complicates the implementation of data transmission in practice. In the presence of noise in the channel, the reconstructed information signal will include all of the channel noise. This is not always possible to filter out in subsequent signal processing. The implementation of chaotic masking enables the transmission of both analog and digital information, with the latter exhibiting enhanced resilience to channel noise.

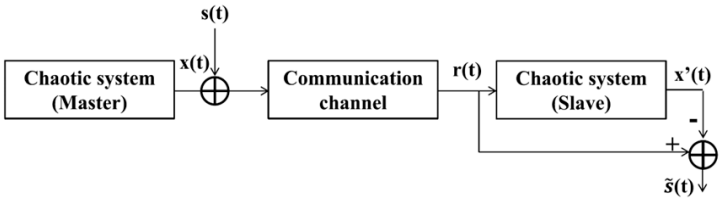


Fig. 1.7. Chaotic masking structure [42].

CSK

In the CSK method (see Fig. 1.8) [52], the transmitter contains two or more chaos generators, the parameters of which differ or the generators themselves differ. The transmission channel's chaos generators are configured in accordance with the information signal. To illustrate, the first generator is switched on for the transmission of a "1" bit, while the second one is switched on for the transmission of a "0" bit. The configuration of this transmission technology varies depending on the design of the receiver and the principles of signal detection. For instance, if two chaos generators are employed in the transmitter, one or two identical generators can be utilized in the receiver. Detection is based on chaotic synchronization. The parameters of the generators must be configured in such a manner that full synchronization occurs exclusively when the corresponding bit is transmitted. The detection process is dependent upon the occurrence of synchronization in the receiver generator or generators. This transmission technology exhibits enhanced resilience to channel noise when compared with chaos masking; however, its operational stability remains constrained. The implementation of signal detection through the utilization of cross-correlation between the input signal and the signal of the synchronized receiver generator is also a possibility. A notable disadvantage of chaotic mode switching technology is the occurrence of transient processes during switching, which results in a reduction in the transmission rate.

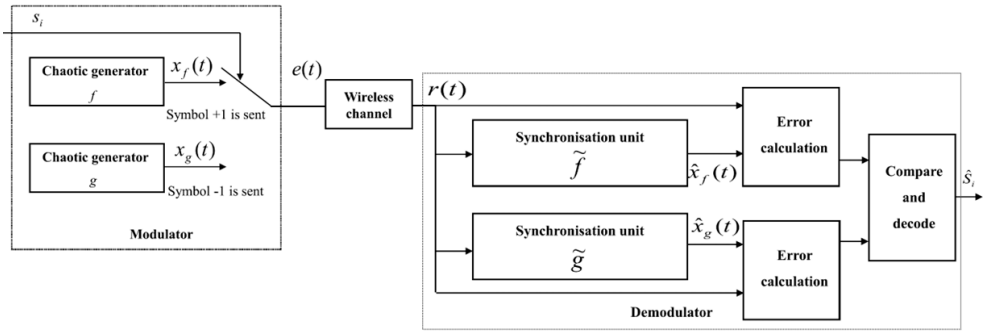


Fig. 1.8. CSK structure [42].

1.4 FPGA Implementation as an Alternative to Analog Implementation

Chaotic oscillators have traditionally been realized through analog electronic circuits, such as the Chua [22], [53] or Rössler [54] designs, as well as variants like the Sprott [13], [55] circuit or Colpitts oscillator [9], [56], incorporating elements like resistors, capacitors, inductors, amplifiers, transistors, and diodes to sustain nonlinear oscillations in a chaotic regime [15]. These systems can be modeled with straightforward ODEs that capture their dynamic behavior. However, deploying analog chaos generators in communication applications presents significant hurdles, including susceptibility to parameter drifts from environmental influences like temperature, humidity, or power supply variations, which cannot be readily corrected during runtime. Such instabilities hinder reliable chaotic synchronization, which is essential for aligning identical master and slave oscillators. Analog circuits also lack control over generators' initial conditions [42].

To overcome these limitations, some researchers have shifted toward discrete digital implementations, particularly using FPGAs, which enable the numerical solution of ODEs to approximate chaotic dynamics in a deterministic manner [10], [16]. This approach eliminates drift issues, supports high-speed parallel processing via pipelining, and facilitates flexible prototyping with hardware description languages (HDLs) [5], [6]. FPGAs thus offer substantial advantages for chaotic communication systems, including enhanced computational throughput and adaptability to standards. Nevertheless, challenges persist in FPGA designs, such as optimizing resource usage, ensuring stability at elevated clock frequencies, selecting appropriate data bus widths in fixed- or floating-point formats, and choosing ODE solving methods, all of which impact signal accuracy, robustness, and overall system efficacy. As FPGA platforms grow more accessible, they revitalize interest in chaotic systems by enabling efficient, reliable prototypes that balance performance with practical constraints.

The implementation of continuous-time chaos generators can also be achieved through a variety of digital platforms, such as digital signal processings (DSPs) and application-specific integrated circuits (ASICs). While these devices can exhibit robust performance in specific applications [18], they are subject to limitations that distinguish them from FPGAs. ASICs offer no capacity for re-

programming or design flexibility, while DSPs lack the parallel processing capacity that is essential for high-throughput operations.

1.5 Review of the Chaotic System's FPGA Implementations

FPGA technology underwent a period of rapid advancement in the early 21st century, driven by increased accessibility due to reduced costs and enhanced capabilities in terms of logic elements and features. This progress enabled the adoption of FPGAs for creating innovative approaches and refining established methods, such as those involving chaos generator-driven communication systems [5].

Despite the fact that the field of chaotic communications has attracted scholarly attention for approximately three decades, it remains under-explored. This phenomenon may be attributed to the inherent complexities of chaos theory and the relative obscurity of chaotic methods when compared to traditional communication frameworks. Nonetheless, the integration of FPGAs for the implementation of chaos generators has witnessed a consistent increase in recent times.

In contemporary settings, the necessity for advanced communication solutions with increased signal protection across diverse sectors remains on the rise. Nevertheless, a review of various scholarly publications on FPGA-integrated chaos generators and associated communication frameworks (outlined below) indicates that such studies are comparatively scarce relative to chaotic systems relying on alternative analog or digital platforms [42]. This scarcity of research focusing on FPGA in the context of chaotic communications may be attributable to the technology's recent evolution and the complexity of low-level programming, which is crucial for FPGA configuration. Consequently, there is a pressing need to further explore FPGA-based chaotic communication systems to elucidate their strengths, limitations, and viable uses.

Table 1.1 provides an overview of research that has utilized FPGAs to generate dynamic chaos. Typically, the focus of these investigations lies in the generator itself; however, four investigations incorporate chaotic synchronization, thereby establishing a linkage between master and slave units. The metrics employed for the purpose of comparison include the generator type, synchronization technique, peak operational clock rate of the prototype, discretization method for ODEs, and system bit precision. As indicated by these studies, the merits of FPGA over analog counterparts include superior reliability, the ability to execute operations concurrently at a high speed, and a reduction in energy usage.

Table 1.2 presents an overview of extant research on chaotic communication systems implemented via FPGAs. This development marks a shift from research into chaos generation and synchronization techniques to the design and verification of practical communication systems. The evaluation criteria are consistent with those outlined in Table 1.1, supplemented by the specific modulation technique employed in each instance.

Table 1.1
Overview of FPGA-implemented chaos generators and associated chaotic synchronization (where relevant)

Paper	**Chaos generator**	**Chaotic synchronization method**	**FPGA prototype max clock frequency, MHz**	**ODE discrete solving method**	**Max throughput, Gbps**	**Bit width of a system (integers with sign + fraction)**
[57]	Lorenz	error linear feedback	29.837	RK-4	0.95	32(16 + 16)
[58]	modified Lu	none	not mentioned	not mentioned	not mentioned	10(4 + 6)
[59]	Lorenz	author's	25.254	RK-4	0.81	32(16 + 16)
[28]	author's	adaptive control	not mentioned	not mentioned	not mentioned	32(16 + 16)
[60]	author's	none	not mentioned	not mentioned	not mentioned	not mentioned
[61]	Lorenz	none	100.59 (32 bit), 136.13 (16 bit)	forward Euler	3.22 (32 bit), 2.18 (16 bit)	32(14 + 18), 16(11 + 15)
[62]	Lorenz, Chen, Lu	none	156.25	forward Euler	5	32(16 + 16)
[63]	modified Chua	none	80.053	forward Euler	2.56	32(8 + 24)
[31]	author's	none	111.739	forward Euler	3.58	32(8 + 24)
[1]	described in [64]	adaptive control	not mentioned	forward Euler	not mentioned	32(not mentioned)
[30]	author's	none	117.4	RK-4	3.76	32(7 + 25)
[65]	Liu	none	50	forward Euler	not mentioned	25(not mentioned)
[66]	Lorenz	none	15	forward Euler	0.96	64(mixed)
[67]	modified Chua	none	180.180	RK-4	5.77	32(floating point)
[27]	author's	none	97.96	forward Euler	3.13	32(13 + 19)
present	modified Chua	error linear feedback	76	forward Euler	1.06	master:14(6 + 8), slave:17(9 + 8)

Among the earlier contributions to this field, [68] is a system tailored for secure image transfer, incorporating a customized Chua chaos generator on FPGA for data encoding alongside traditional Pecora-Carroll synchronization [33] for coherent symbol recovery. A subsequent work [34] details the FPGA deployment of an extended Lorentz chaotic framework, integrated into a communication setup where receiver-side synchronization relies on a personal computer (PC)-based controllable

chaos unit. Another investigation [69] focuses on an FPGA-driven system featuring an original chaos source and Hamiltonian-observer-based synchronization [70]. In a different study [71], the authors explored four variants of a basic COOK scheme using a Sprott generator, assessing performance across FPGA, analog hardware, and Simulink simulations. BER comparisons highlighted superior noise resilience in the FPGA version over analog equivalents. Furthermore, [72] introduces an FPGA implementation that leverages Rössler generator synchronization, thereby achieving significant enhancements in clock speed and data throughput by reducing the bit width from 32 to 19. Next study, [73] surveys multiple chaos generators and synchronization techniques, including open-plus-closed-loop (OPCL) [74], while evaluating various discrete ODE solvers.

The subsequent publications from 2024–2025 underscore the escalating appeal and practical viability of FPGA solutions in chaotic encryption and protected data exchange. For instance, Nuñez-Pérez et al. [75] presented an FPGA-based image encryption framework that utilizes 16-chaotic phase shift keying (C-PSK) within a coherent configuration. A later contribution by the same lead-author group, Nuñez-Pérez et al. [76], introduced a multiuser framework employing a reconfigurable chaotic oscillator (RCO) that emulates twelve oscillators to generate distinct chaotic segments, combined with 2^{24}-order chaos parameter shift keying (CPSK), chaos-based selective symbol (CSS) encoding for user identification, and wideband code division multiple access (WCDMA) spreading, achieving high spectral efficiency and low inter-user interference. In a similar vein, Estudillo-Valdez et al. [77] presented an effective FPGA-deployed image encryption method incorporating DCSK modulation alongside code division multiple access (CDMA) for multi-channel operations. Bonny and Al Nassan [78] examined obstacles in multi-stage chaos-based communication systems, assessing the protection and performance of an FPGA prototype across varying cascade depths. Concurrently, Karagiorgos et al. [79] investigated a novel digital chaotic encryption communication protocol, implemented on both FPGA and the ESP32 Arduino software environment. More recently, Xiao et al. [80] demonstrated FPGA realization of a preset-time convergence fuzzy zeroing neural network (PTCFZNN) controller for Chen chaotic oscillator synchronization under aperiodic parameter excitations, applied to chaos-masking communication, which proved robust to hardware-induced quantization and jitter noise.

The subsequent discussion will explore the strengths and limitations of the attributes of the FPGA-integrated chaotic system studies that are compiled in Tables 1.1 and 1.2.

A multitude of investigations depend on established chaos generators such as Lorenz, Chua, Sprott, or Rossler, though on occasion novel generators devised by the researchers are employed. The FPGA platform has been demonstrated to simplify the implementation of intricate generators in comparison to analog setups. While generators exhibit variability in terms of complexity, signal traits, state variable count, and synchronization potential, it is noteworthy that virtually any generator can be adapted for FPGA implementation in chaotic communication applications. The primary objective of these works is to present methodologies for such adaptations, applicable to diverse gen-

Table 1.2

Overview of FPGA-implemented chaotic communication systems

Paper	**Chaos generator**	**Chaotic synchro-nization method**	**Modulation**	**FPGA prototype max clock frequency, MHz**	**ODE discrete solving method**	**Max through-put, Gbps**	**Bit width of a system (integers with sign + fraction)**
[68]	modified Chua	Pecora-Caroll	authors', digital algorithm	not mentioned	forward Euler	not mentioned	32(not mentioned)
[34]	Lorenz	author's	chaotic masking	not mentioned	not mentioned	not mentioned	27(not mentioned)
[69]	authors'	Hamilton forms	chaotic masking	275.71	forward Euler	5.24	19(15 + 4)
[71]	author's and Sprott	none	COOK	not mentioned	not mentioned	not mentioned	not mentioned
[72]	Rossler	Pecora-Caroll	chaotic masking	70.9	forward Euler	2.85	19(10 + 9)
[81]	authors'	Hamilton forms	chaotic masking	1	forward Euler	not mentioned	28(4 + 24)
[73]	Sprott	Pecora-Caroll, Hamilton forms, OPCL	chaotic masking	116.4	Forward Euler, trapezoidal, RK-4	3.26	28(7 + 21)
[75]	various	Hamilton forms	16-C-PSK	not mentioned	not mentioned	not mentioned	32(not mentioned)
[77]	van Wyk	none	DCSK-CDMA	not mentioned	RK-4	not mentioned	40(21 + 19)
[78]	authors'	not mentioned	chaotic masking	31.8	RK-4	not mentioned	32(8 + 24)
[79]	authors', nonau-tonomous digital	direct coupling	chaotic encryption	not mentioned	not applicable	not mentioned	not mentioned
[76]	RCO	error linear feedback	2^{24}-CPSK + CSS + WCDMA	200	not mentioned	not mentioned	not mentioned
[80]	Chen	PTCFZNN	chaotic masking	not mentioned	not mentioned	not mentioned	not mentioned
pre-sent	modified Chua	error linear feedback	AM-ACSK/ FM-ACSK	76	forward Euler	1.06	master:14(6 + 8), slave:17(9 + 8)

erator types.

With respect to synchronization strategies, the prevailing approaches are the traditional Pecora-Carroll method [33] and the refined Hamiltonian-observer technique [70], with the error feedback

approach [35] being documented in a single case. The selection process generally aims to strike a balance between two key factors: synchronization complexity and reliability. To illustrate, Pecora-Carroll offers a basic yet effective approach, while error feedback requires greater complexity but ensures faster and more consistent alignment between the master and slave oscillators.

The majority of referenced works emphasize merging chaotic communication with image encryption protocols, often employing non-coherent techniques or the basic coherent variant, chaotic masking. In chaotic masking [42], the data stream is appended to the chaotic carrier for the purpose of concealment. While this technique is notable for its simplicity, it demonstrates deficiencies in noise tolerance and overall safeguarding when compared to more advanced CSK approaches. However, implementations based on CSK are rare due to longstanding obstacles. In analog configurations, the wireless implementation of CSK has proven to be a challenge due to the sensitive nature of chaotic synchronization, which ideally requires perfectly matched oscillators – a feat that has been elusive in analog hardware. Furthermore, conventional CSK designs necessitate the use of four oscillators, which can lead to escalating energy demands.

In the context of ODE discretization, the forward Euler technique [6] is widely regarded as a preferred approach due to its simplicity and ease of implementation. The fourth-order Runge-Kutta (RK-4) method [81] is more complex and requires additional FPGA resources; however, it yields superior solution fidelity, which is preferable when precision is essential. Certain generators are more susceptible to approximation errors, thereby necessitating the utilization of RK-4 over Euler to ensure convergence of the ODE solutions.

The prototype clock speeds range from 1.00 to 275.71 MHz, contingent on design sophistication. Beyond certain thresholds, metastability in logic can render operations invalid. A range of strategies have been proposed to elevate frequencies, including enhanced pipelining, reduced asynchronous paths, minimization of bit width, and substitution of multiplications and divisions with bit shifts for constants [82]. However, the inherent complexity of generators frequently restricts the applicability of these optimizations.

Throughput, defined as the minimal bit-width multiplied by the maximum frequency, ranges from 0.81 to 5.77 gigabits per second (Gbps), serving as a benchmark for digital efficiency, which is significantly influenced by achievable clock rates.

The signal bit depths range from 10 to 64, with a variety of fixed-point configurations. Integer bits are determined by the following inequality: $2^{n-1} < k < 2^n$, where k is the peak signal value and n is the bit count. Fractional bits are not subject to rigid guidelines; the incorporation of additional bits enhances precision and value diversity but reduces clock speed and increases resource demands. Thus, a balanced allocation is considered, with careful consideration of accuracy and constraints, considering variations within the system to maintain computational integrity.

In comparison with the overviewed previous studies, the present study features both distinctive and shared characteristics.

- This study incorporates a customized modified Chua generator and an error feedback synchronization method. The former is new and has not been found in the previous studies of other authors, while the latter is found in only one of the studies examined.
- Modulation employs two novel modulation schemes: AM-ACSK and FM-ACSK. These novel schemes are more sophisticated than, for example, chaotic masking, and have not been previously explored in reviewed FPGA systems.
- ODE resolution employs the forward Euler method to ensure adequate precision and stability.
- The maximum clock frequency that can be achieved is 76 MHz; however, the design operates at 50 MHz through the utilization of onboard resources.
- The bit widths in the fixed-point format used are 14 bits in the transmitter's master generator and 17 bits in the receiver's slave generators (including the sign bit), with 8 bits for the fraction.
- The maximum achievable throughput for the master generator is 1.06 Gbps; this capacity is constrained by clock and bit width. Potential enhancements to the system, such as structural redesign and pipelining, could potentially achieve higher performance, but would also incur greater complexity and resource costs. The current approach is sufficient for demonstrating the viability of the systems.

Notably, the AM-ACSK and FM-ACSK configurations are the only ones equipped with passband functionality. Furthermore, the symbol timing recovery mechanism (elaborated in the subsequent section) is not present in the other coherent designs reviewed or depends on an auxiliary signal pathway, which is an impractical choice for wireless scenarios. In the pair of non-coherent systems that lack chaotic synchronization, symbol alignment is determined by direct demodulator output correlation. However, under conditions of noise, supplementary processing becomes essential for ensuring reliable timing. Regrettably, the absence of detailed reporting in these papers impedes the ability to conduct thorough evaluations.

FPGA chaotic implementations are further contrasted by resource consumption in Table 1.3, limited to studies with available data. Metrics are typically associated with generators, although they may also include auxiliary components. Configurable logic blocks (CLBs) denote Xilinx (now AMD) basic units, while adaptive lookup tables (ALUTs) represent Altera (now Intel) equivalents, comprising registers, lookup tables (LUTs), multiplexers, adders, and carry logic. These variations hinder direct cross-FPGA comparisons but offer insights into the complexity of the systems. Memory blocks serve the purpose of storing data in instances where registers or LUTs are insufficient, registers undergo synchronization and retain values, and DSPs accelerate arithmetic operations such as multiplication between variable signals.

The current modified Chua implementation utilizes 564 ALUTs, which is less than the majority of its counterparts, indicating a modest level of complexity. It employs 84 registers, interpretable

Table 1.3

Overview of resource consumption in FPGA-implemented chaos generators

Paper	FPGA brand and series	CLBs/ALUTs *	Embedded memory bits	Registers	DSPs/ multipliers
[57]	Xilinx Virtex-II Pro	2038	813	not mentioned	40
[28]	Xilinx Virtex-6	364	not mentioned	270	not mentioned
[61]	Xilinx Zynq-7020	156 (16-bit system) 336 (32-bit system)	not mentioned not mentioned	48 (16-bit system) 96 (32-bit system)	2 (16-bit system) 8 (32-bit system)
[62]	Xilinx Zynq-7010	not mentioned	0	138	0
[63]	Xilinx Artix-7	not mentioned	not mentioned	787	64
[31]	Xilinx Artix-7	118	119	not mentioned	not mentioned
[30]	Altera Cyclone IV	2064	100000	not mentioned	0
[66]	Altera Cyclone IV	3543	not mentioned	736	48
[67]	Xilinx Virtex-6	not mentioned	not mentioned	21711	not mentioned
[27]	Altera Cyclone IV	1375	not mentioned	777	48
[34]	Altera Cyclone IV	1135	618496	789	not mentioned
[69]	Altera Cyclone IV	49251	5836819	631	92
[72]	Xilinx Zynq-7000	not mentioned	not mentioned	54	2
[81]	Altera Cyclone IV	1093	not mentioned	972	84
[73]	Altera Cyclone IV	3350	not mentioned	1919	not mentioned
present	Altera Cyclone V	564	0	84	4

* in Xilinx/AMD: CLB, in Altera/Intel: ALUT.

as six 14-bit equivalents, which is lower than the typical range. Additionally, it utilizes four DSP multipliers.

1.6 Challenges in Chaotic Communications

Significant challenges arise in the realm of communication systems based on chaos, particularly with regard to ensuring reliable synchronization, especially in coherent setups, and addressing discrepancies in parameters across transmitting and receiving ends. Another critical issue involves balancing the protective qualities of a chaotic channel against its vulnerability to interference, which must be carefully managed during system development. The predictability of chaotic patterns, in conjunction with the simplicity of non-coherent approaches, has the potential to compromise the effectiveness of protection measures. Conversely, increasing the complexity of chaotic mechanisms may reduce resilience to environmental disturbances and signal disruptions. The pursuit of innovative, more effective, and practical modulation strategies for chaos remains a crucial objective, necessary for elevating the recognition of chaotic communication techniques relative to conventional alternatives.

1.7 Introduction to Research

Despite the challenges previously mentioned, there is a strong sense of confidence in the potential of CSK to provide reliable chaotic communication. The adoption of FPGA platforms has enabled the development of chaos generators that are precisely matched to the desired specifications, with manageable initial states. This development has profoundly impacted the field, particularly in the context of coherent chaotic frameworks and CSK models. This advancement enables empirical assessments of CSK systems in real-world contexts, thereby necessitating the conceptualization, construction, and evaluation of novel FPGA-based designs as a foundational milestone.

The AM-ACSK and FM-ACSK chaotic communication system transceiver prototypes, presented in this research, incorporate both digital and analog components. The digital segment, realized on the FPGA, incorporates the modulators and demodulators alongside units for processing the transmitted and received signals. Meanwhile, the link between transmitters and receivers occurs in the analog domain, facilitated by a DAC and an ADC.

The research presented an FPGA-based experimental prototype, supported by a complementary Simulink model, to explore the functional principles of the AM-ACSK and FM-ACSK chaotic communication systems. Evaluations of system performance involve noise sensitivity analyses in AWGN channels, which are represented as BER curves across varying SNR. The main objective is to develop a proof-of-concept for a novel chaotic communication system on FPGA, emphasizing its capacity to enhance physical-layer signal protection within the IoT and wireless infrastructure.

The rationale underlying the AM-ACSK design is derived from the necessity to integrate the protective advantages of chaotic signals with the convenience of amplitude modulation for transmission on the carrier frequency, resulting in a streamlined hardware configuration that maintains pseudo-random characteristics for covert communication. This blended strategy overcomes the constraints present in conventional chaotic architectures by relocating the baseband ACSK waveform to an intermediate frequency range. This approach facilitates seamless analog channel integration without the necessity of elaborate phase restoration at the receiver end, rendering it particularly well-suited for constrained computational settings, such as FPGAs.

The FM-ACSK iteration builds upon this foundation through the incorporation of frequency modulation, thereby enhancing defenses against carrier shifts and channel irregularities while facilitating refined spectrum management and a uniform envelope [83]. This configuration underlies an unexplored combination of frequency strategies with chaotic signal characteristics, enhancing its applicability to radio-based applications. When implemented on FPGA, it employs rapid execution and adaptable reconfiguration, with a substantiated Simulink simulation of digital components enabling precise comparisons between hardware outcomes and simulated estimations.

1.8 Conclusions

This section was devoted to providing an overview of chaos applications in communication systems, including their historical background, fundamental properties, chaos oscillators, synchronization methods, modulation schemes, the role of FPGA as an alternative to analog implementations, a review of existing FPGA-based chaotic systems, key challenges, and an introduction to the research on novel chaotic communication prototypes. The key topics, including the aperiodic and broadband nature of chaotic signals, the examples of non-coherent and coherent modulation techniques, and the advantages of digital FPGA prototyping for reliability and performance, were examined to establish theoretical foundation for chaotic communications. The subsequent sections will address the design, implementation, and experimental verification of AM-ACSK and FM-ACSK systems.

The primary outcomes of this section are as follows.

- Historical context and advantages of chaos in communications: reduced energy, noise-like signals, broadband spectrum, self-synchronization for spread-spectrum use.
- Chaos properties enabling signal protection and robustness: potential indistinguishability from noise, interception resistance, dynamic obfuscation via nonlinear features.
- Examples of chaos oscillators, with detailed mathematical modeling of the Chua circuit using ODEs and its dimensionless form.
- Synchronization methods: Pecora-Carroll, error feedback, adaptive real-time techniques.
- Classification and descriptions of chaotic modulation schemes:
  - non-coherent types, such as COOK (bit-energy estimation), DCSK (differential reference and information segments), CDSK (delayed copy summation), and QCSK (quadrature with orthogonal signals);
  - coherent types, such as chaotic masking (signal addition for concealment) and CSK (generator switching for bit representation).
- FPGA advantages over analog implementations: no parameter drifts, parallel processing, and flexible prototyping to support reliable synchronization and initial condition control.
- Review of FPGA-based chaotic generators and communication systems: metrics (Euler/RK-4 discretization methods, clock speeds, bit precision, resource consumption); comparisons to proposed novel modified Chua and AM-ACSK/FM-ACSK implementations.
- Key challenges in chaotic communication: synchronization reliability, noise vulnerability, parameter mismatches, need for innovative modulation strategies.
- Introduction to the research on AM-ACSK and FM-ACSK transceiver prototypes, Simulink models, BER/AWGN evaluation, and potential for enhancing physical-layer signal protection.

2 DESCRIPTION OF THE AM-ACSK CHAOTIC COMMUNICATION SYSTEM

This section provides a comprehensive overview of the AM-ACSK chaotic communication system transceiver, which is a novel approach that integrates chaotic dynamics with amplitude modulation. The overview begins with the mathematical modeling of key components, including the chaos generator, chaotic synchronization mechanisms, and baseband modulation. This is followed by an examination of its digital implementation on an FPGA. Next, the discussion explores the signal processing chains employed by the transmitter and receiver, including modulation, demodulation, filtering, gain control, timing recovery, and bit detection. AM-ACSK system is partially based on the the author's ACSK chaotic communication system prototype design, which is published in [12], [19].

This communication system employs AM-ACSK modulation combined with chaotic synchronization to send digital data through chaotic waveforms. The prototype of this communication system is designed in Simulink and implemented on the Arrow SocKit development board featuring an Altera Cyclone V FPGA chip. Simulink's mathematical model is utilized to evaluate the performance of the FPGA prototype and its components against the simulation, streamlining the prototype's development and enabling verification of each system node's functionality. The Simulink mathematical model for each prototype unit was designed to align with the FPGA functional implementation, incorporating an identical number of registers, adders, and bits within a bus.

Figure 2.1 depicts the simplified overall architecture of the AM-ACSK communication system. In the AM-ACSK transmitter, the transmitted binary data is first processed by the ACSK modulator, which converts the data into a chaotic waveform. Then, the signal passes through the AM modulator, which shifts the signal to the carrier frequency.

On the receiver side, the AM demodulator processes the signal, and a filter removes the channel noise and unwanted frequencies. An automatic gain control (AGC) module stabilizes the signal amplitude. Finally, the ACSK demodulator decodes the chaotic waveform, resulting in the received binary data.

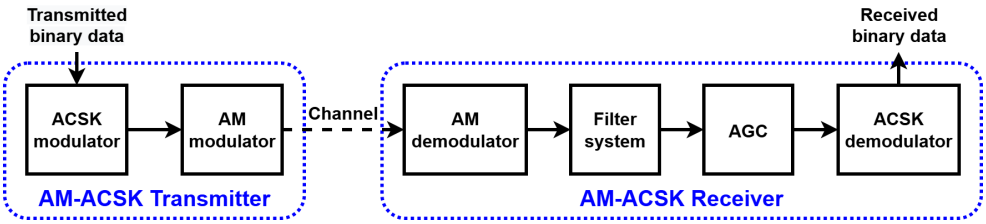


Fig. 2.1. Simplified overall architecture of the AM-ACSK communication system.

2.1 Mathematical Model of the AM-ACSK Communication System

This section describes the mathematical models and design algorithms of the key components of the AM-ACSK communication system, such as the chaos generator, chaotic synchronization, AM and ACSK modulations. Additionally, the Simulink implementation of the model is explained.

2.1.1 Modified Chua Chaos Generator

The designed communication system employs a fourth-order chaos generator based on a modified Chua's circuit [84], which is defined by the following set of ODEs:

$$\begin{cases} \frac{dp_1}{dt} = -g(p_1, p_3) - p_2 \\ \frac{dp_2}{dt} = p_1 + \gamma p_2 \\ \frac{dp_3}{dt} = \theta[g(p_1, p_3) - p_4] \\ \frac{dp_4}{dt} = \sigma p_3 \end{cases}. \tag{2.1}$$

The piecewise linear function is specified as

$$g(p_1, p_3) = \begin{cases} c(p_1 - p_3 - d) & (p_1 - p_3) > d \\ 0 & (p_1 - p_3) \leq d \end{cases}. \tag{2.2}$$

Here, p_m represents the system's state variables, and the parameters $\sigma = 1.5$, $\gamma = 0.5$, $c = 3$, and $d = 1$ are real scalar values that ensure the generator maintains stable chaotic behavior, as demonstrated in [20]. The chaotic output from the generator, $r_{out}(t)$, is derived as a weighted combination of the state variables and the piecewise linear function:

$$r_{out} = p_1 k_1 + p_2 k_2 + p_3 k_3 + p_4 k_4 + g(p_1, p_3), \tag{2.3}$$

where $k_1 = -2.6302$, $k_2 = -0.6054$, $k_3 = 0.5870$, and $k_4 = 0.7763$ are weight coefficients. These coefficients enable adjustment of the chaos generator's output signal parameters without altering the primary chaotic ODEs.

The characteristics of this chaotic generator can be examined from three perspectives: the time domain, the frequency domain, and the phase domain. In the time domain, Fig. 2.2 shows the output signal $r_{out}(t)$, sampled at 50 MHz, with irregular, aperiodic oscillations. The output oscillates between values (can be interpreted as voltage) of 7.055 and -4.277 with rms of 2.422.

In the frequency domain, Fig. 2.3 displays the power spectrum of $r_{out}(t)$ with a dominant peak at 170.4 kHz. The occupied bandwidth of the 99% signal energy is 752.4197 kHz, while a significant portion of the energy (85.274%) concentrates below 250 kHz, forming a semi-broadband profile. In this work, the focus is exclusively on the initial 215 kHz of the r_{out}, which is utilized for communication purposes.

In the phase domain, Fig. 2.4 depicts the chaotic attractor across all six pairwise projections of the

state variables p_1 to p_4, manifesting as intricate, folded strange attractor with dense, non-periodic trajectories that fill the embedding space without self-intersection or convergence to fixed points. Overall, these traits support the generator's effective use in ACSK, balancing unpredictability, spectral efficiency, and reconstruction fidelity. While other chaotic generators may offer characteristics that could enhance the efficiency of ACSK communication system design, this modified Chua generator was selected for its adequate performance in proof-of-concept validation and consistent results across experimental tests, as required for the present research.

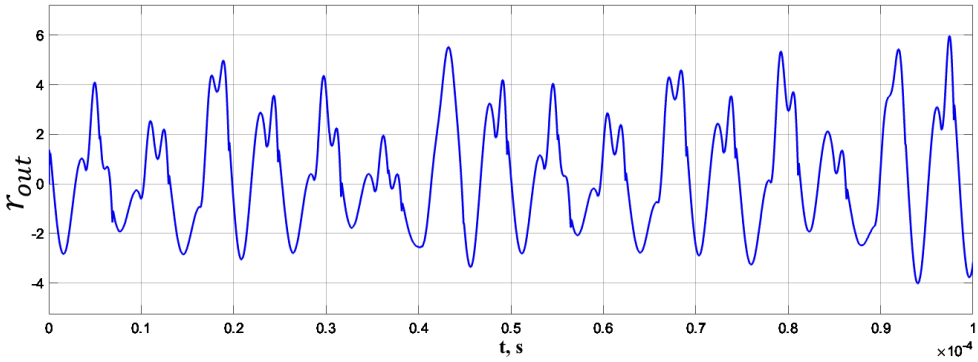


Fig. 2.2. Time domain representation of the chaos generator's output signal.

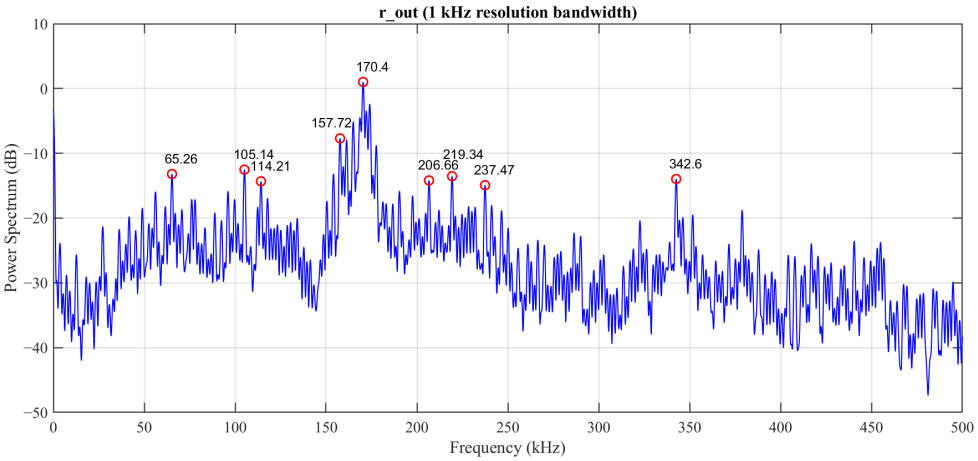


Fig. 2.3. Spectrum of the chaos generator's output signal.

2.1.2 Error Feedback Chaotic Synchronization

The chaotic synchronization between the master and slave modified Chua chaos generators is accomplished using an error linear feedback technique, as detailed in Fig. 2.5, where the master generates a chaotic signal based on equation (2.3) by computing a weighted sum of its state variables p_m and the piecewise linear function $g(p_1, p_3)$. This signal, transmitted as r_{out}, serves as the input

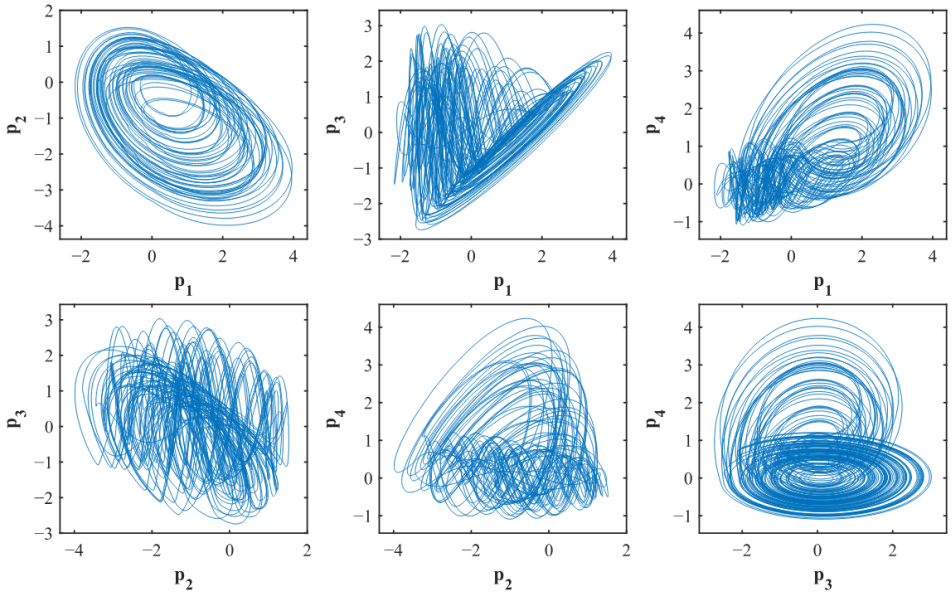


Fig. 2.4. Phase domain projection of the chaotic attractor of the modified Chua chaos generator on different phase planes.

r_{in} to the slave generator, which lacks its own piecewise linear function and instead reconstructs $g'(p'_1, p'_3)$ by subtracting the weighted sum of its own state variables $p'_1k_1 + p'_2k_2 + p'_3k_3 + p'_4k_4$ from the incoming r_{in}, defined as

$$g'(p'_1, p'_3) = r_{in} - (p'_1k_1 + p'_2k_2 + p'_3k_3 + p'_4k_4), \tag{2.4}$$

enabling a simple yet effective feedback loop for the chaotic synchronization. The slave's output mirrors the master's structure but incorporates the absolute value $|g'(p'_1, p'_3)|$ due to the non-negative nature of $g(p_1, p_3)$, with its inversion occurring when an inverted $r_{out}(t)$ signal is received, causing $g'(p'_1, p'_3)$ to take negative or zero values, which the absolute value then corrects back to positive. This feature allows for the synchronization error signal e_{SYNC} to be derived by subtracting the slave's output from its input:

$$e_{SYNC} = p'_1k_1 + p'_2k_2 + p'_3k_3 + p'_4k_4 + |g'(p'_1, p'_3)| - r_{in}. \tag{2.5}$$

2.1.3 ACSK Baseband Modulation

Figure 2.6 illustrates the simplified structure of the ACSK [85] modulation and demodulation process over a baseband channel. This scheme employs a master chaos generator at the transmitter to produce the initial chaotic signal, which is then fed into a conditional inverter. The inverter outputs

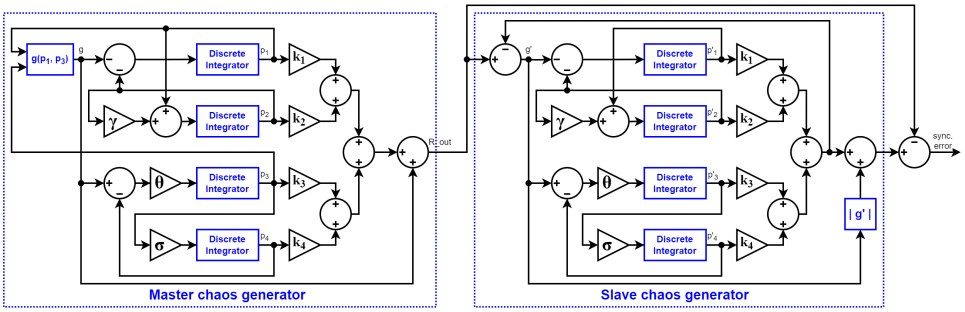


Fig. 2.5. Structure of the chaotic synchronization between master and slave chaos generators.

either the original signal $r_{out}(t)$ or its inverted version, depending on the values of the transmitted data bits $b(t)$. Consequently, the transmitter switches between direct and inverted chaotic signals based on the bit value.

At the receiver, the incoming signal $r_{in}(t)$ is handled by two slave chaos generators: one processes the direct signal, while the other deals with an inverted copy. Chaotic synchronization occurs only at one of these generators, depending on the prior inversion state of the received signal. Error signals are generated for each slave by subtracting its output from its input, followed by smoothing via sliding-window average blocks. The resulting errors, $e_d(t)$ for the direct path and $e_i(t)$ for the inverted path, are subtracted from each other in order to compare the synchronization levels. This difference (Δ_{sync}) is then used in the decision-making unit to detect the values of the received data bits $b'(t)$, with the help of the symbol phase synchronization unit to provide timing recovery.

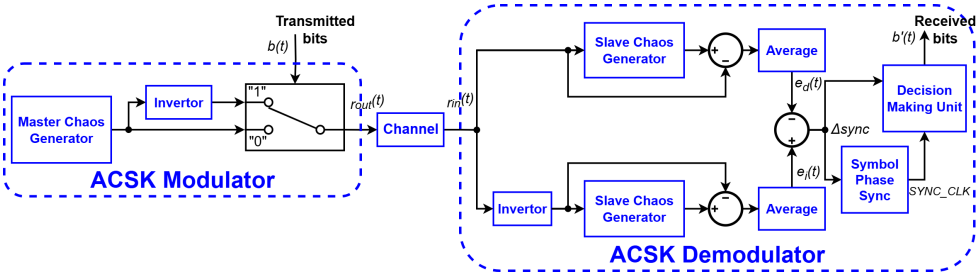


Fig. 2.6. ACSK modulator and demodulator layout with a simplified baseband channel.

Figure 2.7 presents the synchronization errors recorded over a single symbol duration. For the slave chaos generator receiving an inverted chaotic signal, the error distribution varies unevenly across time, a characteristic tied to the piecewise linear function's behavior. To achieve a more consistent and precise error measurement, averaging these errors over the symbol duration is essential.

The symbol duration spans 8192 samples (2^{13}), with each sample lasting 20 ns, corresponding to the 50 MHz clock oscillator period in the FPGA prototype. This same sampling time is used in the Simulink mathematical model. The selection of symbol duration considers the following factors:

1. The sample count needed to compute the synchronization error average with sufficient precision;
2. The samples required to achieve chaotic synchronization;
3. The symbol duration should be a power of 2 (2^n, where n is a natural number), as this significantly simplifies computations for the signal processing in the FPGA design.

Chaotic synchronization typically stabilizes within about 200 samples, significantly less than the selected symbol duration. The primary motivation for the chosen duration is to enhance the accuracy of the synchronization error average, particularly for the inverted input signal. For a 2^{12}-sample error signal, the relative calculation error is approximately 20%, dropping to 4% for 2^{13} and 2% for 2^{14}. Consequently, an 8192-sample symbol duration was adopted, offering a calculation error of about 4%, which ensures adequate accuracy crucial for reliable communication system performance amid channel noise. The primary drawback of extending symbol duration is a reduced data rate, a factor that must be taken into account when developing this communication system for real-world use. For the proof-of-concept study, however, the current method remains adequate.

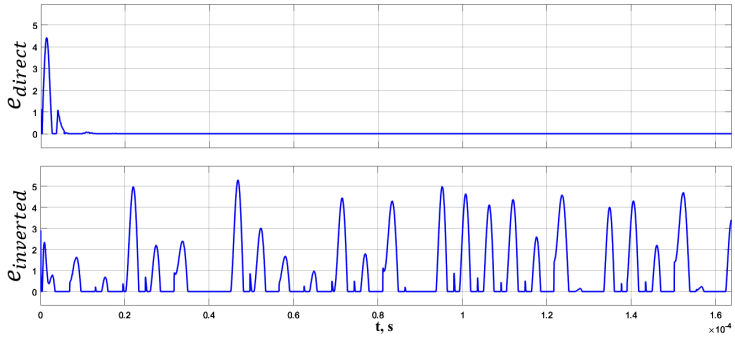


Fig. 2.7. The magnitude of the synchronization error signal for two cases: the slave generator receiving the master chaos generator's direct output (top graph) and its inverted output (bottom graph).

Figure 2.8 demonstrates the principles that allow for bit detection by displaying the synchronization errors for both slave generators during transmission of the bit sequence "10101010," as well as their values' averaged difference (Δ_{sync}). Since the sign of Δ_{sync} correlates with the bit value, detecting the transmitted data essentially involves comparing Δ_{sync} with zero within the valid symbol timing frames. The implementation of this detection is described in the end part of the present section.

2.1.4 Simulink Model of the AM-ACSK Communication System

Figure 2.9 illustrates the main units of the AM-ACSK communication system within the Simulink environment. All signals used in the model are in fixed-point format to match the digital implementation in FPGA. The signal processing modules generally come from the HDL Toolbox

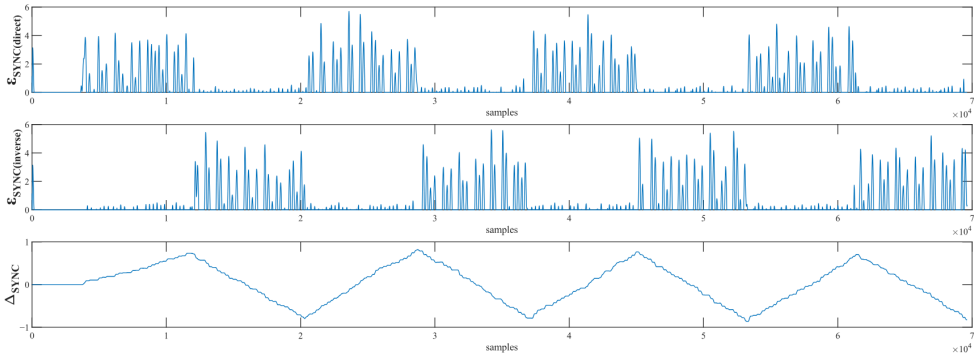


Fig. 2.8. Synchronization error traces (top to bottom): direct input slave generator, inverse input slave generator, and the difference of them averaged (Δ_{sync}) for "10101010" bit sequence transmission.

library in Simulink and are suitable for VHDL code generation. The model uses a discrete solver with no continuous states and a fixed sampling step of 20 ns, which matches the 50 MHz sampling frequency used in the FPGA design.

While the modules designed for the AM-ACSK transmitter and receiver are identical (or nearly so) to their FPGA implementations, the unit that imitates the analog channel, together with the DAC and ADC, essentially considers only two parameters measured from the wired connection setup on the FPGA prototype: delay (10 samples) and attenuation (1/0.35). The performance of the system is measured using Bernoulli random bits as an information source, and a MATLAB script is used for output processing and BER calculation. The system's module parameters are defined in a separate initialization script, which ensures efficient access to the configuration. The implementation structures of the separate modules are described in the subsections that follow.

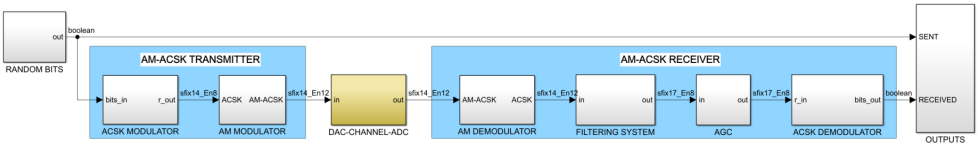


Fig. 2.9. Top-level view of the AM-ACSK system's Simulink model.

2.2 AM-ACSK Communication System Design for FPGA Implementation

This section details the FPGA-based realization of the AM-ACSK transceiver. As shown in Fig. 2.10, the prototype setup incorporates a digital transmitter and receiver on an Altera Cyclone V 5CSXFC6D6F31C6N FPGA, driven by an onboard 50 MHz clock. The analog channel is implemented with a 62 cm coaxial cable connecting the AD9767 14-bit DAC and the AD9248 14-bit ADC, both mounted on the Terasic THDB-ADA_HSMC daughterboard, which interfaces with the Arrow SocKit development via high speed mezzanine card (HSMC). The discussion within

this section focuses on the digital aspects of the system – specifically, modulation, demodulation, and signal processing.

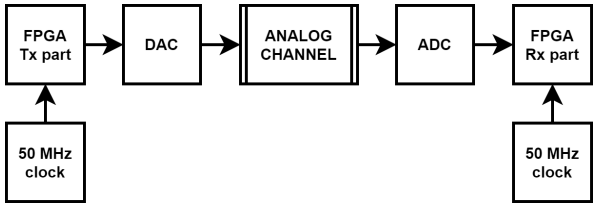


Fig. 2.10. Overview of the prototype configuration.

The FPGA prototype development proceeds through several phases, from initial concept to final hardware deployment. Core components – including the chaos generator, filter system, automatic gain control, and timing recovery unit – are first designed and tested individually and later integrated into the complete system. Figure 2.11 outlines this design workflow. The process begins with crafting a concept that defines the functionality, parameters, and structure of each module, followed by the creation of a Simulink model. This model, built with basic digital blocks and fixed-point signals, mirrors the intended FPGA implementation.

Once validated in Simulink model, each module is implemented in VHDL. Two main approaches can be used:

1. **Manual VHDL coding**, explicitly defining the interconnections among components such as adders, registers, multipliers, submodules, and other hardware elements, along with input/output signal parameters and internal signal processing logic.
2. **Automatic synthesis via Simulink's HDL Coder toolbox**, which generates VHDL code directly from the validated Simulink model.

The manual approach provides precise control over architecture and optimizations such as resource sharing or timing. It is particularly well-suited for simple or critical modules, such as a chaos generator. It allows for a thorough evaluation and refinement of the module's architecture, which fosters a deeper understanding of the underlying hardware implications. It also streamlines debugging and is advantageous when implementing algorithms that are more idiomatically expressed in native VHDL constructs (e.g., custom state machines or intricate bit-level manipulations) compared to Simulink's block-based libraries, or in scenarios where specific HDL-compatible blocks are unavailable in Simulink. However, this method is inherently labor-intensive, time-consuming and prone to human error. It requires significant expertise in VHDL syntax and hardware design principles, and maintaining consistency between Simulink models and manual VHDL descriptions can be challenging as designs evolve.

By contrast, the automatic approach accelerates the development of complex modules, such as digital filters or multi-rate signal processing chains, by automating the translation from model to code

and potentially reducing design time by orders of magnitude. HDL Coder ensures close alignment between behavioral specifications in Simulink and the generated VHDL, reducing discrepancies and supporting rapid iteration. The generated code is typically modular, well-documented, and can be refined manually if needed. Still, this approach may produce less efficient hardware due to generalized mapping strategies, and achieving optimal performance for highly constrained designs may require additional manual intervention.

After VHDL implementation, the design flow continues in Quartus Prime software, which handles editing, automated analysis, and synthesis. A testbench is also created for each module to perform functional simulations in ModelSim, where input stimuli and expected outputs validate module behavior independently of hardware-specific effects. Once verified, Quartus compiles the design into a binary file for FPGA programming. Peripheral and interface pins (e.g., clock, buttons, light-emitting diodes (LEDs)) are assigned in the Quartus Pin Planner before compilation. The compilation process includes automated analysis and synthesis, fitting (mapping logic and routing onto FPGA resources), and assembling (producing the programming file).

For debugging and verification, Signal Tap, Quartus's integrated logic analyzer, allows real-time monitoring of internal FPGA signals. Results can be viewed as timing diagrams or exported to MATLAB for further analysis. This methodology can be scaled and applied both to individual modules and to combinations of modules, making it suitable for designing everything from small subsystems to the complete communication system prototype.

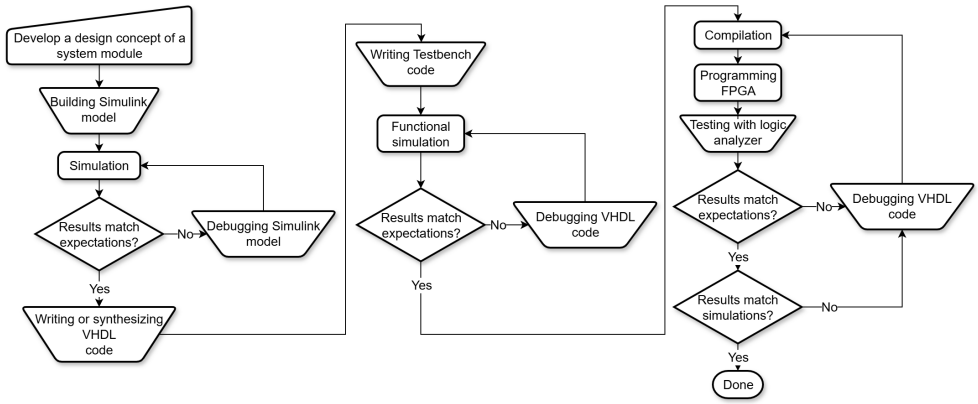


Fig. 2.11. Flowchart outlining the methodological steps for designing communication system modules for FPGA implementation.

2.2.1 Digital Implementation of Modified Chua Chaos Generator

To implement the continuous-time nonlinear chaotic system within an FPGA, which is designed primarily for discrete digital circuits, key decisions include selecting an appropriate numerical method for solving ODEs and adopting a suitable binary representation for the data of signals

[20]. The forward Euler method is employed for ODE discrete solving, which offers simplicity, stability, convergence, and minimal consumption of FPGA hardware resources compared to more complex alternatives, such as RK-4. This method approximates solutions via the iterative formula $y_{n+1} = y_n + hf(y_n)$, where y_n denotes the current state, y_{n+1} – the subsequent approximation, h – the integration step, and $f(y_n)$ – the derivative function; accuracy improves as h diminishes toward zero. In this design, h is set to $(20 \times 50 \times 10^6)/1024$, balancing precision with ease of FPGA deployment, whereas a coarser step like $(20 \times 50 \times 10^6)/256$ leads to divergence due to inadequate approximation fidelity.

Applying this method to the chaotic system's ODEs (2.1) yields the discrete form:

$$\begin{cases} p_{1_{n+1}} = -g(p_{1_n}, p_{3_n}) - p_{2_n} \\ p_{2_{n+1}} = p_{1_n} + \gamma p_{2_n} \\ p_{3_{n+1}} = \theta[g(p_{1_n}, p_{3_n}) - p_{4_n}] \\ p_{4_{n+1}} = \sigma p_{3_n} \end{cases} . \tag{2.6}$$

For numerical handling in the transmitter, signals use a 14-bit signed two's complement fixed-point format, allocating 8 bits to the fractional portion and 5 to the integer and sign bits, enabling a range from -32 to roughly 31.996 with a resolution of about 0.0039 – sufficient for the chaotic attractor's bounds and aligned with the 14-bit constraints of the DAC and ADC interfaces. The receiver generally expands this to 17 bits (8 fractional, 9 integer/sign) for enhanced processing and a gap for additive noise, though signal bit-widths vary within the modules when necessary. An alternative numerical design could employ floating-point format for calculations [8], which would be more adaptive in terms of calculation accuracy. However, such approach would significantly increase complexity and hardware resource usage.

The generator's structure, illustrated in Fig. 2.12, comprises interconnected modular components: integrators (built from adders and registers following the Euler formula, with temporary bit-width expansion and output rounding for accuracy), a piecewise-linear unit $g(p_1, p_3)$ per its defining equation (using adders, a comparator, and a multiplexer without bit adjustments), and coefficient multiplications for θ, σ, and γ achieved resource-efficiently via bit shifts and adders rather than dedicated multipliers – for example, $\times 0.5$ as a right shift, $\times 1.5$ via addition of the signal and its shifted version, and $\times 10$ through shifts by 1 and 3 followed by summation. This is done at the synthesis stage, by the Quartus software. At the output, state variables p_1 through p_4 are scaled by fixed weights $k_1 = -2.6302$, $k_2 = -0.6054$, $k_3 = 0.5870$, and $k_4 = 0.7763$, producing 28-bit products that are rounded back to 14 bits by selecting balanced fractional and integer portions. These weighted terms are then summed with $g(p_1, p_3)$ across four adders to form the final chaotic signal r_{out}.

Registers are confined to integrators, aligning one clock cycle with each Euler iteration for streamlined design, though this caps the operational frequency due to adder propagation delays; in-

corporating additional registers could enhance pipelining and speed but would introduce algorithmic complexity and greater resource demands.

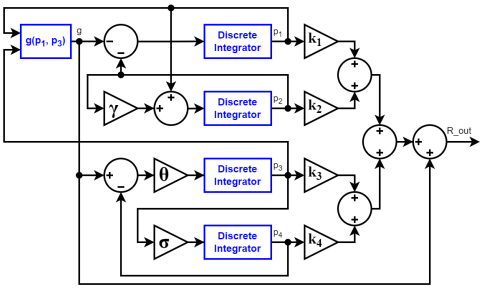


Fig. 2.12. General implementation design of the modified Chua chaos generator.

2.3 AM-ACSK Transmitter Signal Processing

As illustrated in Fig. 2.13, the signal processing in the AM-ACSK transmitter essentially consists of ACSK modulation followed by AM modulation. The process begins with the master chaos generator, which produces a 14-bit fixed-point baseband chaotic signal (r_{out}) that serves as the carrier for the ACSK modulation. This signal is then sent to the inversion controller, a critical module where binary data (inv_en) modulates the carrier through antipodal switching, a hallmark of ACSK. The ACSK-modulated output signal, r_{acsk}, is either inverted ($-r_{out}$) when $inv\_en = 1$ or passed unchanged (r_{out}) when $inv\_en = 0$ over a predetermined symbol duration of 8192 samples. Digital inversion in two's complement format involves two steps. First, all bits are inverted. Then, one is added. A multiplexer selects between the original and inverted r_{out} signal based on the incoming binary data bit.

Next, the ACSK-modulated signal, r_{acsk}, is sent through an AM modulator. This modulator shifts the signal to a passband (r_{am}) in preparation for transmission. The AM modulator module design is described in the next subsection.

Following this, the most significant bit (MSB) inversion module converts the signal to offset binary format (r_{dac}), shifting the AM-ACSK signal to positive unsigned values. This ensures compatibility with the DAC's input requirements. Finally, the processed digital signal is sent to the DAC, which converts it into an analog waveform for transmission.

2.3.1 AM Modulator

Figure 2.14 shows the AM modulator design, implemented in Simulink for the FPGA. Considerations of the maximum and minimum signal amplitudes are important for defining the modulation index and the design of the data bit width at each signal processing stage. Since the ACSK-modulated signal (14-bit signed fixed-point with an 8-bit fraction) oscillates within maximum values of ± 7.055,

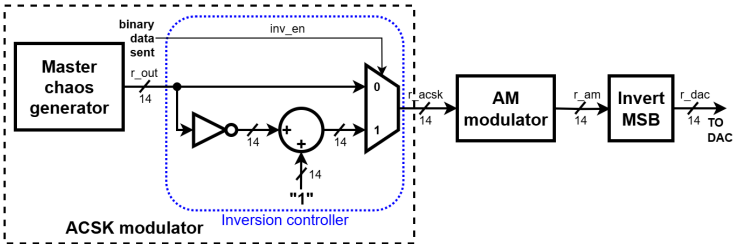


Fig. 2.13. Structure and some design elements of the AM-ACSK transmitter.

it can fit compactly in a 12-bit signed fixed-point with an 8-bit fraction data bus. Using the XOR operation with a high MSB mask results in an inverted MSB value, which essentially adds a DC component of 8. Interpreting this shifted ACSK signal as unsigned causes it to oscillate between the values 15.055 and 0.945. It is then ready to be multiplied by the carrier.

The AM carrier is generated using an NCO, which is a block from the Simulink's DSP HDL Toolbox library that digitally synthesizes a sine wave with a frequency of 10.7 MHz using a compressed LUT. The NCO employs 26 bits for the accumulator, 13 bits for phase quantization, and an additional 13 bits for phase dithering. These parameters ensure an output frequency resolution of approximately 1 Hz and a spurious-free dynamic range (SFDR) of roughly 90 dB. The carrier increment parameter, equivalent to 14,361,297, establishes the 10.7 MHz carrier output. It is calculated as follows:

$$\text{carrier increment} = \text{round}(F_0 \times 2^N \times dt), \tag{2.7}$$

where F_0 is the desired passband carrier frequency, N is the number of bits used in the accumulator, and dt is the sampling step (20 ns).

Although the NCO output is constrained to an amplitude of 1, shifting the binary point four bits to the right increases the maximum absolute value of the carrier signal from 1 to 16. This ensures that the AM modulation index ($15.055/16 = 0.9409375$) is relatively close to 1. This is a critical factor for the AM-ACSK noise resistance performance.

The multiplication of the previously modified ACSK signal by the carrier signal results in the ACSK signal shifting to the passband, which is the key operation for the AM modulation design. The product's maximum amplitude and data bit widths are calibrated to ensure the generation of an amplitude of 0.941 within a 14-bit signed fixed-point with a 12-bit fraction output AM-ACSK signal. A unit delay (equivalent to a register) is applied in the final stage, introducing a one-sample lag to facilitate pipelining in the FPGA design, ensuring stable clock cycles within combinatorial logic.

It is important to note that only 215 kHz of the ACSK signal is utilized for demodulation, with the remainder being filtered out. Consequently, the bandwidth of the AM-ACSK signal is $215 \times 2 = 430$ kHz.

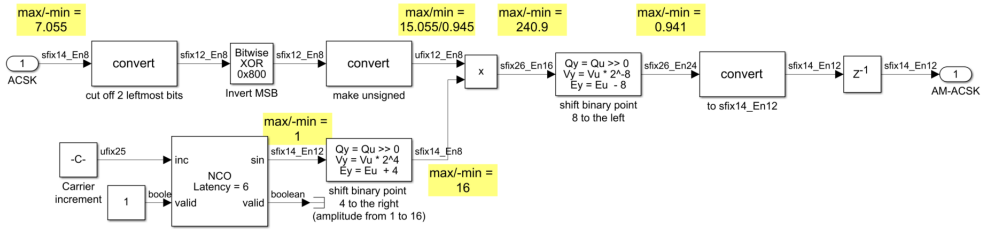


Fig. 2.14. Simulink diagram of the AM modulator design within the AM-ACSK transmitter.

2.4 AM-ACSK Receiver Signal Processing

The incoming signal requires preprocessing before the ACSK demodulator can handle it effectively. As shown in Fig. 2.15, the 14-bit straight offset binary (SOB) digital signal from the ADC (r_{adc}) undergoes an initial transformation in the MSB inversion unit. This shifts the AM-ACSK signal from positive unsigned values to a two's complement signed fixed-point format (r_h). Next, the AM demodulator unit restores the ACSK signal to baseband frequencies (r_l). Subsequent processing includes a filter system stage that removes unwanted frequencies and refines the signal (r_f). Then, an AGC unit adjusts the signal level (r_{in}). The resulting processed signal is fed into the ACSK demodulator, which extracts the transmitted binary data. The design of some of these relevant modules is explained further.

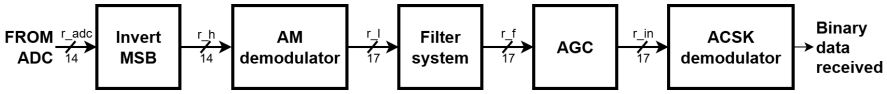


Fig. 2.15. Structure of the AM-ACSK receiver.

2.4.1 AM Demodulator

The AM demodulator in this communication system plays a pivotal role in recovering the baseband ACSK signal from the AM carrier. It is designed for simplicity and efficiency in hardware-constrained environments. Rather than relying on complex carrier phase recovery mechanisms, it uses an incoherent envelope detection approach. The module's design, which avoids phase-locked loops and coherent detection, serves to minimize computational overhead and latency. This makes it well-suited for real-time FPGA implementation while maintaining robustness against phase uncertainties introduced by the channel.

As depicted in Fig. 2.16, the first operation computes the absolute value of the input AM-ACSK signal. This effectively rectifies the modulated waveform and extracts its envelope. This step produces a unipolar signal that approximates the original ACSK modulation envelope, but it also introduces harmonic distortions and a DC offset. To mitigate the DC component, the DC Blocker module from the Simulink library is used, which is configured in cascaded integrator-comb (CIC) filter mode with the differential delay equal 8192 samples. CIC filters are well-suited for low-resource efficient

FPGA implementation. This filtering centers the envelope around zero, restoring a bipolar representation that more closely resembles the original ACSK chaotic carrier signal, while adding only minor spectral distortion closely to 0 Hz.

Following the offset correction, a unit delay is applied. Before output, the signal is converted to a wider fixed-point format, expanding from 14 to 17 bits while preserving the 12 fractional bits. This bit-width increase provides headroom for potential overflow in downstream modules, such as filtering and AGC, while maintaining precision for the recovered ACSK output.

It is important to note that AM demodulation is not finalized within this module alone. The absolute value operation generates unwanted high-frequency artifacts, including harmonics of the carrier and intermodulation products resulting from the chaotic signal's broadband nature. The subsequent filtering system addresses these spurious components by attenuating frequencies outside the desired baseband.

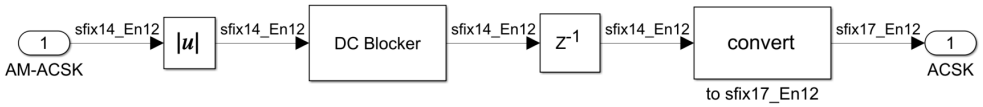


Fig. 2.16. Simulink diagram of the AM demodulator design within the AM-ACSK receiver.

2.4.2 Filter System

To enhance the AM-ACSK communication system's resilience against channel noise and finalize AM demodulation, the filtering system attenuates signal components in non-critical frequency bands while preserving regions vital for chaotic synchronization, specifically those under 6 kHz and spanning 90 kHz to 215 kHz. Although a unified filter with dual passbands could theoretically achieve this, it would demand a high order and intricate design; thus, the approach employs a cascade of simpler, lower-order filters for efficiency. Note that the described sampling rates and signal frequencies are tailored to a 50 MHz FPGA clock; any increase in clock speed would scale these frequencies proportionally. The overall filter architecture integrates down-sampling, a bandstop stage, and up-sampling, utilizing just two core designs: a sixth-order infinite impulse response (IIR) halfband lowpass filter with a cutoff at 0.546 times the Nyquist rate, and a twelfth-order IIR bandstop filter. As illustrated in Fig. 2.17, the bandstop filter is applied iteratively six times, with a twofold down-sampling following each iteration, progressively reducing the sampling rate from 50 MHz to 781.25 kHz. On the final application of the halfband filter in this down-sampling phase, its effective cutoff reaches approximately 213.281 kHz, sufficiently approximating the target of 215 kHz.

This down-sampling strategy enables the use of a less complex bandstop filter to eliminate the 6–90 kHz range, which would otherwise be challenging at the full 50 MHz rate. Post-bandstop processing, the signal undergoes six stages of twofold up-sampling to restore the 50 MHz sampling frequency, essential for subsequent demodulation and synchronization. After each up-sampling step,

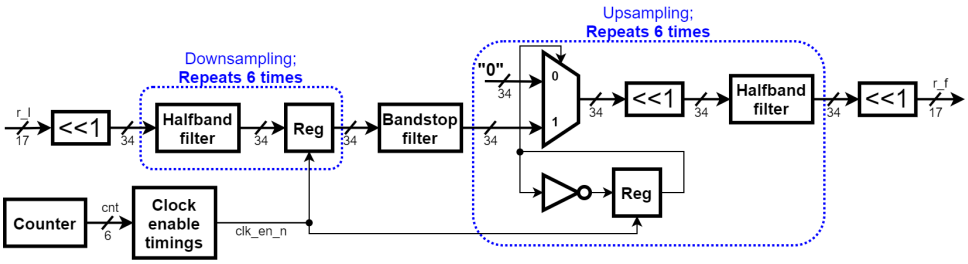


Fig. 2.17. Generalized digital implementation of the filtering system within AM-ACSK receiver.

the signal is scaled by a factor of 2 via a single left shift to offset baseband energy dilution, which arises from the spectrum splitting into baseband and half-sampling-frequency components; the extraneous higher-frequency elements are then suppressed by the halfband filter at each stage. Figure 2.17 depicts the generalized digital implementation, where filtering occurs at an expanded 34-bit bus width to ensure computational precision. Variable sampling rates are managed through modulated clock enable signals for registers, all synchronized to a common clock: for instance, down-sampling by 2 involves enabling the initial stage register every second cycle, the next every fourth, and so on. Upsampling reverses this logic, incorporating a register with inverted feedback to drive a multiplexer that inserts zeros, effectively duplicating samples by alternating with null values at double the prior rate. For reference, Fig. 2.18 presents the baseband spectrum of the transmitter's chaotic ACSK output modulated by a random bit sequence in a noise-free scenario, while Fig. 2.19 illustrates the frequency response of the complete filter system under simulated 0 dBm additive white Gaussian noise as an input in a Simulink model. The filter system provides roughly 20 dB of attenuation in the 6–90 kHz bandstop filter's range and approximately 80 dB of attenuation in the higher frequencies range, with a slope from the passband of approximately 150 kHz.

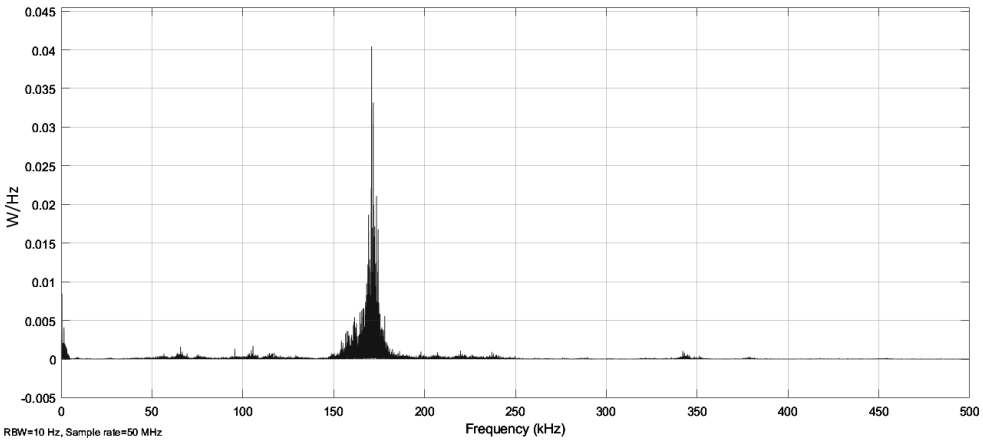


Fig. 2.18. Baseband spectrum of the transmitter's chaotic ACSK output modulated by a random bit sequence in a noise-free scenario.

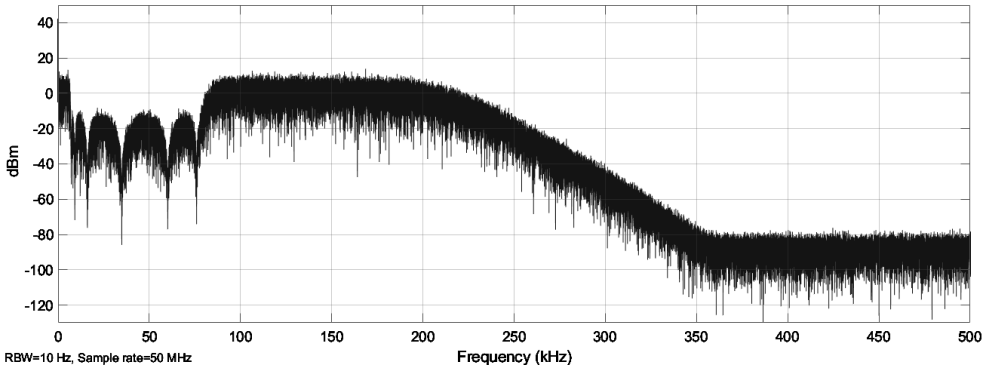


Fig. 2.19. Frequency response of the complete filter system under simulated 0 dBm additive white Gaussian noise as an input.

2.4.3 AGC

To compensate for variations in signal levels between the transmitted output and the received input – stemming from channel noise, attenuation, or other distortions – an AGC module is employed to dynamically scale the received signal toward a predefined reference power level. This ensures that the power delivered to the receiver's slave chaos generators aligns closely with that of the transmitter's master chaos generator, facilitating more reliable chaotic synchronization. The prototype incorporates feedback-based AGC architecture, as detailed in Fig. 2.20, where the input signal is first multiplied by a corrective gain factor derived from the feedback path. This factor is computed by determining the absolute value of the output signal through a combination of a comparator (to detect sign), an arithmetic inverter, and a multiplexer that chooses the inverted version for negative values or the original for positive ones. The resulting absolute value is then subtracted from a reference root mean square (RMS) value of 2.421, derived from the transmitter's modulated chaotic output, before being fed into a gain accumulator. Completing the loop, the accumulator's output undergoes division by 2^K via a logical right shift of K bits. The selection of K influences both the AGC system's dynamic range and response time; evaluations across K values from 10 to 17, with input powers ranging from 0.1 to 10 times the reference, led to $K = 14$ as a balanced performance option. The AGC maintains 17-bit resolution at both input and output, while internally expanding to 34 bits to minimize rounding errors, providing sufficient headroom for products of two 17-bit fixed-point multiplications in accordance with binary arithmetic principles.

2.4.4 ACSK Demodulation

The ACSK demodulator's general design principles were outlined in subsection 2.1.3, while this subsection delves into its digital architecture, as illustrated in Fig. 2.21. The incoming processed signal is split: one branch feeds directly into a slave chaos generator, whereas the other undergoes arithmetic inversion prior to entering a second slave chaos generator. These slave units largely replicate

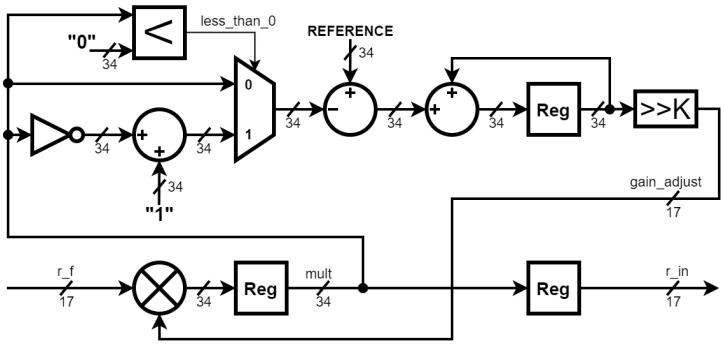


Fig. 2.20. Architecture of the automatic gain control system.

the master chaos generator's design, with key adaptations including a 17-bit signal bus throughout the receiver to handle elevated signal amplitudes during unsynchronized states; the omission of an internal piecewise linear function $g(p_1, p_3)$, which is instead recovered via a negative feedback loop; and the incorporation of the absolute value of this reconstructed function in the output signal.

For each slave generator, the synchronization error is derived by subtracting the input from the output, followed by a sliding-window average computed over 8192 (2^{13}) samples. This averaging mechanism relies on a first input first output (FIFO) buffer capable of holding 8192 entries of 17-bit data, initially outputting zeros until full, then cycling through stored values from oldest to newest. The buffer's output is deducted from the current input and routed to an accumulator operating at 30-bit precision to accommodate the summation of 8192 terms without overflow, as $30 = 17 + 13$. The accumulated result is then scaled by dividing by 8192 through a 13-bit right shift, providing the mean synchronization error. The averaged errors from both slaves are differenced and evaluated in the decision-making unit to determine the transmitted bit value, with the process synchronized via a symbol phase synchronization module that recovers timing.

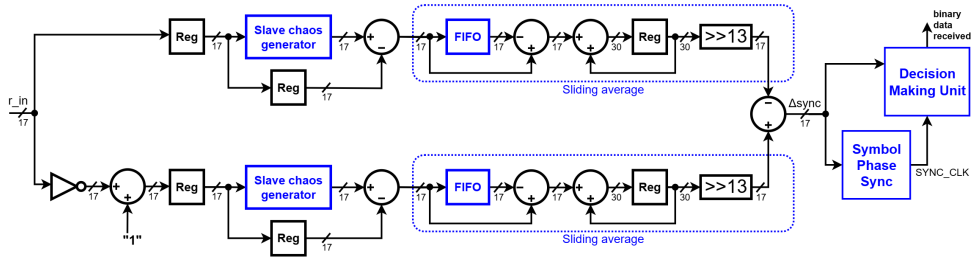


Fig. 2.21. Structure of the ACSK demodulator.

2.4.5 Timing Recovery

In real-world applications, the transmission path between the transmitter and receiver introduces unpredictable delays influenced by physical distance, leading to uncertainty in identifying the precise boundaries of each transmitted symbol (with one bit per symbol in this setup). This challenge

is typically addressed through a timing error detector (TED), which analyzes the received signal to identify and correct phase discrepancies. This aligns the receiver's clock with the incoming data stream. Among prevalent TED techniques [86], decision-directed options like zero-crossing and Mueller-Muller depend on symbol estimates and prior data insights, whereas non-data-aided feedback approaches, including Gardner and early-late gate, operate independently of transmitted content knowledge, offering greater flexibility for symbol phase blind acquisition.

These standard methods are primarily intended for traditional digital modulation schemes. However, within this chaotic communication framework, the receiver uses a known symbol duration of 8192 samples at a 50 MHz clock rate, and the characteristics of the ACSK demodulator's Δ_{sync} output correlate with the bit values. This enables a straightforward implementation of blind symbol phase synchronization via the adapted early-late gate technique.

Figure 2.22 illustrates the operational concept of the early-late gate TED applied to a sample Δ_{sync} signal. It employs two integration gates that sum the signal over the initial and latter halves relative to the assumed clock cycle, where the clock period matches the 8192-sample symbol without adjustments. When the assumed clock aligns perfectly with the symbol timing, the gate outputs differ minimally, signaling successful phase lock. Misalignments produce a significant disparity, with the difference's polarity indicating the direction of the offset (lead or lag) and its magnitude guiding the required correction. This error feeds into a loop that iteratively refines the clock phase until convergence. Post-lock, the system maintains tracking with minor tweaks throughout transmission. The illustration focuses on a "1" bit flanked by "0"s, but since absolute values are compared, it functions equivalently for other types of bit transitions as well. Efficiency drops for single-sided bit transitions ("0" to "1" or vice versa on one gate only), and no adjustment occurs during identical consecutive bits due to absent transitions. Thus, initiating transmissions with alternating bits accelerates locking and minimizes initial bit errors.

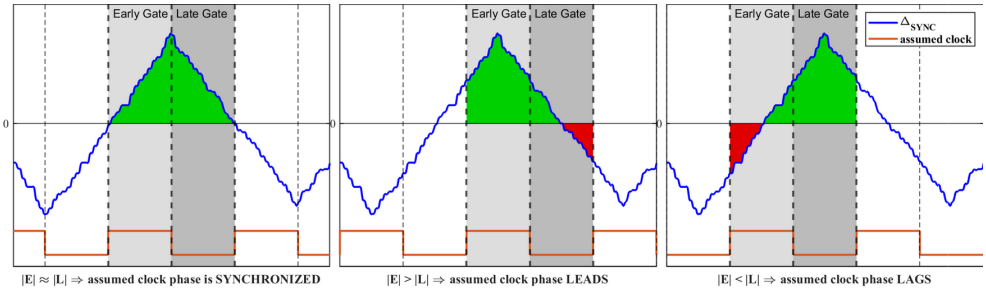


Fig. 2.22. Illustration of the early-late gate timing error detection approach, comparing accumulations in the early and late gates for the Δ_{sync} signal under different assumed clock phases, specifically for a scenario where a received "1" bit is flanked by "0" bits. The green regions highlight positive value accumulations, while red regions denote negative ones.

The developed design adapts a standard early-late gate synchronizer [87] to suit the AM-ACSK system's demands and FPGA constraints, starting with a fixed-point Simulink model converted to

VHDL using HDL Coder. As shown in Fig. 2.23, processing commences by left-shifting the Δ_{sync} input ("err_delta_in") logically by 3 bits (equivalent to multiplying by $2^3 = 8$) to amplify variations, followed by separate integrations over the early and late halves, reset by the SYNC_CLK signal. Each integrator comprises an accumulator handling roughly 2^{12} samples (with slight variations from loop adjustments) and an arithmetic right shift by 12 to approximate division by 2^{12}. This shift-based approach conserves FPGA resources compared to variable division, prioritizing efficiency over exact precision.

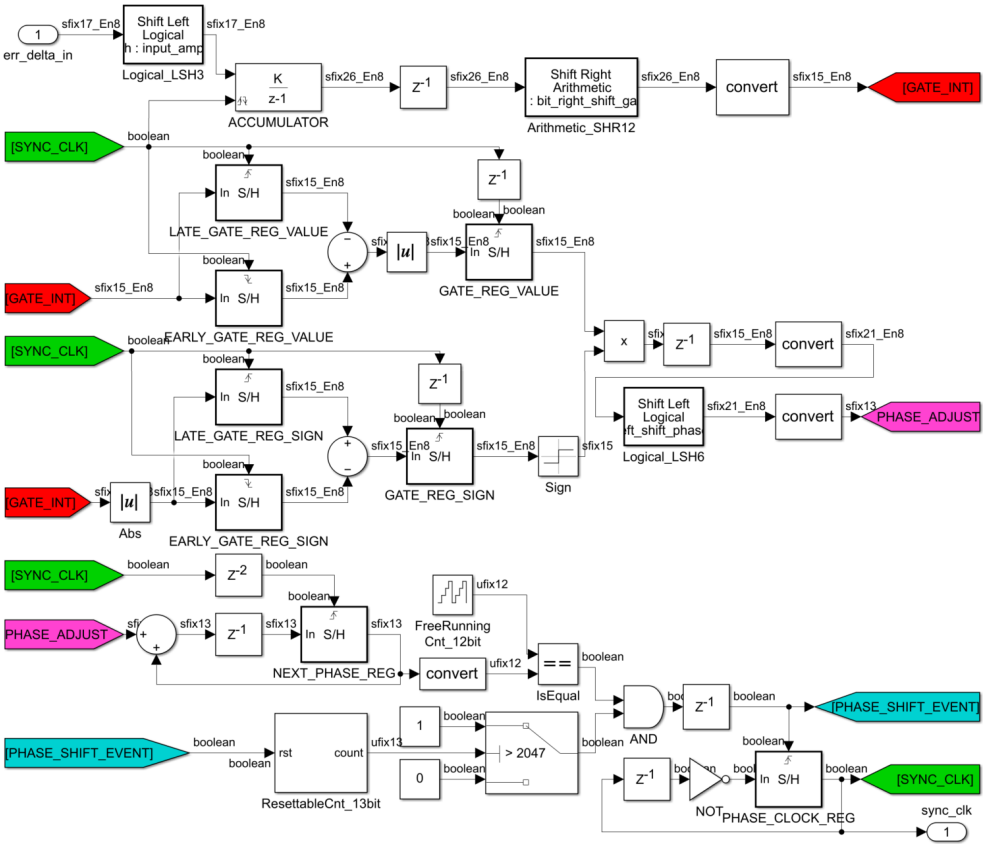


Fig. 2.23. Simulink diagram depicting the implemented early-late gate symbol phase synchronizer. Colored labels denote the signal links, and arrows on clock inputs specify the triggering clock edges.

A basic timing error estimate could be $|\mathrm{E}| - |\mathrm{L}|$ (with E as early gate output and L as late), but this limits utility to phase errors under 25% of symbol duration, yielding near-zero results for larger shifts near sign changes in Δ_{sync}. To cover full-range errors, the computation separates magnitude and direction:

$$\text{timing error} = |\mathrm{E} - \mathrm{L}| \times \text{sign}(|\mathrm{E}| - |\mathrm{L}|). \tag{2.8}$$

For practical adjustment, a refined version scales the shift and inverts the sign for feedback:

$$\text{PHASE\_ADJUST} = 2^6 \times |\text{E} - \text{L}| \times \text{sign}(|\text{L}| - |\text{E}|), \tag{2.9}$$

where the 2^6 factor (via left shift by 6) optimizes loop dynamics. Empirical tests revealed it balances rapid convergence without overshoot or oscillations, with precision.

The PHASE_ADJUST value modulates a 12-bit free-running counter generating SYNC_CLK, while a 13-bit auxiliary counter ensures no premature clock toggles during forward jumps by enforcing at least 2048 samples per half-cycle. The resulting synchronized clock feeds the downstream bit detector module.

2.4.6 Decision-Making Unit (Bit Detector)

The decision-making unit uses the Δ_{sync} signal to determine the values of incoming bits, relying on the phase timing derived from the symbol phase synchronization module. As illustrated in Fig. 2.24, this involves integrating the Δ_{sync} input ("err_delta_in") over the full symbol duration, guided by the recovered bit clock ("sync_clk_in"). Precision in this integration is not critical, since the process solely examines the sign of the outcome for binary classification: a positive result yields a "1," while a negative one produces a "0."

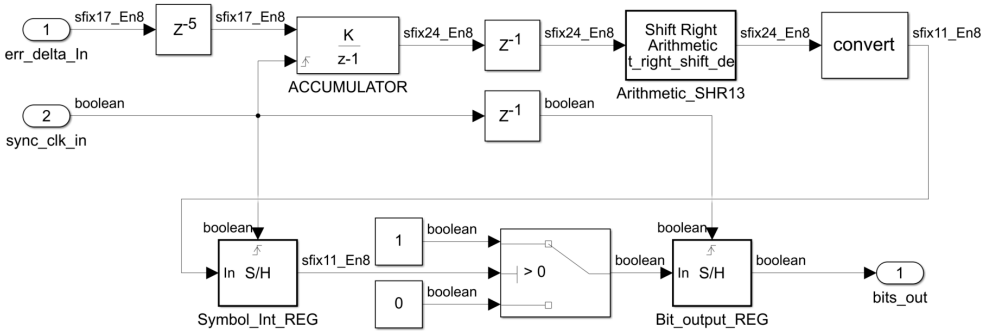


Fig. 2.24. Simulink diagram of the module for detecting received data bits.

2.5 Conclusions

This section was devoted to describing the AM-ACSK chaotic communication system transceiver, including its mathematical model, design principles, and digital implementation on FPGA hardware. The key components, such as the modified Chua chaos generator, error feedback synchronization, ACSK modulation, and signal processing chains for both transmitter and receiver, were examined in detail, with emphasis on their adaptation for efficient prototype development and verification through Simulink simulations. The noise performance of the system will be examined through experimental

verification in the following sections. Table 2.1 provides a comprehensive summary of the parameters of the designed and FPGA-implemented novel AM-ACSK discrete chaotic communication system.

Table 2.1

Overview of the AM-ACSK chaotic communication system design parameters

Parameter	Value
chaos oscillator	modified Chua
chaotic synchronization	error linear feedback
discrete ODE solving method	forward Euler
fixed-point bit width for chaos oscillator (total, fraction)	14, 8 (master) 17, 8 (slave)
FPGA chip family used	Altera/Intel Cyclone V
clock rate and sampling frequency	50 MHz
samples per data transfer bit	8192
data transfer rate	6.1 kbps
channel bandwidth	430 kHz
AM modulation index	0.94
passband carrier frequency	10.7 MHz

The primary outcomes of this section are as follows.

- The formulation of the chaos generator's ODEs.
- Discrete implementation using the forward Euler method, fixed-point arithmetic, ensuring chaotic behavior suitable for modulation.
- The establishment of robust synchronization via a feedback loop with absolute value correction for inverted signals.
- The definition of ACSK modulation with an 8192-sample symbol duration for accurate error averaging.
- Discrete AM modulation implementation with NCO.
- The elaboration of the AM-ACSK receiver modules, including:
  - discrete incoherent AM demodulation;
  - multi-rate filtering for noise rejection;
  - AGC for amplitude stabilization;
  - early-late gate symbol timing recovery;
  - sign-based bit detection via integration.
- The FPGA implementation of AM-ACSK chaotic communication system transceiver described in this section confirms the first thesis stated in the introduction.

3 DESCRIPTION OF THE FM-ACSK CHAOTIC COMMUNICATION SYSTEM

The FM-ACSK communication system transceiver is very similar to the previously described AM-ACSK system, but it has undergone major design changes. As the name suggests, the main difference is that this system uses frequency modulation instead of amplitude modulation for passband transmission. This section explains the design changes relative to the AM-ACSK system, followed by a description of the digitally implemented FM modulator and demodulator units and the new lowpass filtering system. The design and performance insights of FM-ACSK were published in [7], [17].

3.1 Changes relative to the AM-ACSK Communication System

The overall architecture of the FM-ACSK communication system's transmitter and receiver is illustrated in Fig. 3.1. Since the FM-ACSK system is a variation of the previously described AM-ACSK communication system, its core ACSK modulator and demodulator units are identical to those of the AM-ACSK system. The differences lie in the signal processing parts of the transmitter output and receiver input.

- The **AM modulator and demodulator units are replaced with FM modulator and demodulator units**, respectively. Both systems use a 10.7 MHz carrier, but FM-ACSK uses a wider channel bandwidth (1720 kHz versus 430 kHz for AM-ACSK), which aims to increase the resistance to the channel noise.
- An **AGC unit is no longer necessary** before the ACSK demodulator, since the output signal power of the FM demodulator is independent of its input signal power.
- The **filter system has undergone a complete redesign**. The AM-ACSK employs a downsampling and upsampling approach with low-order halfband IIR filters, as well as a bandstop filter for the 6–90 kHz band. For the FM-ACSK, the 6–90 kHz bandstop filtering has been omitted for simplicity's sake, with filtering of only lowpass frequencies below 215 kHz using a specific combination of two FIR filters. An additional 860 kHz lowpass filtering unit is used within the FM demodulator.

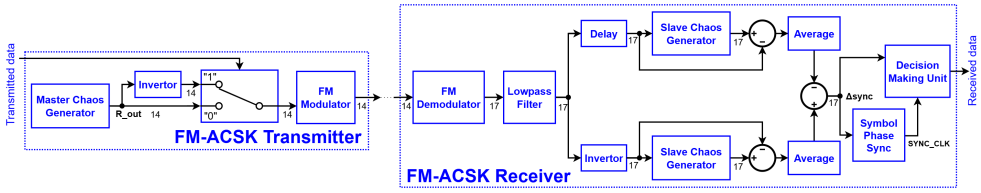


Fig. 3.1. Overall architecture of the FM-ACSK communication system's transmitter and receiver.

3.2 Frequency Modulation

The FM modulator is implemented digitally via an NCO (see Fig. 3.2). Initially, the ACSK signal is interpreted as a 14-bit signed integer, which essentially multiplies the value by the 2^8. This is done for the sake of design convenience, as the NCO expects integer values at the phase increment port input. The deviation increment is a scaling coefficient that, when multiplied by the modified ACSK and added or subtracted from the carrier increment value, ensures the desired deviation of 645 kHz. The NCO internal parameters and carrier increment value are the same as those described for the AM modulator in the previous section. As with the AM modulator, a passband frequency of 10.7 MHz is used, which is a common intermediate frequency in conventional FM communication systems. Given that the maximum ACSK frequency employed in the demodulation is 215 kHz, the deviation of 645 kHz was selected, thereby resulting in a frequency modulation index value of 3 and a 1720 kHz bandwidth for the FM-ACSK transmitted signal, in accordance with Carson's Rule.

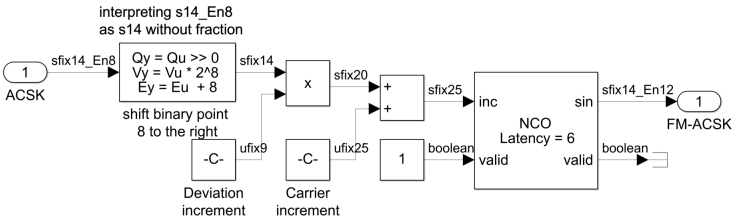


Fig. 3.2. Simulink diagram of the FM modulator unit.

The digital FM demodulator, as illustrated in Fig. 3.3, employs a 30-tap FIR Hilbert filter along with a corresponding delay – equivalent to half the Hilbert filter's group delay plus its inherent latency – to form a complex analytic signal from the noise-impacted FM-ACSK input. This analytic signal is then multiplied by a locally generated carrier signal from an NCO, mirroring the one in the FM modulator. This mixing operation shifts one copy of the FM-ACSK analytic signal to the baseband while moving the other copy to twice the carrier frequency. A 100-tap FIR lowpass filter with a cutoff frequency of 860 kHz, which represents half of the FM-ACSK bandwidth, is employed to suppress the high-frequency replica and mitigate a portion of the channel noise. The instantaneous phase of the resulting complex baseband signal is extracted by computing its angle using a unit from the Simulink HDL Toolbox. Recovery of the modulating ACSK signal, which corresponds to the instantaneous frequency, is achieved through a straightforward two-tap differentiator (see Fig. 3.4), which also eliminates any constant offset stemming from carrier phase mismatches. Phase unwrapping is applied where required, adding or subtracting 2π for values exceeding π or falling below $-\pi$, respectively. To prevent overflows caused by intense noise, hard limiting is imposed on signal excursions beyond the anticipated range dictated by the frequency deviation. Finally, the signal is amplified to make the power level at the ACSK demodulator input similar to the level at the ACSK modulator output.

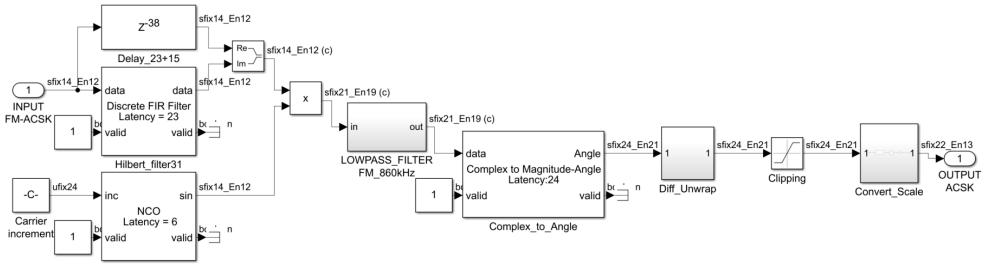


Fig. 3.3. Simulink diagram of the FM demodulator unit.

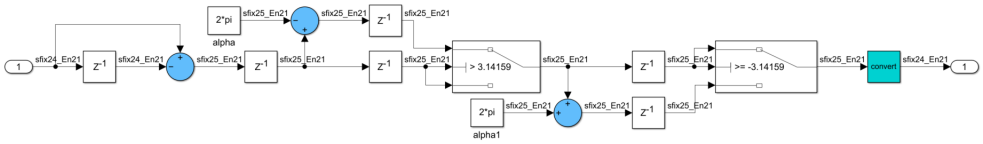


Fig. 3.4. Simulink diagram of the differentiator filter and phase unwrapping within the FM demodulator.

3.3 Filters

Prior to forwarding the reconstructed ACSK signal to the ACSK demodulator, supplementary filtering via a 215 kHz lowpass FIR filter is performed to further attenuate residual channel noise. In the FM-ACSK system, the process of filtering is executed by a cascaded arrangement of two lowpass FIR filters (see Fig. 3.5), designated as "RAISED" and "FIX" within the design framework. The objective of this approach is to achieve a satisfactory performance in FIR filtering while minimizing the utilization of hardware resources.

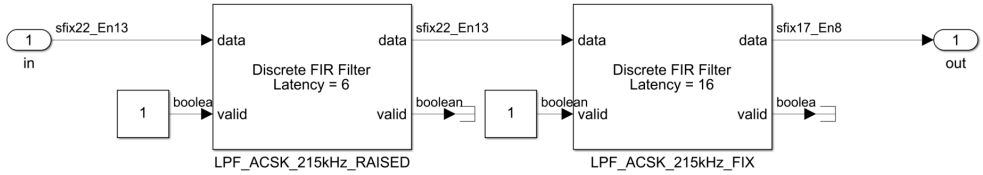


Fig. 3.5. Simulink diagram of the lowpass 215 kHz FIR filter general structure after FM demodulator.

The RAISED FIR filter concept is based on the idea that a filter designed for a cutoff frequency closer to half the Nyquist rate (F_N) requires a lower order than a filter with the same constraints and cutoff slope would require at the edges of the processable frequency span (0 and $F_N = 25\,\text{MHz}$, in this case). One existing approach to address this disadvantage is to employ digital downsampling and upsampling. However, RAISED FIR filter avoids that with a much less complex approach. The initial filter is designed at the higher cutoff frequency, specifically at the target cutoff frequency $F_c = 215\,\text{kHz}$ multiplied by an integer shift coefficient RAISE $= 7$. Subsequently, a RAISE $- 1 = 6$ zeros are appended between every coefficient of the initial filter, thereby providing the coefficients for the RAISED FIR filter. This adjustment effectively resets the cut-

off frequency to F_c, while introducing multiple narrow bandpass regions surrounding frequencies $1 \times F_s$/RAISE, $2 \times F_s$/RAISE, $3 \times F_s$/RAISE, and so forth. The selected RAISE coefficient merely adjusts the initial cutoff frequency from 215 kHz to 1.505 MHz. An increase in this coefficient would offer only a negligible improvement in reducing the initial filter order. However, this would also introduce a disadvantage characterized by an augmented number of zero coefficients, which would continue to exert a certain degree of influence on the utilization of hardware resources.

The FIX FIR filter is a simple low-order lowpass FIR filter whose stopband region begins at the first bandpass region of the RAISED FIR filter, which is located at a frequency near $1 \times F_s$/RAISE $- F_c$. This guarantees that all of the bandpass regions above baseband, introduced by the RAISED FIR filter, are negated to a level of at least the selected stopband attenuation of 60 dB.

The design of the filter architectures is defined by the Discrete FIR filter from the Simulink's DSP HDL Toolbox library, where the fully parallel transposed architecture is selected for the RAISED FIR filter and the fully parallel systolic architecture is chosen for the FIX FIR filter. While parallel systolic architecture is a favorable option for the conventional FIR filter design, as it facilitates the effective utilization of the FPGA DSP blocks, the parallel transposed architecture offers superior hardware optimization (i.e., it reduces the number of registers employed) for the RAISED FIR filter design.

The RAISED FIR filter of order 735 exhibits a baseband frequency cutoff slope that is identical to that of a regular FIR filter of the same order, both designed under identical constraints. The primary distinction lies in the fact that the RAISED FIR filter, among all 735 coefficients, has only 104 non-zero coefficients, and the additional passband regions are canceled with a relatively small FIX FIR filter of the order 17.

The frequency responses of the RAISED and FIX FIR filters, as well as their convolution that represents the overall frequency response of the cascaded two-part lowpass 215 kHz FIR filter for the ACSK signal, are illustrated in Fig. 3.6. While the minimum stopband attenuation of 60 dB is preserved with the two cascaded FIR filters, most of the higher frequencies experience a double attenuation of 120 dB. The RAISED FIR filter introduces only three baseband regions above baseband, centered at 7.14 MHz, 14.29 MHz, and 21.43 MHz. It can be generally stated that for even RAISE values, there will be RAISE/2 bandpass regions situated above baseband and below F_N, with the final bandpass region located at F_N. Additionally, the frequency response spectrum of such a RAISED FIR filter exhibits symmetry about $F_N/2$. With odd RAISE values, there will be (RAISE-1)/2 bandpass regions above baseband, as demonstrated by the present design $(7-1)/2 = 3$. In this case, the frequency response spectrum is asymmetric about $F_N/2$ and does not have a bandpass region at F_N.

The frequency response of the entire lowpass 215 kHz FIR filter within the initial 500 kHz range is illustrated in Fig. 3.7. The dotted line indicates the idealized filter frequency response, exhibiting a 200 kHz over 60 dB slope from the F_c. The design constraint for the passband peak-to-peak ripple is set to approximately 0.2 dB to ensure minimum spectral distortion of the ACSK signal. The

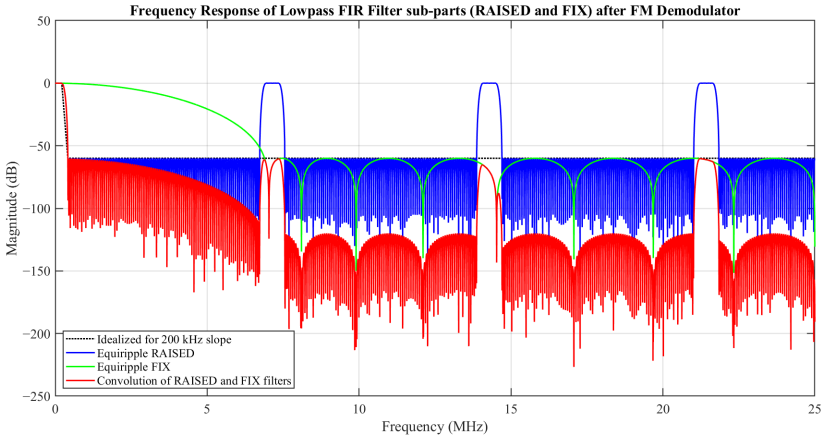


Fig. 3.6. Frequency response of the lowpass 215 kHz FIR filter's cascaded sub-filters (RAISED and FIX) and of their convolution, in the full frequency span.

drawback of this constraint is the relatively slow transition between passband and stopband, where 3 dB attenuation is achieved only at 279 kHz.

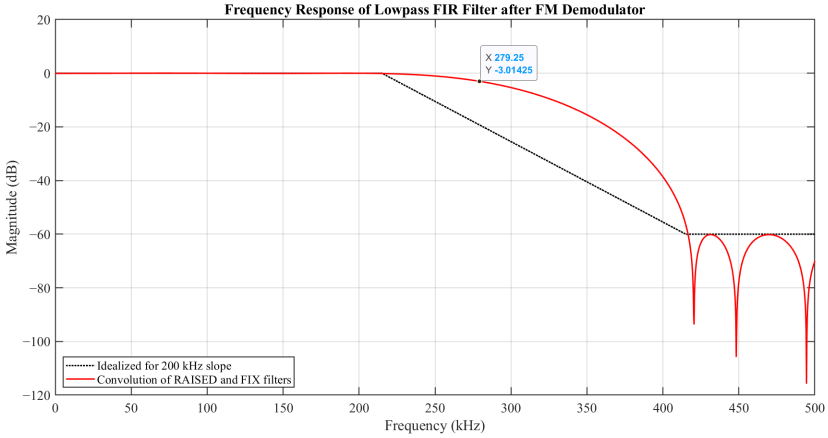


Fig. 3.7. Frequency response of the lowpass 215 kHz FIR filter in the first 500 kHz frequency span.

The lowpass FIR filter in the FM demodulator unit for the cutoff frequency of 860 kHz is designed by the same approach as the 215 kHz FIR filter described above. The stopband attenuation and passband ripple constraints, as well as the RAISE coefficient, are identical. The number of non-zero coefficients in the 860 kHz RAISED FIR filter (of order 693) is 98, and the corresponding FIX FIR filter has an order of 27. However, it should be noted that this filter requires approximately twice the hardware resources due to the processing of a complex signal. In FPGA design, the imaginary and real parts are represented as separate signals and are filtered accordingly.

3.4 Conclusions

This section was devoted to describing the FM-ACSK chaotic communication system transceiver, including its architectural similarities and modifications relative to the AM-ACSK system, design principles, and digital implementation on FPGA hardware. The key components, such as the FM modulator and demodulator units, along with the redesigned lowpass filtering system, were examined in detail. The performance evaluation of the system will be addressed through experimental verification in the subsequent sections. Table 3.1 provides a comprehensive summary of the parameters of the designed and FPGA-implemented novel FM-ACSK discrete chaotic communication system.

Table 3.1

Overview of the FM-ACSK chaotic communication system design parameters

Parameter	**Value**
chaos oscillator	modified Chua
chaotic synchronization	error linear feedback
discrete ODE solving method	forward Euler
fixed-point bit width for chaos oscillator (total, fraction)	14, 8 (master) 17, 8 (slave)
FPGA chip family used	Altera/Intel Cyclone V
clock rate and sampling frequency	50 MHz
samples per data transfer bit	8192
data transfer rate	6.1 kbps
channel bandwidth	1.720 MHz
FM modulation index	3
passband carrier frequency	10.7 MHz

The primary outcomes of this section are as follows.

- The replacement of AM with FM modulation.
- The elimination of AGC due to FM demodulator output power independence.
- The redesign of the filtering system to prioritize lowpass FIRs.
- The formulation of the FM modulator via NCO for enhanced flexibility.
- The elaboration of the FM demodulator using Hilbert filtering and analytic signal mixing.
- The development of cascaded FIR filters (RAISED with zero-inserted coefficients and FIX for artifacts) for 215 kHz cutoff frequency.
- The FPGA implementation of FM-ACSK chaotic communication system described in this section confirms the second thesis stated in the introduction.

4 PERFORMANCE EVALUATION OF THE CHAOTIC COMMUNICATION SYSTEM MODELS USING SIMULATION

To assess the performance of the AM-ACSK and FM-ACSK chaotic communication system transceivers in the Simulink environment, AWGN channel noise performance tests are conducted for both systems, with an additional selective fading channel noise performance test for FM-ACSK. This section describes the experimental setups and results of these simulation tests.

4.1 Simulink Model Experimental Setup

The configuration of the Simulink model was briefly addressed in subsubsection 2.1.4, while the present subsection focuses on describing the general structure of the Simulink model experimental setup for both systems, as illustrated in Fig. 4.1. Initially, input data bits are generated using a Bernoulli-distributed random binary generator unit from the Simulink library with a timing of one data bit per $T_b = 8192$ samples. A DAC-Channel-ADC unit is situated between the transmitter and the receiver (AM-ACSK or FM-ACSK). This unit serves to replicate the approximate attenuation and delay characteristics present within the FPGA prototype, which is connected via a 62 cm coaxial cable with a 10-sample delay and 0.35 gain. This is done to ensure greater consistency between the simulation and FPGA prototype tests of the chaotic communication systems.

AWGN is appended to the signal just before the AM-ACSK/FM-ACSK receiver, utilizing an AWGN unit obtained from the Simulink library. The SNR parameter of the AWGN unit is configured via the MATLAB script. The multipath filter unit is only used in selective fading tests and is described in the next subsection.

The output processing unit sends the transmitted and received data bits to the MATLAB workspace memory. This information is subsequently processed using a MATLAB script that measures the BER. The script also controls execution of the entire Simulink model and saves the results in a file.

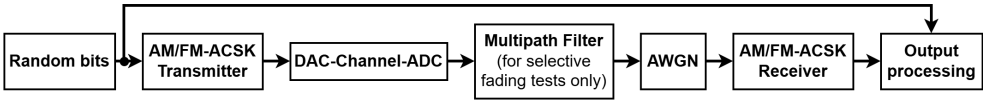


Fig. 4.1. Experimental setup general structure for Simulink performance evaluation tests for both chaotic communication systems.

4.1.1 Selective Fading Channel

The objective of this experimental test is to replicate a basic multipath scenario in a wireless link and assess the system's resistance to noise under various selective fading conditions. This basic

model incorporates only two signal paths: a direct line-of-sight route and a single reflected path (as shown in Fig. 4.2). The chosen delays are fixed to create a time-invariant selective fading effect, with the resulting notch – arising from phase interference – aligned at the carrier frequency of the FM-ACSK signal, representing a challenging worst-case scenario.

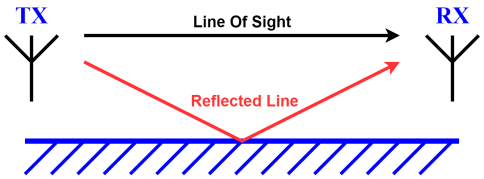


Fig. 4.2. Schematic representation of a two-path propagation model for a selective fading channel.

A straightforward FIR filter was implemented to emulate this two-path, time-invariant selective fading channel (as depicted in Fig. 4.3). The delay parameter τ denotes the time lag of the reflected signal relative to the direct one at the receiver. A delay of 21 samples was selected (with each sample lasting 20 ns at a 50 MHz sampling rate), which determines the fading notch's bandwidth and positions it near the carrier frequency (illustrated in Fig. 4.4). The gain parameter α accounts for losses due to reflection. In the experiment, six distinct gain settings were tested, yielding varying depths of notch attenuation. In order to maintain consistent signal power at the AWGN injection point – thus simplifying SNR computations – a gain adjustment is applied post-FIR filtering. The specific gain (α) and the output correction values are summarized in Table 4.1, with rounding applied to conform to the system's signed 14-bit fixed-point format (12 bits allocated to the fractional part).

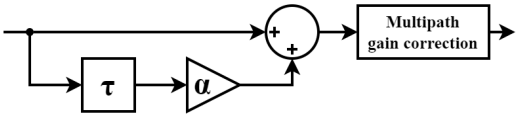


Fig. 4.3. Configuration of the multipath filter, featuring an FIR component for two-path selective fading and subsequent gain normalization.

Table 4.1

Parameters for the multipath filter

Notch attenuation depth, dB	Gain (α)	Multipath gain correction
1.5	0.2919921875	1.292724609375
3.0	0.498779296875	1.548583984375
4.5	0.700927734375	1.76123046875
6.0	0.7490234375	1.79052734375
7.5	0.822265625	1.811279296875
9.0	0.874267578125	1.806884765625

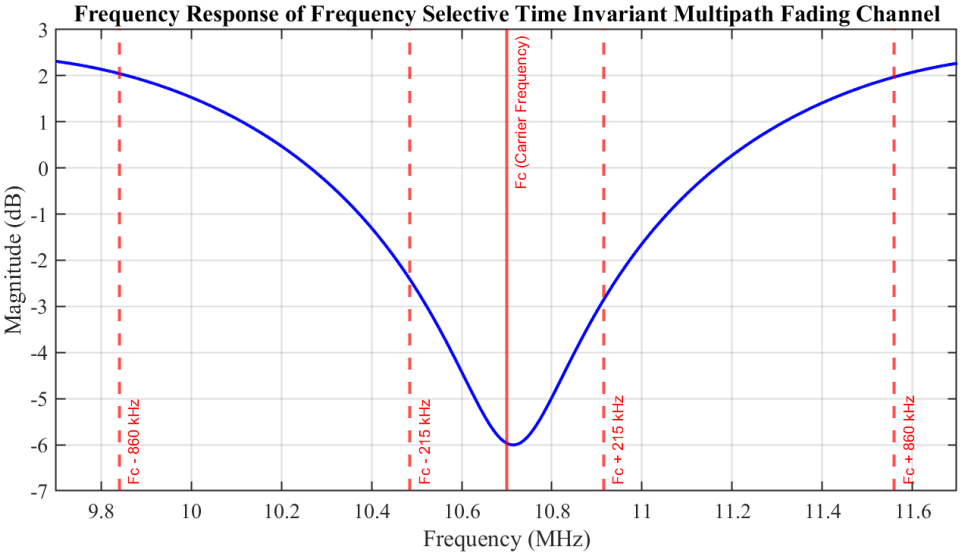


Fig. 4.4. Magnitude response of the selective fading FIR filter exhibiting a 6 dB notch depth.

It is important to note that while the interference between the direct and reflected signals induces fading around the carrier frequency, the channel also slightly boosts frequencies that are further away from the carrier frequency. In the case of the FM-ACSK model, the positive gain values commence from the offset of the carrier by a margin of approximately half the frequency deviation from the carrier. To provide an illustrative example, Fig. 4.5 compares the spectrum of an FM-ACSK signal before and after passing through the multipath filter configured for a 6 dB notch depth.

4.2 AM-ACSK Simulink Model AWGN Channel Performance Evaluation

As shown in Fig. 4.6, the simulated AM-ACSK system reaches a BER below 10^{-3} at approximately 7 dB SNR and continues to improve almost monotonically with increasing SNR, falling toward the 10^{-4}–10^{-5} region by roughly 9–10 dB. The simulated curve is smooth and nearly log-linear over the mid-to-high SNR range, indicating stable performance and predictable noise sensitivity for the idealized model.

4.3 FM-ACSK Simulink Model Performance Evaluation

This section presents the simulation results of performance tests conducted on the AWGN channel and selective fading channel for the FM-ACSK chaotic communication system MATLAB Simulink model.

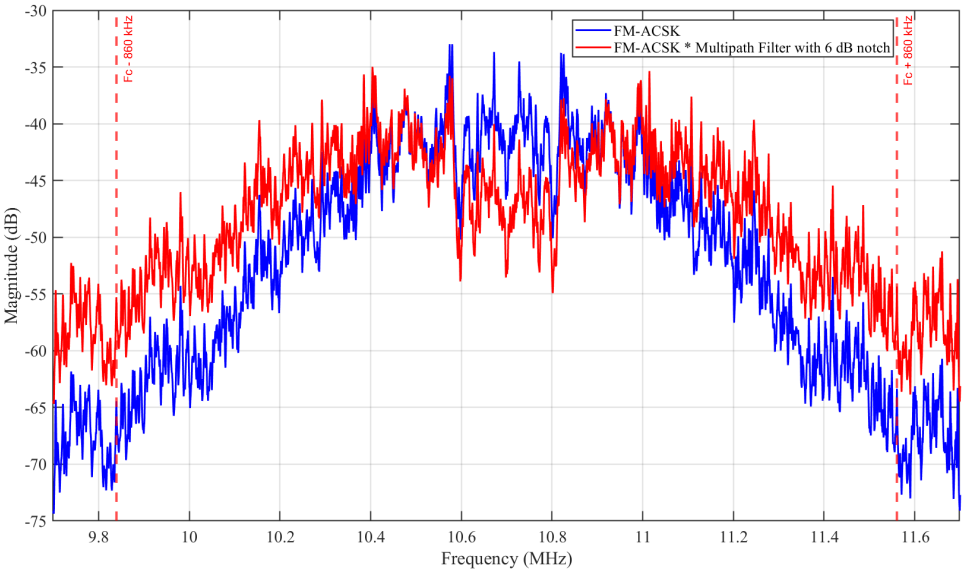


Fig. 4.5. Spectral comparison of the FM-ACSK transmitted signal prior to and following application of the multipath filter with a 6 dB selective fading notch depth.

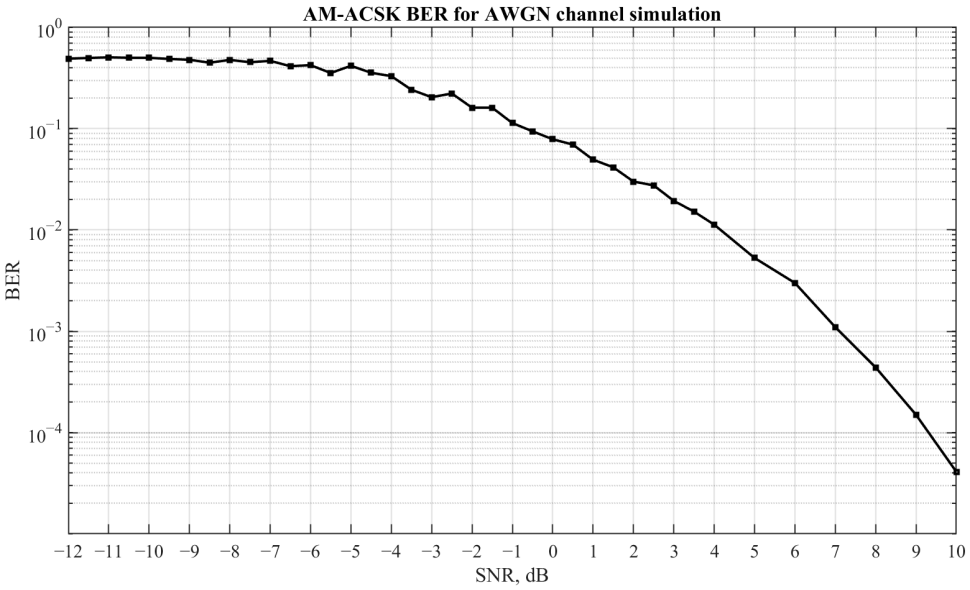


Fig. 4.6. BER for the Simulink simulation of the AM-ACSK system with an AWGN channel.

4.3.1 AWGN Channel Performance

The MATLAB Simulink simulation results for the FM-ACSK chaotic communication system, as depicted in Fig. 4.7, exhibit a smooth BER curve under AWGN channel conditions, showcasing strong noise resistance with BER dropping from approximately 0.5 at SNR values below -3 dB to

near 10^{-4} at 5.6 dB. Simulations employed 2000 data bits transmitted for most points and 20,000 bits for the lowest four BER values to improve precision at higher SNR, where errors are rare. The curve remains flat around 0.5 BER from -7 dB to -3 dB, indicating a threshold effect, before declining steeply – e.g., from ~ 0.3 at -1 dB to ~ 0.001 at 5 dB – with a decelerating slope from over 2 dB per decade initially to under 1 dB per decade at elevated SNR, supporting accurate noise performance forecasting. Additional testing with denser SNR sampling and larger bit volumes might reveal minor BER slope variations, though the current data shows no such anomalies.

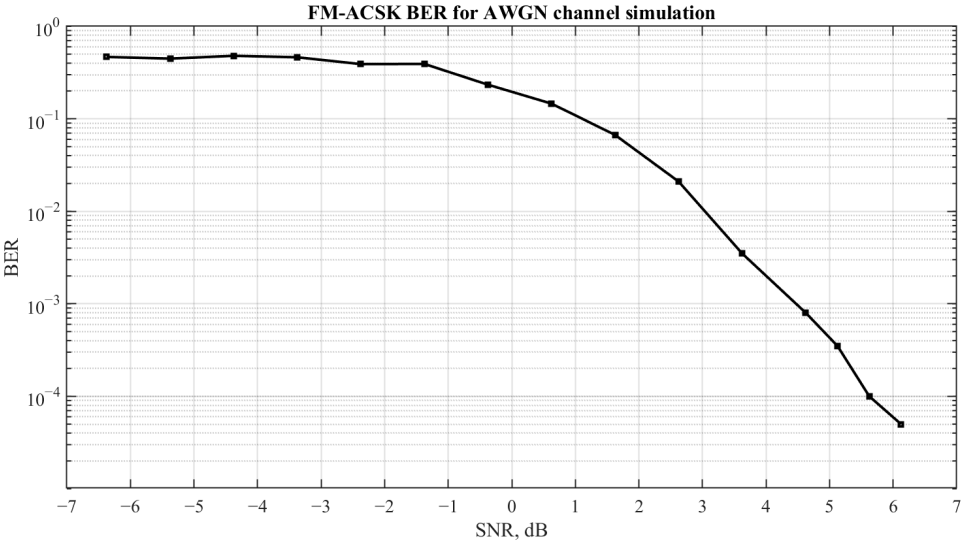


Fig. 4.7. BER for the Simulink simulation of the FM-ACSK system with an AWGN channel.

4.3.2 Selective Fading Channel Performance

Figure 4.8 presents the BER simulation results for different depths of selective fading notches across various SNR levels, which are published in [14]. At elevated SNR values (exceeding 2–3 dB), the system's performance with shallow notches of 1.5 dB and 3.0 dB remains comparable to that in the absence of fading. However, deeper notches result in substantial BER degradation due to the heightened distortion in the FM-ACSK signal's spectrum.

Conversely, under conditions of high noise (SNR below 2–3 dB), the presence of selective fading yields marginally superior performance compared to non-fading scenario. This enhancement is attributable to the channel's amplification of critical sideband frequencies that carry information, positioned beyond approximately half the deviation from the carrier. While this effect is discernible in BER plots, its practical utility is constrained, as operating at BER levels above 10^{-2} poses significant challenges, even with sophisticated forward error correction (FEC) techniques.

In order to elucidate the impact on the transmitted ACSK signal under diverse fading and noise conditions, Fig. 4.9 compares the spectrum of the original ACSK signal prior to FM modulation with

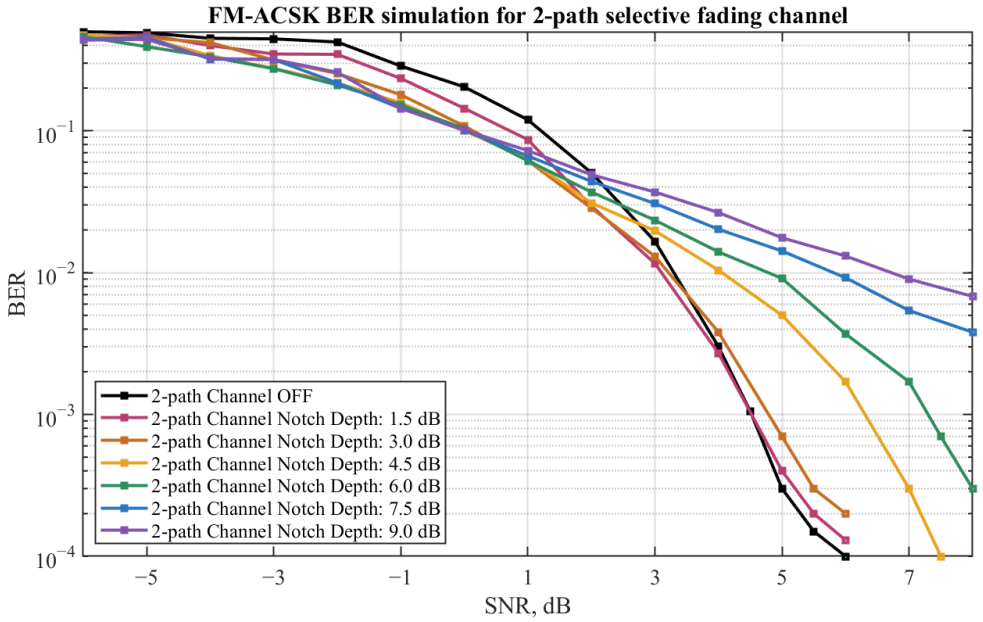


Fig. 4.8. BER curves for varying two-path selective fading notch depths at different SNR values in MATLAB Simulink simulation.

the recovered version post-FM demodulation and filtering. In this low-noise example (8 dB SNR) with a moderate 6 dB notch depth, the restored spectrum demonstrates minimal noise influence but exhibits slight frequency distortions from the fading notch, sufficient to cause 3 errors per 10,000 bits that were observed during test.

The subsequent examples underscore the recovery of ACSK spectra in the presence of high noise (0 dB SNR): without fading (Fig. 4.10) and with a 6 dB notch (Fig. 4.11). In both instances, noise overwhelms lower-amplitude frequencies; however, key ACSK bands near 170 kHz and DC persist partially. It is noteworthy that the notched case amplifies the restored frequencies slightly higher (above -10 dB at 170 kHz) than the non-faded scenario (below -10 dB), as compared to the original's near -5 dB peak. This contributes significantly to demodulation resilience in noisy environments.

4.4 Conclusions

This section described the performance evaluation of the chaotic communication system mathematical models using simulation in the MATLAB Simulink environment. The simulation testing setup for both AM-ACSK and FM-ACSK system transceivers was described in detail, including the definition of the selective fading channel model, followed by BER analyses performed in AWGN (for both systems) and two-path selective fading (for FM-ACSK) channels across a wide range of SNR

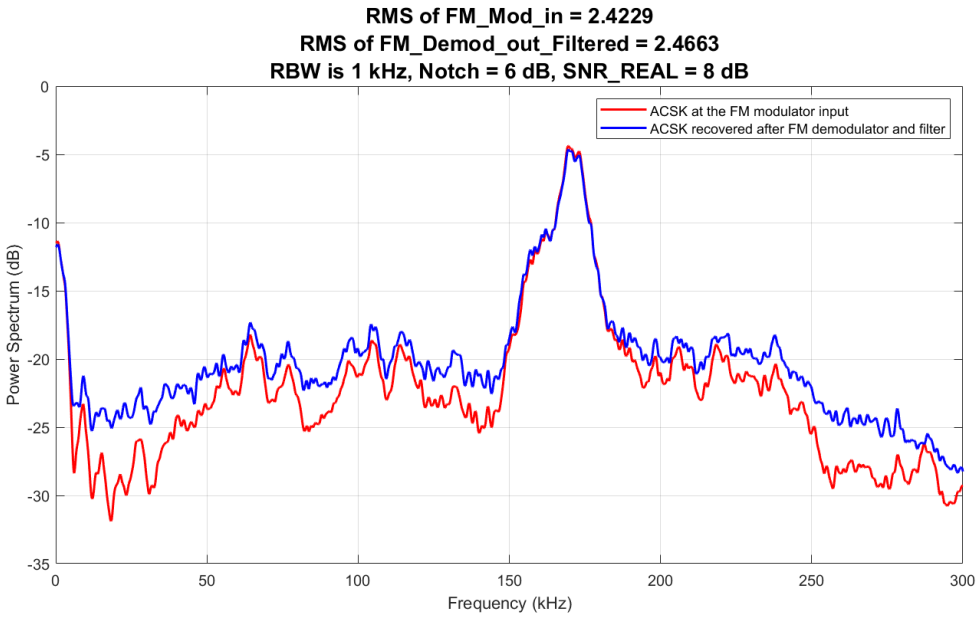


Fig. 4.9. Spectral comparison of the ACSK signal at the FM modulator input (red) versus the recovered ACSK after demodulation and filtering, under 8 dB SNR and 6 dB notch depth (blue).

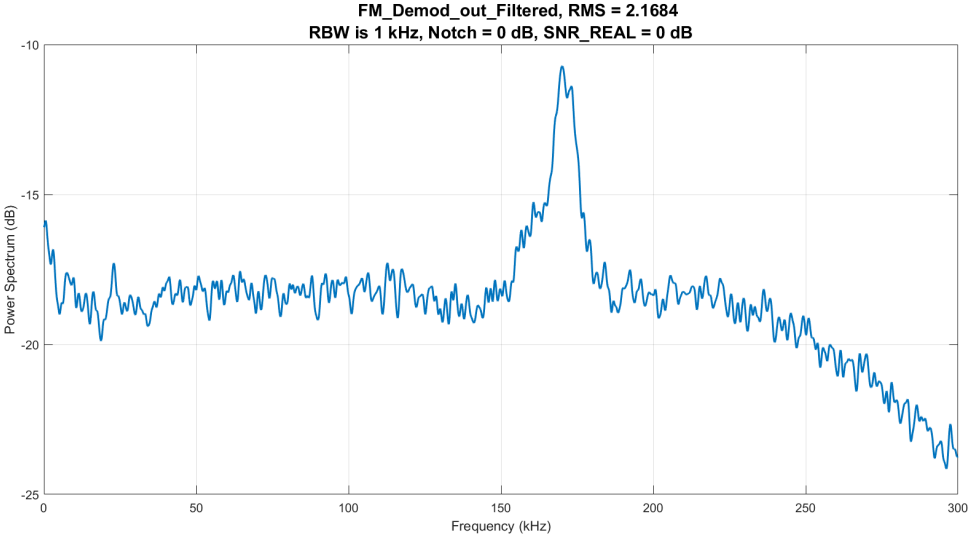


Fig. 4.10. Spectrum of the recovered ACSK post-FM demodulation and filtering, at 0 dB SNR without selective fading.

values.

The primary outcomes of this section are as follows.

- The development of a Simulink mathematical models that replicate the digital signal process-

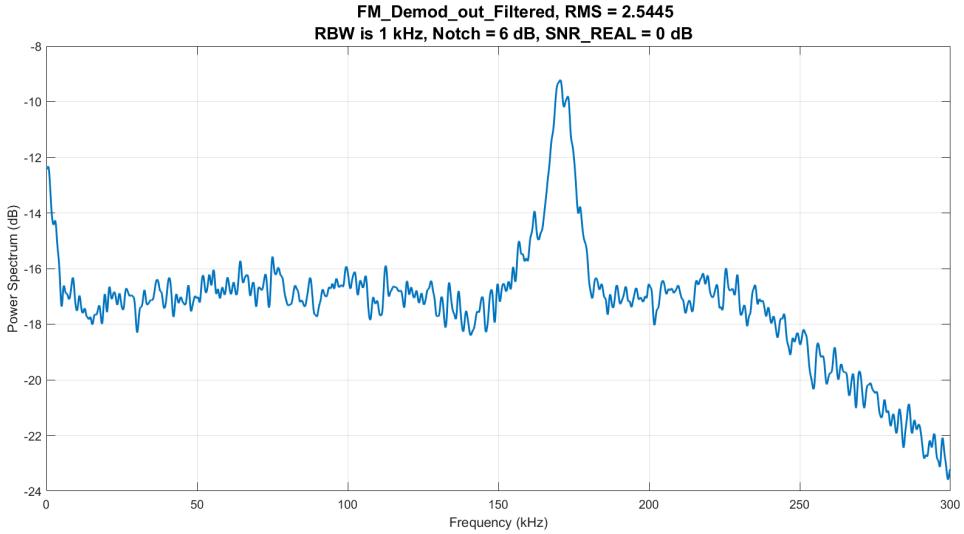


Fig. 4.11. Spectrum of the recovered ACSK post-FM demodulation and filtering, at 0 dB SNR with 6 dB notch depth.

ing chains of the FPGA prototypes, which enables direct comparison between the simulation and hardware results.

- FM-ACSK's model superior noise immunity in AWGN channels, attributable to its constant envelope property and broader spectrum occupancy.
- Quantitative BER characterization showing that FM-ACSK model reaches a target of 10^{-3} BER at approximately 4.4 dB SNR, while AM-ACSK requires 7.1 dB SNR under identical conditions.
- Detailed analysis of two-path selective fading impact on FM-ACSK system demonstrates that shallow notches (1.5–3.0 dB) cause negligible degradation at high SNR, whereas deeper notches progressively impair performance.

An overview of some noteworthy BER performance metrics derived from the Simulink models of the novel AM-ACSK and FM-ACSK discrete chaotic communication systems is presented in Table 4.2.

Table 4.2

BER performance metrics for AM-ACSK and FM-ACSK Simulink models

BER	**≈ SNR for AM-ACSK**	**≈ SNR for FM-ACSK**
10^{-1}	−0.9 dB	1.0 dB
10^{-2}	4.1 dB	3.0 dB
10^{-3}	7.1 dB	4.4 dB
10^{-4}	9.3 dB	5.6 dB

5 PERFORMANCE EVALUATION OF THE CHAOTIC COMMUNICATION SYSTEM FPGA PROTOTYPES

In a manner consistent with the preceding section, this section examines the performance of AM-ACSK and FM-ACSK for AWGN channels in the absence and the presence of selective fading channels. These evaluations are conducted on the FPGA prototypes, which offer a more comprehensive understanding of the practical performance characteristics of the chaotic communication systems. The experimental configuration for the AM-ACSK and FM-ACSK FPGA transceiver prototypes necessitated the design of multiple supplementary signal processing units. The details of these units are addressed below.

5.1 FPGA Prototype Experimental Setup

The FPGA prototypes were developed using the Arrow SoCKit board (see Fig. 5.1), which is equipped with an Altera (Intel) Cyclone V 5CSXFC6D6F31C6N FPGA chip. The establishment of the transmitter-receiver link is facilitated by an analog channel, employing DAC and ADC on the Terasic THDB-ADA daughterboard. This configuration is connected to the SoCKit through a HSMC interface. While the AM-ACSK/FM-ACSK systems can be deployed on various FPGA devices, smaller chips might necessitate substantial optimizations to manage resource constraints. The transmitter and receiver are co-located on a single FPGA chip; however, the use of separate chips is also a viable option.

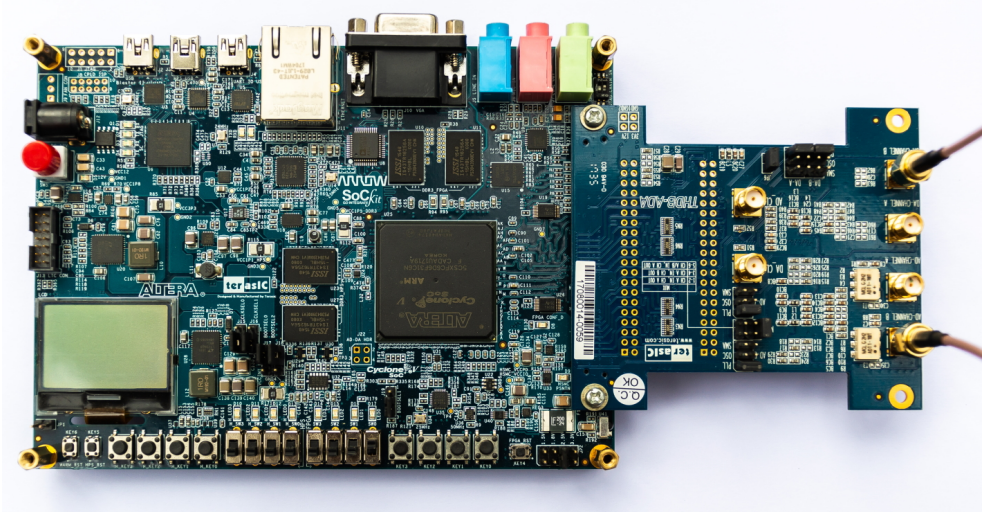


Fig. 5.1. The photograph of the FPGA chaotic communication systems prototype setup on an Arrow SoCKit development board with a Terasic THDB-ADA daughter board and a coaxial cable for analog signal transmission.

The Cyclone V system-on-chip (SoC) exhibits a range of advanced features that extend beyond the fundamental capabilities of FPGAs. It incorporates a hard processing system (HPS) that contains an Arm Cortex-A9 microprocessor unit (MPU) and peripheral controllers. This facilitates the seamless integration of FPGA signal processing with MPU-based software operating on a dedicated Linux platform. The Arrow SoCKit board facilitates the access of peripherals such as clocks, buttons, switches, LEDs, synchronous dynamic random-access memory (SDRAM), and a USB-to-UART bridge, all of which are utilized in the test configuration.

Figure 5.2 provides a high-level view of the experimental setup. Green elements denote custom-designed FPGA modules, blue elements represent Altera/Intel intellectual property (IP) cores from the Quartus library, red blocks indicate integrated HPS modules in the Cyclone V SoC, yellow blocks are external integrated circuits (ICs) on the SoCKit and on the daughterboard, and white blocks encompass miscellaneous external elements. For reference, a full, structured list of VHDL files for the green blocks of FM-ACSK system's experimental setup is provided in Appendix A, where green files indicate scripts that were written manually, orange files – manually designed in Simulink and generated into VHDL scripts using Simulink's HDL Coder, and blue files are sub-modules automatically designed with Simulink according to desired set of parameters.

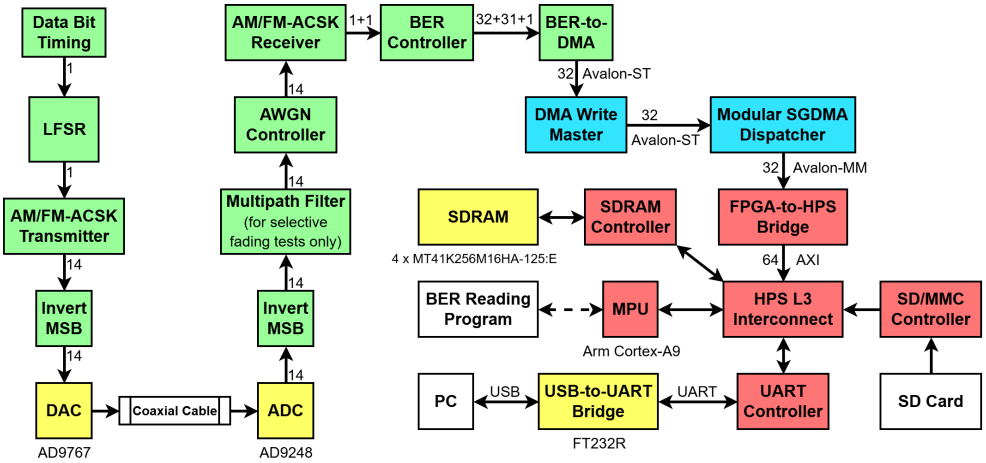


Fig. 5.2. Simplified schematic of the AM-ACSK/FM-ACSK FPGA prototype's overall experimental configuration.

In essence, the AM-ACSK/FM-ACSK transmitter generates chaotic symbols (8192 samples per modulating bit) from a pseudorandom binary data input sequence. The output is converted to analog via DAC, transmitted over a 62 cm coaxial cable to ADC, and digitized back to a 14-bit signal. Converters are designed to process unsigned signals; therefore, the MSB is inverted both prior to the DAC (where two's complement is converted to offset binary) and after the ADC (where the reverse process occurs). Subsequently, channel noise and selective fading (when necessary) are added to the signal before it is processed by the AM-ACSK/FM-ACSK receiver. The recovered bits are then

utilized for BER computation, with the results transmitted to the PC for display, through the HPS.

Beyond the core AM-ACSK/FM-ACSK system (the transceiver), additional FPGA modules were developed for the purpose of evaluating the system's performance (see Fig. 5.2). The following elements are included: data bit timing, linear feedback shift register (LFSR), AWGN controller, BER controller, and BER-to-DMA.

The data bit timing module is a 13-bit counter outputting "1" every $2^{13} = 8192$ cycles, matching the data bit/symbol period (T_b) in the AM-ACSK/FM-ACSK system. The LFSR module produces a pseudorandom sequence via a 16-bit maximum-length Fibonacci structure with polynomial $x^{16} + x^{14} + x^{13} + x^{11}$ [88], approximating random data for transmission and aiding BER evaluation.

The AWGN controller is responsible for the addition of noise and the management of SNR. It adapts MathWorks' "HDL Implementation of AWGN Generator" model [89] with adjustments such as bus width modifications and real-only output. The model generates AWGN for SNR relative to 1 W (0 dBW) in 0.1 dB steps. The actual SNR values are determined by considering the input signal power, which is -12.125556852733562 dBW for FM-ACSK and -17.755585122847151 dBW for AM-ACSK, in relation to 0 dBW. Furthermore, the bandwidth of the transmitted signal is taken into consideration, which is 1.72 MHz for FM-ACSK and 0.43 MHz for AM-ACSK, relative to the Nyquist frequency of 25 MHz. Real-time SNR control utilizes onboard switches and buttons of the SocKit to select one of 127 preset states, with a 0.2 dB variation in SNR values. Alternatively, the addition of AWGN can be disabled. The 14-bit signed fixed-point format (12 fractional bits) imposes a limit on very low SNR values, which were avoided in the tests. For more information on the AWGN controller design, a VHDL code for its implementation (FM-ACSK version) is provided in Appendix C.1.

The BER controller processes and calculates the sum of errors in the received bits, mirroring the transmitter's 16-bit LFSR plus two regular linear shift registers (LSRs): a 16-bit one that extends LFSR output and a 32-bit one that stores received bits from the AM-ACSK/FM-ACSK receiver output (see Fig. 5.3). Upon matching the 32-bit received sequence to the preloaded key sequence (combined 16-bit LFSR and LSR), the system shifts from the "INITIAL" state to the "LOCKED" state. This change is accompanied by the illumination of an LED and the enablement of registers and activation of counters for bit errors and total bits received. LFSR then initiates the generation of pseudorandom bits, which are identical to the bits generated at the transmitter input. The identification of bit detection errors is achieved through a process of comparison between the locally generated and received bits.

In addition, a "soft" reset input (board button, separate from global reset) is utilized to clear counters and reset the BER controller state to "INITIAL." This process is initiated, for instance, when restarting the error counting for a new SNR value. Locking relies on the precise reception of the key sequence, a process that becomes more challenging under conditions of low SNR. The duration of one full cycle of 16-bit LFSR-generated pseudorandom bit sequence transmission is

equivalent to $(2^{16}-1)\times T_b \times dt = 65,535 \times 8192 \times 20^{-9} = 10.7372544$ seconds. Should noise-introduced bit detection errors cause the wrong bit to fall within the key sequence, locking will be delayed until one of the next sequence cycle iterations. The 16-bit LFSR ensures short cycles; however, the 32-bit key minimizes the probability of false positives. Specifically, the probability of 16-bit key matching from stochastic noise input to the receiver is $2^{-16} = 1.526 \times 10^{-5}$, whereas for the 32-bit key it decreases to $2^{-32} = 2.328 \times 10^{-10}$. Furthermore, the 32-bit key reduces the risk of partial bit correlation. With a 16-bit key, 15 out of 16 bits correlate within the sequence 16 times. In the event that the remaining single bit were to flip as a result of erroneous detection, locking to the pseudorandom bit sequence would be invalid. However, when utilizing a 32-bit key, only 14 partial bit correlations are observed, with a mere 27 bits correlating within the sequence and no correlations evident with higher bit counts. For additional reference, VHDL code for BER controller is provided in Appendix C.2.

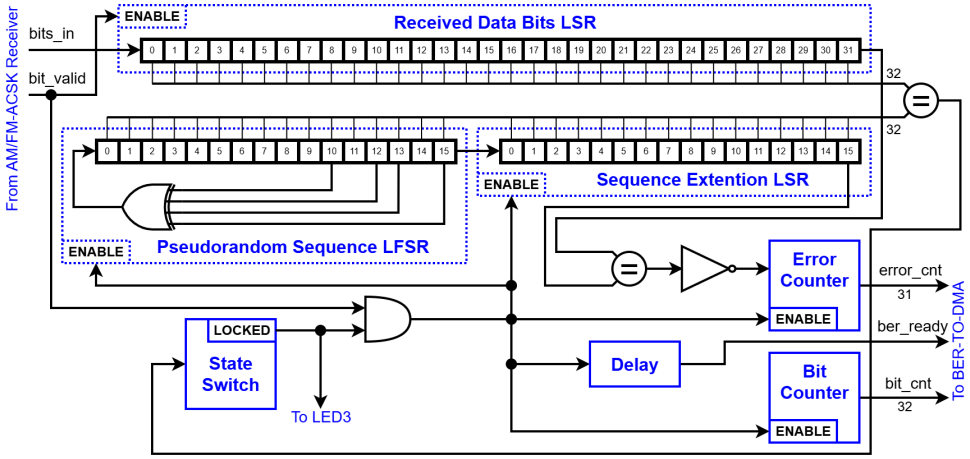


Fig. 5.3. BER controller diagram highlighting primary signals (excluding reset and clock).

The BER-to-DMA unit is responsible for transmitting bit/error counts to the direct memory access (DMA) Write Master IP core for SDRAM via the Avalon Streaming interface [90]. Subsequently, several Altera IP cores and HPS modules are responsible for the interconnection process within the SoC design. A specially designed C program (see Appendix B) running on a Linux operating system retrieves, processes, and displays BER data on a PC monitor screen, accessing the SoC through the universal asynchronous receiver/transmitter (UART) interface.

A comparison of the speeds reveals that simulating 20,000 bits in MATLAB Simulink takes approximately 10 hours on a 3.80 GHz central processing unit (CPU), whereas on an FPGA, the same task completes in 3.2768 seconds. This represents a speedup of approximately 11,000. The data rate of the FPGA is fixed at 6.1 kbps (50 MHz clock, 8192-sample symbols). This allows for the execution of high-SNR BER tests with significantly enhanced precision.

5.2 AM-ACSK FPGA Prototype AWGN Channel Performance Evaluation

This section presents the FPGA prototype experimental results of performance tests conducted on the AWGN channel for the AM-ACSK chaotic communication system, compared with the MATLAB Simulink simulation.

As depicted in Fig. 5.4, the FPGA implementation of AM-ACSK yields a BER curve that closely follows the Simulink simulation across the overlapping SNR range, confirming hardware fidelity to the model. The FPGA results extend to 16 dB SNR, achieving BER below 10^{-6} (versus the simulation limit of below 10^{-4} at 10 dB). The observed discrepancies are small and expected for a hardware implementation: analog passband channel imperfections, synchronization offsets and other nonidealities introduce additional effective noise and implementation loss that explain the ∼1–2 dB degradation.

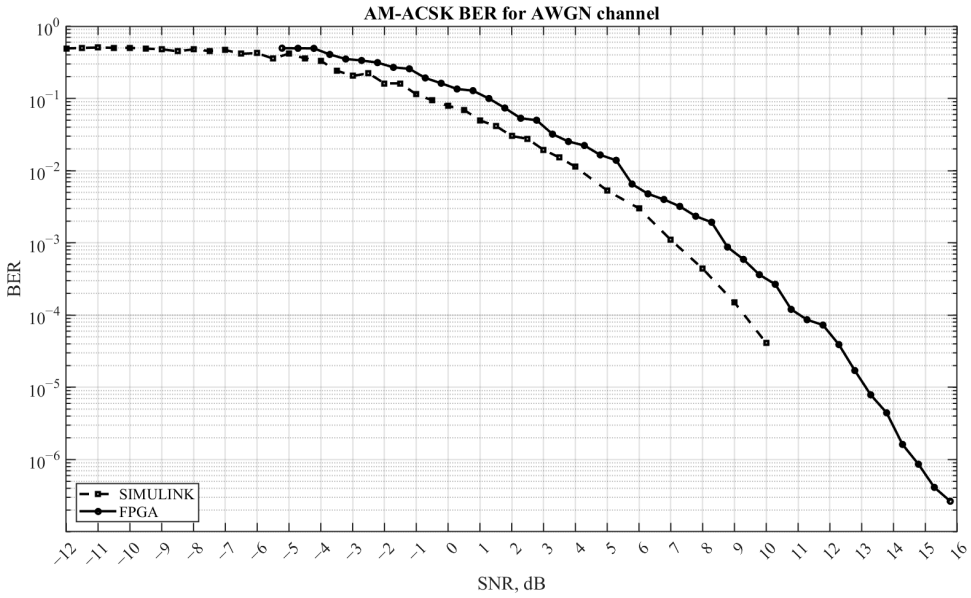


Fig. 5.4. AM-ACSK BER for AWGN channel – MATLAB Simulink and FPGA results.

5.3 FM-ACSK FPGA Prototype Performance Evaluation

This section presents the experimental results of performance tests conducted on the AWGN channel and selective fading channel for the FM-ACSK chaotic communication system FPGA prototype, compared with the MATLAB Simulink simulation.

5.3.1 AWGN Channel Performance

As illustrated in Fig. 5.5, the BER performance of the FM-ACSK chaotic communication system is demonstrated under AWGN conditions. Same as in the AM-ACSK case, this figure consists of two components: the MATLAB Simulink simulation curve, which was already presented in the prior section, and the experimental results from the FPGA prototype, which are used for direct comparison. The FPGA results demonstrate a BER curve that is somewhat less smooth than the simulation, with minor irregularities, and exhibit reduced tolerance to noise – shifted by approximately 1 dB in SNR toward higher values for equivalent BER levels. This finding aligns with expectations, given the FPGA implementation's increased complexity, which encompasses analog circuitry and real-world hardware constraints that are not present in the idealized simulation environment.

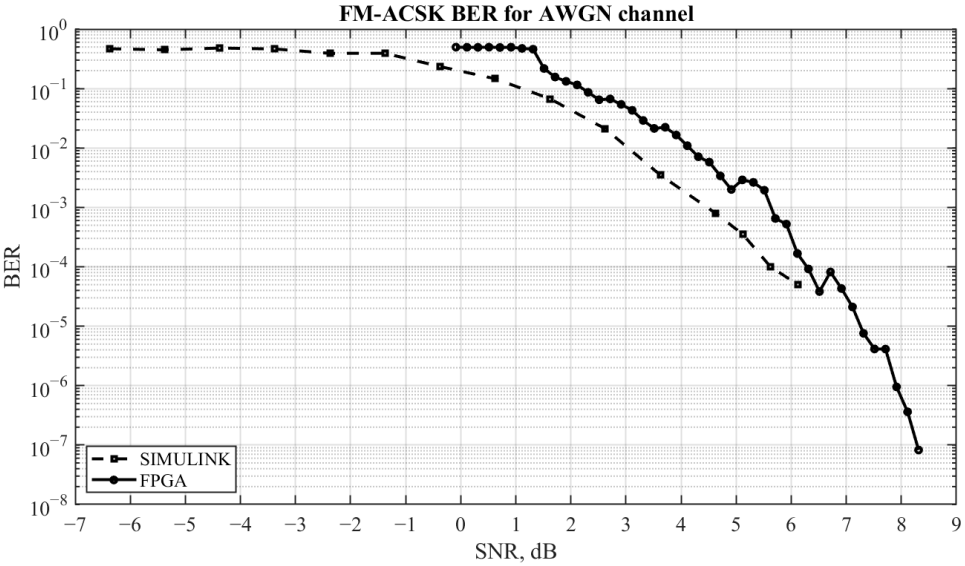


Fig. 5.5. FM-ACSK BER for AWGN channel – MATLAB Simulink and FPGA results.

The irregularities observed in the FPGA BER curve, which exhibit a recurring pattern, remain a subject of interest and necessitate further investigation. These phenomena could be attributed to quantization artifacts in the fixed-point arithmetic employed on the data bus, such as subtle shifts in bit representation that cause values to approach bus boundaries. Such phenomena tend to be subtle amid high error counts (e.g., thousands per test) but become more evident when errors are sparse (e.g., in the hundreds), as occurs at elevated SNR.

It is important to note that these fluctuations cannot be attributed to inadequate sampling in BER estimation. The FPGA experiments utilized significantly larger datasets, typically ranging from approximately 1,563,000 transmitted bits per point to as high as 473 billion bits for specific high-SNR measurements aimed at maintaining BER precision. This is substantially larger than the simulation's use of 2000 bits for standard points and 20,000 bits for the four lowest BER cases.

Conducting Simulink tests with finer SNR increments and significantly expanded bit volumes may reveal similar variations in the model.

5.3.2 Selective Fading Channel Performance

Figure 5.6 presents the FPGA prototype BER test results for varying depths of selective fading notches across a range of SNR levels. At elevated SNR values (exceeding 4–5 dB), system performance with shallow notches of 1.5 dB and 3.0 dB remains comparable to that in the absence of fading. However, deeper notches result in substantial BER degradation due to the heightened distortion in the FM-ACSK signal's spectrum. Conversely, under conditions of high noise (SNR below 2–3 dB), the presence of selective fading yields performance comparable to the non-fading scenario, with most curves overlapping closely around 10^{-1}. Exceptional behavior is demonstrated by the 7.5 dB notch depth test, which showed noticeably worse result than 9.0 dB notch depth test with irregular slope at higher noise levels.

When compared with the MATLAB Simulink simulation results of the same experiment (Fig. 4.8), the FPGA curves exhibit high level of overall agreement in trend and relative ordering of notch-depth effects (except for 7.5 dB notch depth). Both confirm that shallow notches (1.5 dB and 3.0 dB) cause negligible degradation at high SNR, while deeper notches introduce progressive impairment; the hardware prototype, however, attains more than two orders of magnitude lower BER floors and extends to higher SNR values.

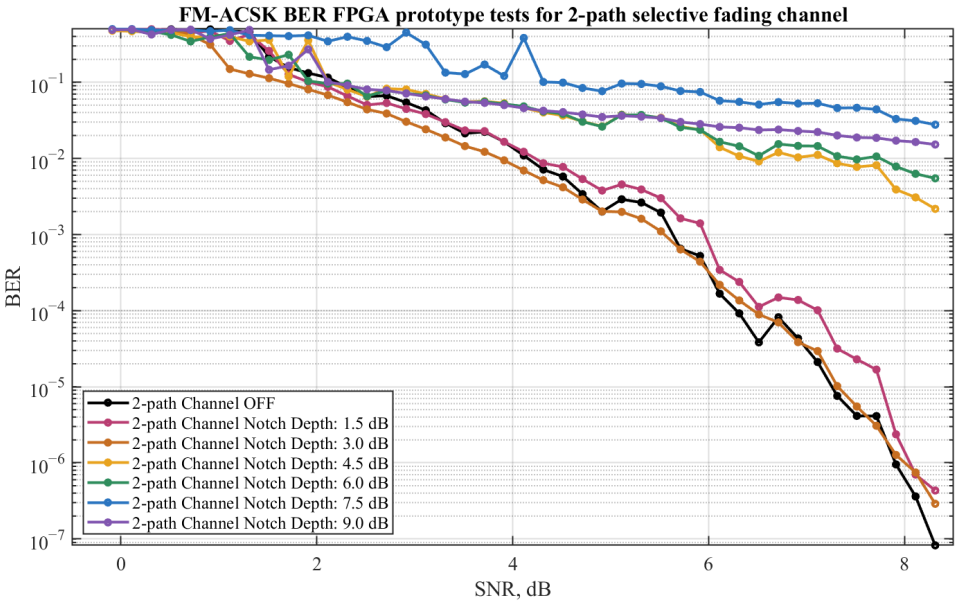


Fig. 5.6. BER curves for varying two-path selective fading notch depths at different SNR values in FPGA prototype tests.

5.4 Conclusions

This section was devoted to the performance evaluation of the chaotic communication system FPGA prototypes. The experimental setup utilizing the Arrow SoCKit board, auxiliary modules (LFSR data generator, AWGN controller, BER controller, and DAC/ADC interfacing via coaxial cable), and real-time testing methodology was presented, followed by BER measurements for both AM-ACSK and FM-ACSK transceiver systems in AWGN and two-path selective fading channels. These empirical findings provide a solid basis for anticipating the noise-handling capabilities of the implemented chaotic communication systems.

The primary outcomes of this section are as follows.

- Successful real-time operation of both prototypes on a single Altera/Intel Cyclone V FPGA at 50 MHz, confirming the practical implementability of the complete transmitter and receiver signal processing chains.
- BER results in AWGN channels that closely track the Simulink simulations, with only minor shifts attributable to fixed-point arithmetic effects and analog channel imperfections.
- Experimental validation of the FM-ACSK's superior noise performance, achieving 10^{-3} BER at approximately 5.6 dB SNR (versus 8.7 dB SNR for AM-ACSK), which serve to confirm the third and fourth theses stated in the introduction.
- Selective fading tests for FM-ACSK demonstrated overall agreement with simulation trends: shallow notches produce negligible degradation, while deeper notches introduce progressive impairment, yet the hardware prototype consistently attains lower BER floors than predicted.

An overview of some noteworthy BER performance metrics derived from experimental verifications for the FPGA prototypes of the novel AM-ACSK and FM-ACSK discrete chaotic communication systems is presented in Table 5.1.

Table 5.1

BER performance metrics for AM-ACSK and FM-ACSK FPGA prototypes

BER	**≈ SNR for AM-ACSK**	**≈ SNR for FM-ACSK**
10^{-1}	1.2 dB	2.1 dB
10^{-2}	5.5 dB	4.1 dB
10^{-3}	**8.7 dB**	**5.6 dB**
10^{-4}	11.0 dB	6.3 dB
10^{-5}	11.2 dB	7.2 dB
10^{-6}	13.2 dB	7.9 dB
10^{-7}	–	8.3 dB

OVERALL CONCLUSIONS

This Thesis presents an investigation into the development of FPGA-based digital chaotic communication transceiver systems, centered on a modified Chua oscillator combined with error-feedback synchronization and two innovative hybrid modulation schemes – AM-ACSK and FM-ACSK. These schemes integrate digital coherent baseband antipodal chaos shift keying with analog passband amplitude and frequency modulation to deliver robust physical-layer signal protection while addressing practical constraints in wireless and IoT environments. The research encompasses mathematical modeling, simulation validation, FPGA hardware prototyping, and performance benchmarking under realistic channel conditions.

The primary aim – to research the coherent discrete chaotic communication system design options and to develop a set of approaches for the FPGA implementation of such systems – has been realized. Every task set out in the Introduction was carried out: a thorough review of chaotic communication principles and prior FPGA solutions; formulation of system models using forward-Euler discretization of the modified Chua circuit's ODE in fixed-point arithmetic; proof-of-concept testing in Simulink; VHDL-based prototype development on a single Altera/Intel Cyclone V FPGA; and experimental validation through custom auxiliary modules for data generation, noise injection, and BER assessment.

The key achievements of the study can be highlighted as follows.

- A fully digital version of the modified Chua generator, together with error-feedback synchronization and ACSK modulation (fixed at 8192 samples per symbol), was realized and paired with NCO-driven AM and FM passband stages operating at a 10.7 MHz carrier. This architecture maintains chaotic behavior while fitting within the logic resources of a single Cyclone V FPGA device running at 50 MHz.
- Complete transmitter and receiver chains were implemented, incorporating digital incoherent AM and FM demodulators, 215 kHz cutoff filters for noise suppression, AGC (AM-ACSK only), early-late gate timing recovery, and integration-based bit detection. The resulting prototypes deliver a consistent uncoded data rate of 6.1 kbps within 430 kHz (AM-ACSK) and 1.720 MHz (FM-ACSK, modulation index 3) bandwidths.
- High-fidelity Simulink models mirroring the prototypes' digital signal paths enabled direct simulation comparison with hardware results, confirming close agreement once real world effects and analog-channel imperfections are accounted for.
- Comprehensive BER characterization in AWGN channel demonstrated clear performance differences: FM-ACSK exhibits noticeably better noise tolerance owing to its constant envelope and wider spectral occupancy, reaching 10^{-3} BER at 5.6 dB SNR versus 8.7 dB SNR required by AM-ACSK.

- The four theses stated in the Introduction were empirically confirmed through the realized prototypes, establishing the implementability of both chaotic communication systems on an FPGA hardware with the respective BER levels.

Both system variants were developed and assessed within a unified simulation-to-hardware workflow. Simulink served for rapid design iteration and theoretical benchmarking, while the FPGA prototypes – linked via DAC/ADC and coaxial cable – provided real-time testing under controlled AWGN and multipath conditions using LFSR-generated data and dedicated BER controller for the test data acquisition. This dual validation approach not only verified the theoretical models but also exposed the minor implementation gaps arising from quantization and analog interfacing, providing practical guidance for future refinements.

Taken together, these outcomes illustrate that coherent chaotic communication systems can be deployed on FPGA platforms and utilized for communication purposes with enhanced physical-layer signal protection. The noise-like, spread-spectrum properties of the generated carriers, combined with the strict synchronization requirement at the receiver, inherently raise the barrier for unauthorized interception, while the measured BER performance remains suitable for low-rate applications such as secure IoT sensor links, industrial monitoring, or covert tactical communications. The FM-ACSK variant stands out for its superior noise resilience, suggesting it as a strong candidate for environments with challenging propagation or interference.

In summary, this Thesis successfully fulfills its stated objective, completes all research tasks, and fully substantiates the defended theses. The developed methodologies, prototypes, and performance data establish a practical foundation for advancing FPGA-based chaotic communication systems in real-world communication scenarios.

The results of this study should be interpreted within the context of the selected system configuration, which includes low-rate uncoded transmission, one specific FPGA platform, fixed-point arithmetic, and controlled AWGN/selective fading channel conditions. Subsequent efforts are necessary to attain enhanced data transmission rates, refined synchronization mechanisms, advanced channel coding techniques, and a comprehensive evaluation of signal protection measures.

Directions for continued study include scaling data rates through higher clock frequencies and deeper pipelining level, incorporating adaptive mechanisms that respond to dynamic channel variations, exploring FEC integration for further BER improvement, and conducting detailed signal protection evaluations against modern eavesdropping and jamming techniques. Multi-user access could also widen the application scope of these chaotic communication systems. The frameworks and insights provided here offer a solid starting point for these extensions.

BIBLIOGRAPHY

[1] T. Tami et al. “Chaos secure communication’ implementation in FPGA.” In: *2018 International Conference on Applied Smart Systems (ICASS)*. 2018, pp. 1–6. DOI: 10.1109/ICASS.2018.8652046.

[2] K. Sun. *Chaotic Secure Communication: Principles and Technologies*. De Gruyter, 2016. ISBN: 978-3-11-043406-4.

[3] P. Stavroulakis. *Chaos applications in telecommunications*. CRC Press, 2006. ISBN: 978-0-203-02531-4. DOI: 10.1201/9780203025314.

[4] S. Çiçek, U. E. Kocamaz, and Y. Uyaroğlu. “Secure Chaotic Communication with Jerk Chaotic System Using Sliding Mode Control Method and Its Real Circuit Implementation.” In: *Iranian Journal of Science and Technology – Transactions of Electrical Engineering* 43.3 (2019), pp. 687–698. ISSN: 23641827. DOI: 10.1007/s40998-019-00184-9.

[5] B. Muthuswamy and S. Banerjee. *A Route to Chaos Using FPGAs*. Vol. 16. Springer International Publishing, 2015. ISBN: 978-3-319-18104-2. URL: http://link.springer.com/10.1007/978-3-319-18105-9.

[6] E. Tlelo-Cuautle, J. d. J. Rangel-Magdaleno, and L. G. de la Fraga. *Engineering applications of FPGAs: Chaotic systems, artificial neural networks, random number generators, and secure communication systems*. Springer International Publishing, 2016. ISBN: 978-3-319-34115-6. DOI: 10.1007/978-3-319-34115-6.

[7] F. Capligins et al. “Frequency-Modulated Antipodal Chaos Shift Keying Chaotic Communication on Field Program Gate Array: Prototype Design and Performance Insights.” en. In: *Applied Sciences* 15.3 (Jan. 2025), p. 1156. ISSN: 2076-3417. DOI: 10.3390/app15031156.

[8] M. F. Taşdemir et al. “Performance Evaluation of FPGA-Based Design of Modified Chua Oscillator.” en. In: *Chaos Theory and Applications* 6.4 (Nov. 2024), pp. 249–256. ISSN: 2687-4539. DOI: 10.51537/chaos.1466919.

[9] D. Cirjulina et al. “Experimental Study on Colpitts Chaotic Oscillator-Based Communication System Application for the Internet of Things.” en. In: *Applied Sciences* 14.3 (Jan. 2024), p. 1180. ISSN: 2076-3417. DOI: 10.3390/app14031180.

[10] R. Babajans et al. “Synchronization of Analog-Discrete Chaotic Systems for Wireless Sensor Network Design.” en. In: *Applied Sciences* 14.2 (Jan. 2024), p. 915. ISSN: 2076-3417. DOI: 10.3390/app14020915.

[11] R. Babajans et al. “Performance Analysis of Vilnius Chaos Oscillator-Based Digital Data Transmission Systems for IoT.” en. In: *Electronics* 12.3 (Jan. 2023), p. 709. ISSN: 2079-9292. DOI: 10.3390/electronics12030709.

[12] F. Capligins et al. “FPGA-Based Antipodal Chaotic Shift Keying Communication System.” en. In: *Electronics* 11.12 (June 2022), p. 1870. ISSN: 2079-9292. DOI: 10.3390/electronics11121870.

[13] F. Capligins et al. “Experimental Study of the Chaotic Jerk Circuit Application for Chaos Shift Keying.” In: *Latvian Journal of Physics and Technical Sciences* 58.4 (2021), pp. 55–68. DOI: 10.2478/lpts-2021-0033.

[14] F. Capligins, A. Litvinenko, and D. Kolosovs. “Selective Fading Channel Performance Evaluation for Frequency-Modulated Antipodal Chaos Shift Keying Communication System.” In: *2025 IEEE 12th Workshop on Advances in Information, Electronic and Electrical Engineering (AIEEE)*. 2025, pp. 1–5. DOI: 10.1109/AIEEE66149.2025.11050770.

[15] D. Cirjulina et al. “Fundamental Frequency Impact on Vilnius Chaos Oscillator Dynamics.” In: *16th Chaotic Modeling and Simulation International Conference*. Ed. by C. H. Skiadas and Y. Dimotikalis. Cham: Springer Nature Switzerland, 2024, pp. 87–101. ISBN: 978-3-031-60907-7. DOI: 10.1007/978-3-031-60907-7_8.

[16] R. Babajans et al. “Study on Analog and Discrete Chaos Oscillators Synchronization.” In: *16th Chaotic Modeling and Simulation International Conference*. Ed. by C. H. Skiadas and Y. Dimotikalis. Cham: Springer Nature Switzerland, 2024, pp. 33–46. ISBN: 978-3-031-60907-7. DOI: 10.1007/978-3-031-60907-7_4.

[17] F. Capligins, A. Litvinenko, and D. Kolosovs. “Performance Evaluation of Frequency Modulated Antipodal Chaos Shift Keying Digital Communication System.” en. In: *2023 Workshop on Microwave Theory and Technology in Wireless Communications (MTTW)*. Riga, Latvia: IEEE, Oct. 2023, pp. 29–33. ISBN: 9798350393491. DOI: 10.1109/MTTW59774.2023.10320019.

[18] A. Aboltins et al. “Implementation of chaotic frequency modulation based spread spectrum communication system in software-defined radio.” In: *2023 IEEE Wireless Communications and Networking Conference (WCNC)*. 2023, pp. 1–6. DOI: 10.1109/WCNC55385.2023.10118612.

[19] F. Capligins, A. Litvinenko, and D. Kolosovs. “FPGA Implementation and Study of Antipodal Chaos Shift Keying Communication System.” In: *2021 IEEE Microwave Theory and Techniques in Wireless Communications (MTTW)* (2021), pp. 1–6. DOI: 10.1109/mttw53539.2021.9607226.

[20] F. Capligins et al. “FPGA Implementation and Study of Synchronization of Modified Chua’s Circuit-Based Chaotic Oscillator for High-Speed Secure Communications.” In: 2021. ISBN: 978-1-66542-538-4. DOI: 10.1109/AIEEE51419.2021.9435783.

[21] E. N. Lorenz. “Deterministic Nonperiodic Flow.” In: *The Theory of Chaotic Attractors*. Ed. by B. R. Hunt et al. New York, NY: Springer New York, 2004, pp. 25–36. ISBN: 978-0-387-21830-4. DOI: 10.1007/978-0-387-21830-4_2.

[22] L. O. Chua et al. “A Universal Circuit for Studying and Generating Chaos – Part II: Strange Attractors.” In: *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 40.10 (1993), pp. 745–761. ISSN: 10577122. DOI: 10.1109/81.246150.

[23] A. Sambas et al. “Design and analysis bidirectional chaotic synchronization of Rossler circuit and its application for secure communication.” In: *Applied Mathematical Sciences* 7.1-4 (2013), pp. 11–21. ISSN: 1312885X. DOI: 10.12988/ams.2013.13002.

[24] M. Kennedy. “Chaos in the Colpitts oscillator.” en. In: *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 41.11 (Nov. 1994), pp. 771–774. ISSN: 10577122. DOI: 10.1109/81.331536.

[25] G. Chen and X. Dong. “On feedback control of chaotic continuous-time systems.” In: *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 40.9 (1993), pp. 591–601. DOI: 10.1109/81.244908.

[26] S. Yu et al. “Design and implementation of n-scroll chaotic attractors from a general jerk circuit.” In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 52.7 (2005), pp. 1459–1476. DOI: 10.1109/TCSI.2005.851717.

[27] S. Vaidyanathan et al. “A new 4-D multi-stable hyperchaotic system with no balance point: Bifurcation analysis, circuit simulation, FPGA realization and image cryptosystem.” In: *IEEE Access* 9 (2021), pp. 144555–144573. ISSN: 21693536. DOI: 10.1109/ACCESS.2021.3121428.

[28] R. R. Babu and R. Karthikeyan. “Adaptive synchronization of novel chaotic system and its FPGA implementation.” In: *2015 International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials, ICSTM 2015 - Proceedings* May (2015), pp. 449–454. DOI: 10.1109/ICSTM.2015.7225459.

[29] Z. A. S. Rahman et al. “A new fractional-order chaotic system with its analysis, synchronization, and circuit realization for secure communication applications.” In: *Mathematics* 9.20 (2021). ISSN: 22277390. DOI: 10.3390/math9202593.

[30] F. Wang et al. “A Novel Multi-Shape Chaotic Attractor and Its FPGA Implementation.” In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 66.12 (2019), pp. 2062–2066. ISSN: 15583791. DOI: 10.1109/TCSII.2019.2907709.

[31] N. S. Soliman et al. “FPGA Implementation of X- and Heart-shapes Controllable Multi-Scroll Attractors.” In: *Proceedings - IEEE International Symposium on Circuits and Systems* 2018-May (May 2018), pp. 1–5. ISSN: 02714310. DOI: 10.1109/ISCAS.2018.8351760.

[32] E. E. Mahmoud, M. Higazy, and T. M. Al-Harthi. “A new nine-dimensional chaotic lorenz system with quaternion variables: Complicated dynamics, electronic circuit design, anti-anticipating synchronization, and chaotic masking communication application.” In: *Mathematics* 7.10 (2019). ISSN: 22277390. DOI: 10.3390/math7100877.

[33] T. Carroll and L. Pecora. “Synchronizing chaotic circuits.” In: *IEEE Transactions on Circuits and Systems* 38.4 (1991), pp. 453–456. DOI: 10.1109/31.75404.

[34] H. G. Chou et al. “A fuzzy-model-based chaotic synchronization and its implementation on a secure communication system.” In: *IEEE Transactions on Information Forensics and Security* 8.12 (2013), pp. 2177–2185. ISSN: 15566013. DOI: 10.1109/TIFS.2013.2286268.

[35] G. Kolumbán, M. P. Kennedy, and L. O. Chua. “The role of synchronization in digital communications using chaos - Part II: Chaotic modulation and chaotic synchronization.” In: *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 45.11 (1998), pp. 1129–1140. ISSN: 10577122. DOI: 10.1109/81.735435.

[36] J. C. Pizolato, M. A. Romero, and L. G. Neto. “Chaotic communication based on the particle-in-a-box electronic circuit.” In: *IEEE Transactions on Circuits and Systems I: Regular Papers* 55.4 (2008), pp. 1108–1115. ISSN: 10577122. DOI: 10.1109/TCSI.2008.916424.

[37] Z. A. S. Rahman et al. “Adaptive control synchronization of a novel memristive chaotic system for secure communication applications.” In: *Inventions* 4.2 (2019). ISSN: 24115134. DOI: 10.3390/inventions4020030.

[38] J. Zhang et al. “Adaptive coupled synchronization among three coupled chaos systems and its application to secure communications.” In: *Eurasip Journal on Wireless Communications and Networking* 2016.1 (2016). ISSN: 16871499. DOI: 10.1186/s13638-016-0630-4.

[39] A. A. Kekha Javan et al. “Design of adaptive-robust controller for multi-state synchronization of chaotic systems with unknown and time-varying delays and its application in secure communication.” In: *Sensors (Switzerland)* 21.1 (2021), pp. 1–21. ISSN: 14248220. DOI: 10.3390/s21010254.

[40] R. Kharel, K. Busawon, and Z. Ghassemlooy. “Modified chaotic shift keying using indirect coupled chaotic synchronization for secure digital communication.” In: *Chaos Theory*. July 2011, pp. 207–214. DOI: 10.1142/9789814350341_0024.

[41] P. Louodop et al. “Practical finite-time synchronization of jerk systems: Theory and experiment.” In: *Nonlinear Dynamics* 78.1 (2014), pp. 597–607. ISSN: 0924090X. DOI: 10.1007/s11071-014-1463-5.

[42] G. Kaddoum. “Wireless Chaos-Based Communication Systems: A Comprehensive Survey.” en. In: *IEEE Access* 4 (2016), pp. 2621–2648. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2016.2572730.

[43] E. Günay, K. Altun, and C. Ünal. “Implementation of CSK communicating system with switched SC-CNN based chaos generator.” In: *2017 10th International Conference on Electrical and Electronics Engineering (ELECO)*. 2017, pp. 1364–1367.

[44] G. Kaddoum and N. Tadayon. “Differential Chaos Shift Keying: A Robust Modulation Scheme for Power-Line Communications.” en. In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 64.1 (Jan. 2017), pp. 31–35. ISSN: 1549-7747, 1558-3791. DOI: 10.1109/TCSII.2016.2546901.

[45] G. Kolumban et al. “FM-DCSK: a novel method for chaotic communications.” In: *1998 IEEE International Symposium on Circuits and Systems (ISCAS)*. Vol. 4. May 1998, 477–480 vol.4. DOI: 10.1109/ISCAS.1998.698936.

[46] S. Wang and X. Wang. “M-DCSK-Based Chaotic Communications in MIMO Multipath Channels With No Channel State Information.” In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 57.12 (2010), pp. 1001–1005. DOI: 10.1109/TCSII.2010.2083091.

[47] H. Yang and G.-P. Jiang. “High-Efficiency Differential-Chaos-Shift-Keying Scheme for Chaos-Based Noncoherent Communication.” In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 59.5 (2012), pp. 312–316. DOI: 10.1109/TCSII.2012.2190859.

[48] G. Kaddoum and E. Soujeri. “NR-DCSK: A Noise Reduction Differential Chaos Shift Keying System.” In: *IEEE Transactions on Circuits and Systems II: Express Briefs* 63.7 (2016), pp. 648–652. DOI: 10.1109/TCSII.2016.2532041.

[49] M. Sushchik, L. Tsimring, and A. Volkovskii. “Performance analysis of correlation-based communication schemes utilizing chaos.” In: *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 47.12 (2000), pp. 1684–1691. DOI: 10.1109/81.899920.

[50] Z. Galias and G. Maggio. “Quadrature chaos-shift keying: theory and performance analysis.” In: *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 48.12 (Dec. 2001), pp. 1510–1519. ISSN: 1558-1268. DOI: 10.1109/TCSI.2001.972858.

[51] J. Li et al. “A secure communication scheme using chaotic system array.” In: *Chinese Control Conference, CCC* 2015-Septe.2 (2015), pp. 407–412. ISSN: 21612927. DOI: 10.1109/ChiCC.2015.7259672.

[52] L. Jie et al. “Chaos-shift-keying secure digital communications using feedback to synchronize Chua’s circuit: simulation and realization.” In: *Proceedings of International Conference on Communication Technology. ICCT ’96*. Vol. 1. 1996, 552–554 vol.1. DOI: 10.1109/ICCT.1996.545244.

[53] L. Chua et al. “A universal circuit for studying and generating chaos. I. Routes to chaos.” In: *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications* 40.10 (Oct. 1993), pp. 732–744. ISSN: 10577122. DOI: 10.1109/81.246149.

[54] O. Rössler. “An equation for continuous chaos.” In: *Physics Letters A* 57.5 (July 1976), pp. 397–398. ISSN: 0375-9601. DOI: 10.1016/0375-9601(76)90101-8.

[55] J. C. Sprott. “A new class of chaotic circuit.” In: *Physics Letters, Section A: General, Atomic and Solid State Physics* 266.1 (2000), pp. 19–23. ISSN: 03759601. DOI: 10.1016/S0375-9601(00)00026-8.

[56] A. Elwakil and M. Kennedy. “A family of Colpitts-like chaotic oscillators.” en. In: *Journal of the Franklin Institute* 336.4 (May 1999), pp. 687–700. ISSN: 00160032. DOI: 10.1016/S0016-0032(98)00046-5.

[57] M. S. Azzaz et al. “An FPGA implementation of a Feed-Back Chaotic Synchronization for secure communications.” In: *2010 7th International Symposium on Communication Systems, Networks and Digital Signal Processing, CSNDSP 2010* 1 (2010), pp. 239–243. DOI: 10.1109/csndsp16145.2010.5580426.

[58] A. Qi, C. Zhang, and H. Wang. “A switched hyperchaotic system and its FPGA circuitry implementation.” In: *Journal of Electronics* 28.3 (2011), pp. 383–388. ISSN: 02179822. DOI: 10.1007/s11767-011-0421-3.

[59] S. Sadoudi, M. S. Azzaz, and C. Tanougast. “Novel experimental synchronization technique for embedded chaotic communications.” In: *Proceedings - 2014 International Conference on Control, Decision and Information Technologies, CoDIT 2014* (2014), pp. 669–672. DOI: 10.1109/CoDIT.2014.6996976.

[60] E. Gunay, K. Altun, and C. Unal. “FPGA implementation of SC-CNN based chaos generator.” In: *2018 International Conference on Artificial Intelligence and Data Processing (IDAP)* (2018), pp. 1–7. DOI: 10.1109/siu.2017.7960281.

[61] L. Zhang. “System generator model-based FPGA design optimization and hardware co-simulation for Lorenz chaotic generator.” In: *2017 2nd Asia-Pacific Conference on Intelligent Robot Systems, ACIRS 2017* 2 (2017), pp. 170–174. DOI: 10.1109/ACIRS.2017.7986087.

[62] M. S. Azzaz et al. “FPGA Hardware Design of a Unified Chaotic System for CTRNG.” In: *2018 International Conference on Signal, Image, Vision and their Applications, SIVA 2018* (2019), pp. 1–4. DOI: 10.1109/SIVA.2018.8661042.

[63] A. J. El-Maksoud et al. “FPGA implementation of fractional-order Chua’s chaotic system.” In: *2018 7th International Conference on Modern Circuits and Systems Technologies, MOCAST 2018* (2018), pp. 1–4. DOI: 10.1109/MOCAST.2018.8376632.

[64] W. Guang-Yi, B. Xu-Lei, and W. Zhong-Lin. “Design and FPGA Implementation of a new hyperchaotic system.” In: *Chinese Physics B* 17.10 (2008), pp. 3596–3602. DOI: 10.1088/1674-1056/17/10/011.

[65] E. Dong et al. “Topological horseshoe analysis and FPGA implementation of a classical fractional order chaotic system.” In: *IEEE Access* 7 (2019), pp. 129095–129103. ISSN: 21693536. DOI: 10.1109/ACCESS.2019.2938556.

[66] H. Orabi et al. “On the Implementation of a Rotated Chaotic Lorenz System on FPGA.” In: *Proceedings - APCCAS 2019: 2019 IEEE Asia Pacific Conference on Circuits and Systems: Innovative CAS Towards Sustainable Energy and Technology Disruption* (2019), pp. 417–422. DOI: 10.1109/APCCAS47518.2019.8953183.

[67] F. Yu et al. “CCII and FPGA Realization: A Multistable Modified Fourth-Order Autonomous Chua’s Chaotic System with Coexisting Multiple Attractors.” In: *Complexity* 2020 (2020), pp. 1–17. ISSN: 10990526. DOI: 10.1155/2020/5212601.

[68] Y. Luo, S. Yu, and J. Liu. “Design and implementation of image chaotic communication via FPGA embedded ethernet transmission.” In: *2009 International Workshop on Chaos-Fractals Theories and Applications, IWCFTA 2009* (2009), pp. 148–152. DOI: 10.1109/IWCFTA.2009.38.

[69] E. Tlelo-Cuautle et al. “FPGA realization of a chaotic communication system applied to image processing.” In: *Nonlinear Dynamics* 82.4 (2015), pp. 1879–1892. ISSN: 1573269X. DOI: 10.1007/s11071-015-2284-x.

[70] H. Sira-Ramirez and C. Cruz-Hernandez. “Synchronization of chaotic systems: a generalized Hamiltonian systems approach.” In: *Proceedings of the American Control Conference* 2 (2000), pp. 769–773. ISSN: 07431619. DOI: 10.1109/acc.2000.876602.

[71] E. Gunay and K. Altun. “A performance comparison study of programmable platforms: FPAA and FPGA implementation of COOK communication system.” In: *2017 European Conference on Circuit Theory and Design, ECCTD 2017* (2017), pp. 1–4. DOI: 10.1109/ECCTD.2017.8093237.

[72] J. Schmitz and L. Zhang. “Rössler-based chaotic communication system implemented on FPGA.” In: *2017 IEEE 30th Canadian Conference on Electrical and Computer Engineering (CCECE)*. 2017, pp. 1–4. DOI: 10.1109/CCECE.2017.7946729.

[73] O. Guillén-Fernández et al. “On the synchronization techniques of chaotic oscillators and their FPGA-based implementation for secure image transmission.” In: *PLoS ONE* 14.2 (2019), pp. 1–34. ISSN: 19326203. DOI: 10.1371/journal.pone.0209618.

[74] E. A. Jackson and I. Grosu. “An open-plus-closed-loop (OPCL) control of complex dynamic systems.” In: *Physica D: Nonlinear Phenomena* 85.1-2 (1995), pp. 1–9. ISSN: 01672789. DOI: 10.1016/0167-2789(95)00171-Y.

[75] J.-C. Nuñez-Perez et al. “FPGA Realization of an Image Encryption System Using a 16-CPSK Modulation Technique.” en. In: *Electronics* 13.22 (Nov. 2024), p. 4337. ISSN: 2079-9292. DOI: 10.3390/electronics13224337.

[76] J.-C. Nuñez-Perez et al. “224-CPSK–CSS–WCDMA FPGA-Based Reconfigurable Chaotic Modulation for Multiuser Communications in the 2.45 GHz Band.” In: *Electronics* 14.20 (2025). ISSN: 2079-9292. DOI: 10.3390/electronics14203995.

[77] M.-A. Estudillo-Valdez, V.-A. Adeyemi, and J.-C. Nuñez-Perez. “FPGA realization of an image encryption system using the DCSK-CDMA technique.” en. In: *Integration* 96 (May 2024), p. 102157. ISSN: 01679260. DOI: 10.1016/j.vlsi.2024.102157.

[78] T. Bonny and W. Al Nassan. “Optimizing Security and Cost Efficiency in N-Level Cascaded Chaotic-Based Secure Communication System.” en. In: *Applied System Innovation* 7.6 (Oct. 2024), p. 107. ISSN: 2571-5577. DOI: 10.3390/asi7060107.

[79] N. F. Karagiorgos et al. “Unconventional Security for IoT: Hardware and Software Implementation of a Digital Chaotic Encrypted Communication Scheme.” en. In: *IEEE Internet of Things Journal* 11.11 (June 2024), pp. 19914–19925. ISSN: 2327-4662, 2372-2541. DOI: 10.1109/JIOT.2024.3371091.

[80] L. Xiao, L. Zhao, and J. Jin. “Preset-Time Convergence Fuzzy Zeroing Neural Network for Chaotic System Synchronization: FPGA Validation and Secure Communication Applications.” In: *Sensors* 25.17 (2025). ISSN: 1424-8220. DOI: 10.3390/s25175394.

[81] E. Tlelo-Cuautle et al. “Dynamics, FPGA realization and application of a chaotic system with an infinite number of equilibrium points.” In: *Nonlinear Dynamics* 89.2 (2017), pp. 1129–1139. ISSN: 1573269X. DOI: 10.1007/s11071-017-3505-2.

[82] U. Meyer-Baese. *Digital Signal Processing with Field Programmable Gate Arrays*. Springer-Verlag Berlin Heidelberg, Jan. 2014. ISBN: 978-3-642-45308-3. DOI: 10.1007/978-3-642-45309-0.

[83] M. H. Abd et al. “Synchronization of Monostatic Radar Using a Time-Delayed Chaos-Based FM Waveform.” In: *Remote Sensing* 14.9 (2022). ISSN: 2072-4292. DOI: 10.3390/rs14091984.

[84] T. Hoang. *Simulink model for observer based synchronization in Chua’s systems*. MATLAB Central File Exchange. URL: https://www.mathworks.com/matlabcentral/fileexchange/26246-observer-based-synchronization-in-chua-s-systems (visited on 02/12/2019).

[85] S. Bendoukha, S. Abdelmalek, and A. Ouannas. “Secure communication systems based on the synchronization of chaotic systems.” In: *Studies in Systems, Decision and Control* 200 (2019), pp. 281–311. ISSN: 21984190. DOI: 10.1007/978-3-030-12232-4_9.

[86] U. Mengali and A. N. D’Andrea. *Synchronization Techniques for Digital Receivers*. en. Boston, MA: Springer US, 1997. ISBN: 978-1-4899-1807-9. DOI: 10.1007/978-1-4899-1807-9.

[87] P. Zicari, P. Corsonello, and S. Perri. “A high flexible Early-Late Gate bit synchronizer in FPGA-based software defined radios.” en. In: *2008 4th European Conference on Circuits and Systems for Communications*. Bucharest, Romania: IEEE, July 2008, pp. 252–255. ISBN: 978-1-4244-2419-1. DOI: 10.1109/ECCSC.2008.4611687.

[88] J. G. Proakis and M. Salehi. *Digital communications*. en. 5. ed. Boston, Mass.: McGraw-Hill, 2008. ISBN: 978-0-07-295716-7.

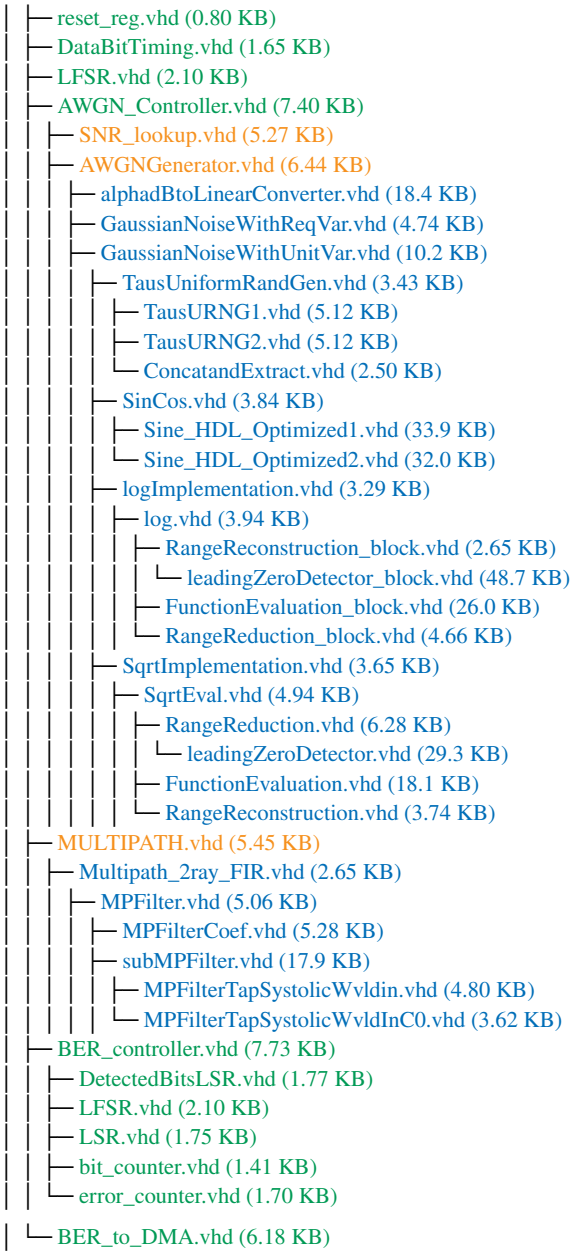
[89] *HDL Implementation of AWGN Generator*. URL: https://www.mathworks.com/help/wireless-hdl/ug/hdlawgngenerator.html (visited on 02/03/2024).

[90] *Avalon Streaming Interfaces*. URL: https://www.intel.com/content/www/us/en/docs/programmable/683091/22-3/streaming-interfaces.html (visited on 06/11/2024).

A VHDL FILE LIST FOR FM-ACSK FPGA PROTOTYPE

main.bdf (59.1 KB)
- Transmitter /
  - Chaos_Generator_Master.vhd (7.7 KB)
    - k_n.vhd (1.64 KB)
    - d_integrator.vhd (1.54 KB)
    - g_pwl.vhd (2.12 KB)
    - f_p1.vhd (1.18 KB)
    - f_p2.vhd (1.60 KB)
    - f_p3.vhd (1.69 KB)
    - f_p4.vhd (1.52 KB)
    - k_sum.vhd (1.02 KB)
    - r_output.vhd (1.19 KB)
  - invertor.vhd (1.01 KB)
  - invertion_controller.vhd (2.32 KB)
  - FM_MODULATOR_pac.vhd (0.75 KB)
  - FM_MODULATOR.vhd (3.18 KB)
    - FM_Modulator_block.vhd (4.39 KB)
      - NCO1.vhd (6.71 KB)
        - DitherGen.vhd (12.1 KB)
        - WaveformGen.vhd (5.81 KB)
          - LookUpTableGen.vhd (14.9 KB)
  - msb_replace.vhd (0.64 KB)
- Receiver /
  - msb_replace.vhd (0.64 KB)
  - FM_DEMODULATOR_pac.vhd (1.08 KB)
  - FM_DEMODULATOR.vhd (66.3 KB)
    - NCO1.vhd (6.71 KB)
      - DitherGen.vhd (12.1 KB)
      - WaveformGen.vhd (5.81 KB)
        - LookUpTableGen.vhd (14.9 KB)
    - Hilbert_filter31.vhd (2.42 KB)
      - Filter_block1.vhd (4.95 KB)
        - FilterCoef_block1.vhd (6.37 KB)
        - subFilter_block1.vhd (16.8 KB)
          - FilterTapSystolicPreAddWvlIn_block.vhd (6.7 KB)
          - FilterTapSystolicWvldInC0.vhd (3.48 KB)
    - LPF_FM_860kHz.vhd (4.43 KB)
      - LPF_FM_860kHz_RAISED.vhd (3.08 KB)
        - Filter_block.vhd (6.80 KB)
          - FilterCoef_block.vhd (122 KB)
          - subFilter_block.vhd (270 KB)
            - FilterTapWvldInC0.vhd (2.27 KB)
            - FilterTapWMultOut.vhd (4.10 KB)
            - FilterTapWvldIn.vhd (3.87 KB)
            - FilterTapPostAdd.vhd (2.51 KB)
      - LPF_FM_860kHz_FIX.vhd (3.04 KB)
        - Filter.vhd (6.70 KB)
          - FilterCoef.vhd (5.90 KB)
          - subFilter.vhd (14.0 KB)

```
│ │ │ │ │ │ └── FilterTapSystolicPreAddWvlIn.vhd (6.68 KB)
│ │ ├── Quadrant_Mapper.vhd (4.89 KB)
│ │ ├── CordicKernelMag.vhd (4.31 KB)
│ │ ├── Quadrant_Correction.vhd (3.38 KB)
│ │ └── DiffUnwrap1.vhd (5.02 KB)
│ ├── LOWPASS_FILTER_pac.vhd (0.61 KB)
│ ├── LOWPASS_FILTER.vhd (4.14 KB)
│ │ ├── LPF_ACSK_215kHz_RAISED.vhd (2.46 KB)
│ │ │ ├── Filter_block.vhd (4.98 KB)
│ │ │ │ ├── FilterCoef_block.vhd (129 KB)
│ │ │ │ ├── subFilter_block.vhd (287 KB)
│ │ │ │ │ ├── FilterTapWvldInC0.vhd (2.27 KB)
│ │ │ │ │ ├── FilterTapWMultOut.vhd (4.03 KB)
│ │ │ │ │ ├── FilterTapPostAdd.vhd (2.51 KB)
│ │ │ │ │ ├── FilterTapPostSub.vhd (2.50 KB)
│ │ │ │ │ └── FilterTapWvldIn.vhd (3.75 KB)
│ │ ├── LLPF_ACSK_215kHz_FIX.vhd (2.41 KB)
│ │ │ ├── Filter.vhd (4.89 KB)
│ │ │ │ ├── FilterCoef.vhd (4.17 KB)
│ │ │ │ ├── subFilter.vhd (10.7 KB)
│ │ │ │ │ └── FilterTapSystolicPreAddWvlIn.vhd (6.68 KB)
│ ├── invertor.vhd (1.01 KB)
│ ├── single_delay.vhd (1.00 KB)
│ ├── Chaos_Generator_Slave.vhd (7.89 KB)
│ │ ├── k_n.vhd (1.64 KB)
│ │ ├── g_module.vhd (0.82 KB)
│ │ ├── d_integrator.vhd (1.54 KB)
│ │ ├── g_feedback.vhd (0.64 KB)
│ │ ├── f_p1.vhd (1.18 KB)
│ │ ├── f_p2.vhd (1.60 KB)
│ │ ├── f_p3.vhd (1.69 KB)
│ │ ├── f_p4.vhd (1.52 KB)
│ │ ├── k_sum.vhd (1.02 KB)
│ │ └── r_output.vhd (1.19 KB)
│ ├── delta_abs.vhd (0.98 KB)
│ ├── Sliding_Average.vhd (3.17 KB)
│ │ └── fifo_17_8192.vhd (7.41 KB)
│ ├── delta.vhd (1.01 KB)
│ ├── Symbol_Phase_Sync.vhd (19.6 KB)
│ │ ├── FreeRunningCnt_12bit.vhd (2.86 KB)
│ │ │ ├── Increment_Real_World.vhd (1.44 KB)
│ │ │ └── Wrap_To_Zero.vhd (1.02 KB)
│ │ ├── EARLY_GATE_REG_SIGN.vhd (2.46 KB)
│ │ ├── GATE_REG_SIGN.vhd (2.43 KB)
│ │ ├── NEXT_PHASE_REG.vhd (2.41 KB)
│ │ └── PHASE_CLOCK_REG.vhd (2.17 KB)
│ ├── Detector_pac.vhd (0.5 KB)
│ ├── Detector.vhd (7.51 KB)
│ │ ├── Symbol_Int_REG.vhd (2.42 KB)
│ │ └── Bit_output_REG.vhd (2.16 KB)
├── Other modules /
```


reset_reg.vhd (0.80 KB)
DataBitTiming.vhd (1.65 KB)
LFSR.vhd (2.10 KB)
AWGN_Controller.vhd (7.40 KB)
SNR_lookup.vhd (5.27 KB)
AWGNGenerator.vhd (6.44 KB)
alphadBtoLinearConverter.vhd (18.4 KB)
GaussianNoiseWithReqVar.vhd (4.74 KB)
GaussianNoiseWithUnitVar.vhd (10.2 KB)
TausUniformRandGen.vhd (3.43 KB)
TausURNG1.vhd (5.12 KB)
TausURNG2.vhd (5.12 KB)
ConcatandExtract.vhd (2.50 KB)
SinCos.vhd (3.84 KB)
Sine_HDL_Optimized1.vhd (33.9 KB)
Sine_HDL_Optimized2.vhd (32.0 KB)
logImplementation.vhd (3.29 KB)
log.vhd (3.94 KB)
RangeReconstruction_block.vhd (2.65 KB)
leadingZeroDetector_block.vhd (48.7 KB)
FunctionEvaluation_block.vhd (26.0 KB)
RangeReduction_block.vhd (4.66 KB)
SqrtImplementation.vhd (3.65 KB)
SqrtEval.vhd (4.94 KB)
RangeReduction.vhd (6.28 KB)
leadingZeroDetector.vhd (29.3 KB)
FunctionEvaluation.vhd (18.1 KB)
RangeReconstruction.vhd (3.74 KB)
MULTIPATH.vhd (5.45 KB)
Multipath_2ray_FIR.vhd (2.65 KB)
MPFilter.vhd (5.06 KB)
MPFilterCoef.vhd (5.28 KB)
subMPFilter.vhd (17.9 KB)
MPFilterTapSystolicWvldin.vhd (4.80 KB)
MPFilterTapSystolicWvldInC0.vhd (3.62 KB)
BER_controller.vhd (7.73 KB)
DetectedBitsLSR.vhd (1.77 KB)
LFSR.vhd (2.10 KB)
LSR.vhd (1.75 KB)
bit_counter.vhd (1.41 KB)
error_counter.vhd (1.70 KB)
BER_to_DMA.vhd (6.18 KB)

B BER READING C PROGRAM FOR FPGA PROTOTYPES

```c
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <unistd.h>
#include <math.h> // for pow()
#include <string.h> // for strcmp()
#include "cma_api.h"
#include "msgdma_api.h"

#define _I(fmt, args...) printf(fmt"\n", ##args)
#define _E(fmt, args...) printf("ERROR: "fmt"\n", ##args)

#define ARRAY_LENGTH 2*(UINT8_MAX + 1)

#define DATA_ENTRIES 2 // Number of data entries are received from FPGA
#define DATA_PACKETS DATA_ENTRIES*4 // Two 32-bit entries each sent by 8-bit packet
#define DMA_BUFFER 64 // Buffer storing outdated data that must be skipped
#define SKIP_PACKETS ARRAY_LENGTH-DATA_PACKETS // Skip all buffered entries except the most recent
    complete entry

/* Global MSGDMA descriptor */
msgdma_device_t msgdma_RX; // DMA used for reading

// Display the selected memory buffer
void display_buffer(uint32_t *buffer, int length, int skip) {
  for (int i = skip; i < length; i++) {
    printf("%.3u ", buffer[i]);

    // new line after each 8th element
    if (i % 8 == 7) {
      putchar('\n');
    }
    else if (i % 4 == 3) {
      printf(" | ");
    }
  }
}

void fpga_read(void *dst, unsigned size) {
  struct msgdma_dscr dscr_RX; // descriptor RX

  dscr_RX.read_addr = 0;
  dscr_RX.write_addr = cma_get_phy_addr(dst);
  dscr_RX.length  = size;
  dscr_RX.control = MSGDMA_DSCR_GO | MSGDMA_DSCR_TRANSFER_COMPLETE_IRQ_MASK;

  write_standard_descriptor(msgdma_RX, &dscr_RX);
}

void info(void) {
  _I("");
  _I("--------INFORMATION ABOUT THE FM-ACSK SYSTEM--------");
  _I("Operation mode: 16-bit LFSR pseudo random bit transmit for BER counting");
  _I("\tNote 1: 32-bit key is used for locking into bit track&compare state (BER counter sync)");
  _I("\tNote 2: usually locking happens at the start on new LFSR cycle (10,7 seconds)");
  _I("\tNote 3: if SNR is too low, locking might take N (1,2,3... Inf) LFSR cycles (10,7*N seconds)");
  _I("FM modulation index: 3");
  _I("FM carrier frequency: 10,7 MHz");
  _I("FM bandwidth: 1720 kHz");
  _I("FPGA clock speed: 50 MHz");
  _I("FPGA clock cycles per 1 data bit: 8192");
  _I("Transmissions speed: 6,103515625 kbps");
  _I("Time to transmit 10^5 bits (for BER 10^-4 and above): 16,384 seconds");
  _I("Time to transmit 10^6 bits (for BER 10^-5 and above): 2 minutes 43,84 seconds");
  _I("Time to transmit 10^7 bits (for BER 10^-6 and above): 27 minutes 18,4 seconds");
  _I("Time to transmit 10^8 bits (for BER 10^-7 and above): 4 hours 33 minutes 4 seconds");
  _I("Time to transmit 10^9 bits (for BER 10^-8 and above): 45 hours 30 minutes 40 seconds");
  _I("Bit counter is 32-bit register (8 days to fill)");
  _I("FPGA control: \n\tKEY3-->whole FM-ACSK reset \n\tKEY2-->BER Counter reset & re-lock");
  _I("\tSW3...SW0--> reserved for SNR level control");
  _I("FPGA indicators: \n\tLED3-->BER Counter LOCKED; \n\tLED1-->bits sent; \n\tLED0-->bits received");
  _I("--------END OF INFO SECTION--------");
  _I("");
}

int main(int argc, char **argv) {
```

```
   _I("BER data reading from FPGA for FM-ACSK communication system.");
   _I("Author: Filips Capligins");
   _I("Compiled at %s %s", __DATE__, __TIME__);

   _I("");
   _I("Starting Contiguous Memory Allocator module and MSGDMA driver... (ignore the errors on repeated inits)");
   system("insmod modules/cma.ko && insmod modules/msgdma.ko");

// _I("Initializing CMA APIs...");
   if (cma_init() != 0) {
      _E("Failed to initialize CMA!");
      return 1;
   }

// _I("Initializing MSGDMA devices...");
   msgdma_RX = msgdma_init("/dev/msgdma1");
   if (msgdma_RX < 0) {
      _E("Failed to initialize msgdma_RX device!");
      return 1;
   }

   if (enable_global_interrupt_mask(msgdma_RX) != 0) {
      _E("Failed to enable interrupts!");
      return 1;
   }
// Checking if debug mode is evoked...
   if ( argc == 2 && (strcmp(argv[1], "-d") == 0) ) _I("Running program in DEBUG mode.");
// _I("Allocating contiguous memory...");
   uint32_t *buffer_dst = cma_alloc_noncached(ARRAY_LENGTH * sizeof(uint32_t));
// Variables used for storing and processing received data
   uint32_t bit_cnt, err_cnt;
   double ber;
   char key = '0';

   info();

   for (;;) { // start of the infinite loop
      if (key != 'i') {
         _I("Reading FM-ACSK BER data from FPGA...");
         fpga_read(buffer_dst, ARRAY_LENGTH * sizeof(uint32_t));
         if ( argc == 2 && (strcmp(argv[1], "-d") == 0) ) {
            _I("Received FULL RAW data (32-bit bit_cnt and err_cnt split by 8 bits):"); // DEBUG
            display_buffer(buffer_dst, ARRAY_LENGTH, 0); // DEBUG
         }

         bit_cnt = buffer_dst[SKIP_PACKETS] * pow(2, 24) + buffer_dst[SKIP_PACKETS + 1] * pow(2, 16) + buffer_dst[SKIP_PACKETS + 2] * pow(2, 8) + buffer_dst[SKIP_PACKETS + 3];
         err_cnt = buffer_dst[SKIP_PACKETS + 4] * pow(2, 24) + buffer_dst[SKIP_PACKETS + 5] * pow(2, 16) + buffer_dst[SKIP_PACKETS + 6] * pow(2, 8) + buffer_dst[SKIP_PACKETS + 7];
         ber = (float)err_cnt / (float)bit_cnt;

         _I("");
         _I("\t\                          t");
         _I("\t\ t          S  U  M  M  A  R  Y            ");
         _I("\t\                          t");
         _I("\t\ tReceived bits:  %d                       ", bit_cnt);
         _I("\t\                          t");
         _I("\t\ tErrors (bits):  %d                       ", err_cnt);
         _I("\t\                          t");
         _I("\t\ tBER:           %.16e                     ", ber);
         _I("\t\                          t");

         _I("Enter:\n\t- 'q' to stop program; \n\t- 'i' to print info section. \nHit Enter to read BER data from FPGA...");
         printf("=>");
         key = getchar(); // Get the character that was pressed
         if (key == 'q')
         {
            _I("Exiting...");
            break;
         }
         else if (key == 'i') {
            info();
         }
         // }
      }
      else key = getchar(); // process the pending Enter key (return) pressed (stacked) after 'i'
   } // end of inf loop

```

```
// Exit routine:
// _I("Freeing memory...");
   cma_free(buffer_dst);

   // _I("De-initializing MSGDMA devices...");
   if (msgdma_release(msgdma_RX) != 0) {
      _E("Failed to de-initialize msgdma_RX!");
      return 1;
   }

   // _I("De-initializing CMA API...");
   if (cma_release() != 0) {
      _E("Failed to de-initialize CMA!");
      return 1;
   }
   _I("Program executed successfully!");
   return 0;
}
```

C VHDL CODES FOR SOME PROTOTYPE MODULES

C.1 AWGN Controller VHDL (for FM-ACSK)

```
-- Module name: AWGN_Controller
-- Description: White noise source for FM-ACSK received signal, with controllable SNR using switches and keys
-- Usage: Module accepts 128 different SNR states, defined by SW3, SW2, SW1, SW0, LED2, LED1, LED0 or SW_STATE&LED_STATE
-- SW_STATE is set directly by switches, but LED_STATE is set using decrement/increment with KEY1/KEY0
-- IF SW_STATE & LED_STATE = 0, the input is passed directly to the output without adding noise, and AWGN generator is disabed
-- SW_STATE & LED_STATE values 1..127 are mapped to appropriate sfix14_En12 values representing SNR values -4.4871:0.2:20.7129 dB
-- These SNR values are approximate real values, taking into account both SNR adjustment factors:
--    1 - AWGN generator attentuation by 12.1112642 dB to match the FM-ACSK input power at 0 dB SNR
--    2 - FM (filtered) bandwitdh-to-Nyquist-frequency ratio of 1.72/25 = 0.0688 --> -11.6241156 dB
-- Considering both SNR adjustment factors together, the real snr values are reduced by 0.4871486 dB
-- before being passed to the AWGN generator, since its lookup table contains mapped values with 0.1 dB
-- SNR precision and is configured to cover the range of -4:0.2:21.2 dB.
-- ------------------------------------------------------------------------------
-- Inputs:
-- clk     - global clock
-- rst     - global asynchronous reset
-- clk_en  - global clock enable
-- sw3..0  - SocKit DevKit onboard switches to set the SW_STATE
-- key1..0 - SocKit DevKit onboard keys used to decrement/increment LED_STATE by one for each button press
-- RX_in   - 14-bit input from ADC after MSB_Replace, expected to be transmitted FM_ACSK signal
-- ------------------------------------------------------------------------------
-- Outputs:
-- led2..0 - SocKit DevKit onboard LEDs for LED_STATE register representation
-- RX_awgn - 14-bit output equal to RX_in + AWGN when noise is enabled, or RX_in when noise is disabled

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.sll;

entity AWGN_Controller is
  port(
    clk          : in  std_logic;
    rst          : in  std_logic;
    clk_en       : in  std_logic;
    sw3          : in  std_logic;
    sw2          : in  std_logic;
    sw1          : in  std_logic;
    sw0          : in  std_logic;
    key1         : in  std_logic;
    key0         : in  std_logic;
    RX_in        : in  std_logic_vector(13 downto 0); -- sfix14_En12
    led2         : out std_logic;
    led1         : out std_logic;
    led0         : out std_logic;
    RX_awgn      : out std_logic_vector(13 downto 0) -- sfix14_En12
```

```
  );
end AWGN_Controller;

architecture rtl of AWGN_Controller is
-- Signals:
signal key1_reg0       : std_logic := '0';
signal key1_reg1       : std_logic := '0';
signal key1_f_edge     : std_logic := '0';
signal key0_reg0       : std_logic := '0';
signal key0_reg1       : std_logic := '0';
signal key0_f_edge     : std_logic := '0';
signal KEY_STATE       : std_logic_vector(1 downto 0) := (others => '0');
signal LED_STATE       : unsigned(2 downto 0) := (others => '0');
signal LED_STATE_next  : unsigned(2 downto 0) := (others => '0');
signal SW_STATE        : std_logic_vector(3 downto 0) := (others => '0');
signal SW_STATE_next   : std_logic_vector(3 downto 0) := (others => '0');
signal SNR_STATE       : std_logic_vector(6 downto 0) := (others => '0');
signal SNR_dB_next     : std_logic_vector(15 downto 0) := (others => '0'); -- sfix16_En9
signal SNR_dB          : std_logic_vector(15 downto 0) := (others => '0'); -- sfix16_En9
signal awgn            : std_logic_vector(13 downto 0) := (others => '0'); -- sfix14_En12
signal valid_awgn      : std_logic := '0';
signal noise_en        : std_logic := '0';
signal RX_awgn_next    : std_logic_vector(13 downto 0) := (others => '0'); -- sfix14_En12
signal RX_awgn_s       : std_logic_vector(13 downto 0) := (others => '0'); -- sfix14_En12

-- Component Declaration:
component SNR_lookup
  port(
    snr_index_in  : in  std_logic_vector(6 downto 0); -- ufix7
    snr_dB_out    : out std_logic_vector(15 downto 0) -- sfix16_En9
  );
end component;

component AWGNGenerator
  port(
    clk           : in  std_logic;
    reset         : in  std_logic;
    clk_enable    : in  std_logic;
    snrdB         : in  std_logic_vector(15 downto 0); -- sfix16_En9
    awgn          : out std_logic_vector(13 downto 0); -- sfix14_En12
    valid         : out std_logic
  );
end component;

begin
-- Component initialization:

SNR_lookup1 : SNR_lookup
  port map(
    snr_index_in  => SNR_STATE,
    snr_dB_out    => SNR_dB_next
  );

AWGNGenerator1 : AWGNGenerator
  port map(
    clk           => clk,
    reset         => rst,
    clk_enable    => (clk_en and noise_en),
    snrdB         => SNR_dB,
    awgn          => awgn,
    valid         => valid_awgn
  );

-- Synchronous part:

-- KEY 1 falling edge detector regs:
k1_f_edge_detector : process(clk,rst)
begin
  if(rst='1') then
    key1_reg0        <= '0';
    key1_reg1        <= '0';
  elsif(rising_edge(clk)) then
    key1_reg0        <= key1;
    key1_reg1        <= key1_reg0;
  end if;
end process k1_f_edge_detector;

-- KEY 0 falling edge detector regs:
k0_f_edge_detector : process(clk,rst)
begin
```

```
  if(rst='1') then
    key0_reg0        <= '0';
    key0_reg1        <= '0';
  elsif(rising_edge(clk)) then
    key0_reg0        <= key0;
    key0_reg1        <= key0_reg0;
  end if;
end process k0_f_edge_detector;

-- SW_STATE reg:
sw_state_reg : process(clk, rst)
begin
  if rst = '1' then
    SW_STATE <= (others => '0');
  elsif rising_edge(clk) then
    SW_STATE <= SW_STATE_next;
  end if;
end process sw_state_reg;

-- LED_STATE reg:
led_state_reg : process(clk, rst)
begin
  if rst = '1' then
    LED_STATE <= (others => '0');
  elsif rising_edge(clk) then
    LED_STATE <= LED_STATE_next;
  end if;
end process led_state_reg;

-- SNR_dB reg:
process(clk, rst)
begin
   if rst = '1' then
      SNR_dB <= (others => '0');
   elsif rising_edge(clk) then
      if (clk_en and noise_en) = '1' then
        SNR_dB <= SNR_dB_next;
      else
        SNR_dB <= SNR_dB;
      end if;
   end if;
end process;

-- RX in + AWGN output reg:
process(clk, rst)
begin
   if rst = '1' then
      RX_awgn_s <= (others => '0');
   elsif rising_edge(clk) then
      if clk_en = '1' then
        RX_awgn_s <= RX_awgn_next;
      else
        RX_awgn_s <= RX_awgn_s;
      end if;
   end if;
end process;

-- Asynchronous part:
key1_f_edge         <= not key1_reg0 and key1_reg1;
key0_f_edge         <= not key0_reg0 and key0_reg1;

KEY_STATE           <= key1_f_edge & key0_f_edge;

SW_STATE_next       <= sw3 & sw2 & sw1 & sw0;

led2                <= LED_STATE(2);
led1                <= LED_STATE(1);
led0                <= LED_STATE(0);

-- LED_STATE_next = LED_STATE(KEY_STATE) switch:
with KEY_STATE select LED_STATE_next <=
  LED_STATE     when "00",
  LED_STATE + 1 when "01",
  LED_STATE - 1 when "10",
  LED_STATE     when others;

SNR_STATE           <= SW_STATE & std_logic_vector(LED_STATE);

with SNR_dB_next select noise_en <=
  '0'    when x"7FFF",
```

```
  '1'    when others;

with (valid_awgn nand noise_en) select RX_awgn_next <=
  std_logic_vector( signed(RX_in) + signed(awgn) ) when '0',
  RX_in                                            when others;

RX_awgn             <= RX_awgn_s;

end rtl;
```

C.2 BER Controller VHDL

```
-- Module name: BER_controller
-- Description: BER measurement module to be placed between the AM-ACSK/FM-ACSK receiver output and the
    memory interface used by the integrated ARM application.
-- Requires an identical LFSR to the one used as the bit source for the AM-ACSK/FM-ACSK transmitter input.
-- ------------------------------------------------------------------------------
-- Inputs:
-- clk     - global clock
-- rst     - global asynchronous reset
-- ber_en  - "bit_is_valid" output flag impulse from AM-ACSK/FM-ACSK RX detector AND global clock enable
-- bits_in - output bits from AM-ACSK/FM-ACSK RX detector
-- ------------------------------------------------------------------------------
-- Outputs:
-- err_cnt   - 31-bit error count (from the moment KEY matches and state is LOCKED)
-- bit_cnt   - 32-bit received bit count (from the moment KEY matches and state is LOCKED)
-- ber_ready - for BER reading synchronization, is equal to processing_step by one clock cycle
-- locked    - LOCKED-state flag indended for a LED indication
-- ------------------------------------------------------------------------------
-- State machine description:
-- S1 (Initial) - LFSR, LSR and counters are frozen, DetectedBitsLSR is moving; LOCKED == '0'
--             Transition to S2 condition: DetectedBitsLSR == LFSR&LSR
-- S2 (LOCKED)  - LFSR, LSR, DetectedBitsLSR, and counters are advancing; LOCKED == '1'
--             Transition to S1 condition: rst == '1'

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity BER_controller is
  generic(
    DW_LFSR       : natural := 16;
    DW_LSR        : natural := 16;
    DW_DetBitsLSR : natural := 32;
    DW_BitCNT     : natural := 32;
    DW_ErrCNT     : natural := 31
  );
  port(
    clk         : in  std_logic;
    rst         : in  std_logic;
    ber_en      : in  std_logic;
    bits_in     : in  std_logic;
    err_cnt     : out std_logic_vector (DW_ErrCNT - 1 downto 0);
    bit_cnt     : out std_logic_vector (DW_BitCNT - 1 downto 0);
    ber_ready   : out std_logic;
    locked      : out std_logic
  );
end BER_controller;

architecture BER_controller_beh of BER_controller is
-- constants
-- signal declaration
signal KEY             : std_logic_vector (DW_DetBitsLSR - 1 downto 0);
signal processing_step : std_logic;
signal dblsr_state     : std_logic_vector (DW_DetBitsLSR - 1 downto 0);
signal dblsr_out       : std_logic;
signal lfsr_state      : std_logic_vector (DW_LFSR - 1 downto 0);
signal lfsr_out        : std_logic;
signal lsr_state       : std_logic_vector (DW_LSR - 1 downto 0);
signal lsr_out         : std_logic;
signal lock_event      : std_logic := '0';
signal is_locked       : std_logic := '0';
signal bit_det         : std_logic := '0';
signal bit_lfsr        : std_logic := '0';
signal ber_ready_s     : std_logic := '0';
```

```
63
64  type States is (InitialState, LockedState);
65     signal State : States := InitialState;
66
67  -- components
68
69  component LFSR
70    generic(
71      DATA_WIDTH      : natural
72    );
73    port(
74      clk             : in  std_logic;
75      rst             : in  std_logic;
76      lfsr_en         : in  std_logic;
77      lfsr_state_out  : out std_logic_vector (DW_LFSR - 1 downto 0);
78      lfsr_out        : out std_logic
79    );
80  end component;
81
82  component LSR
83    generic(
84      DATA_WIDTH     : natural
85    );
86    port(
87      clk            : in  std_logic;
88      rst            : in  std_logic;
89      lsr_en         : in  std_logic;
90      lsr_in         : in  std_logic;
91      lsr_state_out  : out std_logic_vector (DW_LSR - 1 downto 0);
92      lsr_out        : out std_logic
93    );
94  end component;
95
96  component DetectedBitsLSR
97    generic(
98      DATA_WIDTH      : natural
99    );
100   port(
101     clk             : in  std_logic;
102     rst             : in  std_logic;
103     dblsr_en        : in  std_logic;
104     dblsr_in        : in  std_logic;
105     dblsr_state_out : out std_logic_vector (DW_DetBitsLSR - 1 downto 0);
106     dblsr_out       : out std_logic
107   );
108 end component;
109
110 component bit_counter
111   generic(
112     CNT_BITS    : natural
113   );
114   port(
115     clk         : in  std_logic;
116     rst         : in  std_logic;
117     bcnt_en     : in  std_logic;
118     bit_cnt_out : out std_logic_vector (DW_BitCNT - 1 downto 0)
119   );
120 end component;
121
122 component error_counter
123   generic(
124     CNT_BITS    : natural
125   );
126   port(
127     clk         : in  std_logic;
128     rst         : in  std_logic;
129     ecnt_en     : in  std_logic;
130     bit_det     : in  std_logic;
131     bit_lfsr    : in  std_logic;
132     err_cnt_out : out std_logic_vector (DW_ErrCNT - 1 downto 0)
133   );
134 end component;
135
136 begin
137 -- Component initialization:
138
139 LFSR1 : LFSR
140  generic map(
141    DATA_WIDTH      => DW_LFSR
142  )
```

```
 port map(
   clk             => clk,
   rst             => rst,
   lfsr_en         => processing_step,
   lfsr_state_out => lfsr_state,
   lfsr_out        => lfsr_out
 );

LSR1 : LSR
  generic map(
    DATA_WIDTH     => DW_LSR
  )
  port map(
    clk             => clk,
    rst             => rst,
    lsr_en          => processing_step,
    lsr_in          => lfsr_out,
    lsr_state_out => lsr_state,
    lsr_out         => lsr_out
  );

DetectedBitsLSR1 : DetectedBitsLSR
  generic map(
    DATA_WIDTH      => DW_DetBitsLSR
  )
  port map(
    clk             => clk,
    rst             => rst,
    dblsr_en        => ber_en,
    dblsr_in        => bits_in,
    dblsr_state_out => dblsr_state,
    dblsr_out       => dblsr_out
  );

bit_counter1 : bit_counter
  generic map(
    CNT_BITS    => DW_BitCNT
  )
  port map(
    clk         => clk,
    rst         => rst,
    bcnt_en     => processing_step,
    bit_cnt_out => bit_cnt
  );

error_counter1 : error_counter
  generic map(
    CNT_BITS    => DW_ErrCNT
  )
  Pport map(
    clk         => clk,
    rst         => rst,
    ecnt_en     => processing_step,
    bit_det     => bit_det,
    bit_lfsr    => bit_lfsr,
    err_cnt_out => err_cnt
  );

-- synchronous part:
-- Finite State Machine:
process(clk, rst) is
begin
   if rising_edge(clk) then
      if rst = '1' then
         State <= InitialState;
         is_locked <= '0';
      else
         case State is
            when InitialState =>
               is_locked <= '0';

               if lock_event then
                  State <= LockedState;
               end if;
            when LockedState =>
               is_locked <= '1';
         end case;
      end if;
   end if;
end process;
```

```
223
224 -- ber_ready = processing_step 1 delay
225    process(clk, rst)
226    begin
227       if rst = '1' then
228          ber_ready_s <= '0';
229       elsif rising_edge(clk) then -- no clk_en dependency!
230          ber_ready_s <= processing_step;
231       end if;
232    end process;
233
234 -- asynchronous part:
235
236 KEY             <= lfsr_state & lsr_state;
237 lock_event      <= '1' when dblsr_state = KEY else '0';
238
239 processing_step <= (ber_en and is_locked);
240 locked          <= is_locked;
241 bit_det         <= dblsr_out;
242 bit_lfsr        <= lsr_out;
243
244 ber_ready       <= ber_ready_s; -- MAIN
245
246 end BER_controller_beh;
```

Filips Čapligins was born in 1993 in Riga, Latvia. He received an Academic Bachelor's degree in Electrical Engineering (2019) and a Professional Master's degree in Electronics and the qualification of Lead Electronics Engineer (2022) from Riga Technical University (RTU). Since 2020, he has been employed at RTU, initially as a research assistant and, since 2022, as a researcher. Since 2026, he has been a radio electronics engineer at JSC "SAF Tehnika". His research interests include the application of chaos theory in telecommunications, digital signal processing, and FPGA technologies.