DESIGN AND IMPLEMENTATION OF AN EDUCATIONAL FM TRANSMITTER WITH FPGA USING SDR TECHNIQUES

2022
MASTER THESIS
ELECTRICAL AND ELECTRONICS ENGINEERING

Ahmed ALGHHRIUNI

Thesis Advisor
Assist. Prof. Dr. Bilgehan ERKAL

DESIGN AND IMPLEMENTATION OF AN EDUCATIONAL FM TRANSMITTER WITH FPGA USING SDR TECHNIQUES

Ahmed ALGHHRIUNI

Thesis Advisor

Assist. Prof. Dr. Bilgehan ERKAL

T.C.

Karabuk University

Institute of Graduate Programs

Department of Electrical and Electronics Engineering

Prepared as

Master Thesis

KARABUK

December 2022

I certify that in my opinion the thesis submitted by Ahmed ALGHHRIUNI titled "DESIGN AND IMPLEMENTATION OF AN EDUCATIONAL FM TRANSMITTER WITH FPGA USING SDR TECHNIQUES" is fully adequate in scope and in quality as a thesis for the degree of Master of Science.

Assist. Prof. Dr. Bilgehan ERKAL ....
Thesis Advisor, Department of Electrical and Electronics Engineering

This thesis is accepted by the examining committee with a unanimous vote in the Department of Electrical and Electronics Engineering as a Master of Science thesis. 19,12,2022

Examining Committee Members (Institutions)		Signature
Chairman	: Assoc. Prof. Dr. Salih GÖRGÜNOGLU (KU)	....
Member	: Assoc. Prof. Dr. Hüseyin DEMIREL (KBU)	....
Member	: Assist. Prof. Dr. Bilgehan ERKAL (KBU)	....

The degree of Master of Science by the thesis submitted is approved by the Administrative Board of the Institute of Graduate Programs, Karabuk University.

Assoc. Prof. Dr. Müslüm KUZU ....
Director of the Institute of Graduate Programs

“I declare that all the information within this thesis has been gathered and presented in accordance with academic regulations and ethical principles and I have according to the requirements of these regulations and principles cited all those which do not originate in this work as well.”

Ahmed Ibrahim M Alghhriuni

ABSTRACT

M. Sc. Thesis

DESIGN AND IMPLEMENTATION OF AN EDUCATIONAL FM TRANSMITTER WITH FPGA USING SDR TECHNIQUES

Ahmed Ibrahim M AlGHHRIUNI

Karabük University
Institute of Graduate Programs
The Department of Electrical and Electronics Engineering

Thesis Advisor:
Assist. Prof. Dr. Bilgehan ERKAL
December 2022, 75 pages

The basic purpose of Software Defined Radio (SDR) systems is to use a digital signal processor to numerically handle radio signals. The use of a processor like a Field Programmable Gate Array (FPGA) to perform tasks like modulation, demodulation, signal creation, and line coding on these systems dramatically decreases the demand for analog circuit-based hardware. FPGAs are digital integrated circuits with a wide range of applications which are made up of links between programmable logic blocks. It's aimed at assisting the creator realize the logic functions that the designer needs. In consequence, the user may change the function of each logic block. VHSIC Hardware Description Language (VHDL) is commonly used in FPGA programming. In this study, VHDL code was created, and a FM transmitter was implemented on a FPGA board (CMOD A7) in this research. The sound card interface on the PC used to send and receive the signals while suitable ADC and DAC cards are used on the FPGA side for the same purpose. Audacity program was used to playback the sample wav files while HDSDR SDR program was used to monitor and record the signals in wav file format. Finally, using the MATLAB code, the recorded transmitter signal was demodulated offline, and the output was stored to the hard drive. The demodulated signal obtained is identical to the initial modulation signal, indicating that the modulation was correctly executed. As a result, a perfect foundation for the development and training of SDR systems using FPGA has been established.

Key Words : Sdr, Matlab, Fm, Tx, Fpga
Science Code : 90523

ÖZET

Yüksek Lisans Tezi

EĞİTSEL SDR TEKNİKLERİNE DAYALI FPGA TABANLI FM RADYO VERİCİ TASARIM VE UYGULAMASI

Ahmed Ibrahim M ALGHHRIUNI

Karabük Üniversitesi
Lisansüstü Eğitim Enstitüsü
Elektrik ve Elektronik Mühendisliği Anabilim Dalı

Tez Danışmanı:
Dr. Öğr. Üyesi Bilgehan ERKAL
Aralık 2022, 75 sayfa

Yazılım Tanımlı Radyo (SDR) sistemlerinin temel amacı, radyo sinyallerini sayısal olarak işlemek için bir dijital sinyal işlemcisi kullanmaktır. Bu sistemlerde modülasyon, demodülasyon, sinyal oluşturma ve hat kodlama gibi görevleri gerçekleştirmek için Alan Programlanabilir Kapı Dizisi (FPGA) gibi bir işlemcinin kullanılması, analog devre tabanlı donanıma olan talebi önemli ölçüde azaltır. FPGA'lar, programlanabilir mantık blokları arasındaki bağlantılardan oluşan çok çeşitli uygulamalara sahip dijital entegre devrelerdir. Yaratıcının, tasarımcının ihtiyaç duyduğu mantık işlevlerini gerçekleştirmesine yardımcı olmayı amaçlar. Sonuç olarak, kullanıcı her bir mantık bloğunun işlevini değiştirebilir. VHSIC Donanım Tanımlama Dili (VHDL), FPGA programlamada yaygın olarak kullanılır. Bu çalışmada VHDL kodu oluşturulmuş ve bu araştırmada bir FPGA kartına (CMOD A7) bir FM vericisi uygulanmıştır. Sinyalleri göndermek ve almak için PC üzerindeki ses kartı arayüzü kullanılırken, FPGA tarafında ise aynı amaç için uygun ADC ve DAC kartları kullanılmaktadır. Örnek wav dosyalarını oynatmak için Audacity programı, sinyalleri izlemek ve wav dosya formatında kaydetmek için HDSDR SDR programı kullanıldı. Son olarak, MATLAB kodu kullanılarak, kaydedilen verici sinyali çevrimdışı olarak demodüle edildi ve çıktı, sabit sürücüye depolandı. Elde edilen demodüle edilmiş sinyal, modülasyonun doğru bir şekilde yürütüldüğünü gösteren ilk modülasyon sinyaliyle aynıdır. Sonuç olarak, FPGA kullanan SDR sistemlerinin geliştirilmesi ve eğitimi için mükemmel bir temel oluşturulmuştur.

Anahtar Kelimeler : Sdr, Fpga, Matlab, Fm, Tx
Bilim Kodu : 90523

ACKNOWLEDGMENT

The present work is an effort to do the study on “Design and Implementation of an Educational FM Transmitter with FPGA using SDR Techniques”. The work would not have been possible to come to the present shape without the able guidance and supervision of my thesis mentor Mr. Assist. Prof. Dr. Bilgehan ERKAL.

Also, I wish to convey my thanks to my family members for their morale during the course of study.

CONTENTS

Page

Page

LIST OF FIGURES

LIST OF TABLES

ABBREVITIONS INDEX

ABBREVITIONS

SDR	: Software Defined Radio
FPGA	: Field Programmable Gate Array
VHDL	: Very High-Speed Integrated Circuit Hardware Description Language
ICNIA	: Integrated Communication Navigation Identification and Avionics System
DSP	: Digital Signal Processing
MMITS	: Modular Multifunction Information Transfer System
DUC	: Digital Up Converter
IF	: Intermediate Frequency
DAC	: Digital Analog Converter
ADC	: Analog Digital Converter
DDC	: Digital Down Converter
FIR	: Finite Impulse Response
CPLD	: Complex Programmable Logic Device
PROM	: Programmable Read Only Memory
AM	: Ampiltude Modulation
FM	: Frequency Modulation
RTL	: Register Transfer Level

CHAPTER 1

INTRODUCTION

Software Defined Radio (SDR) is a combination of programmable hardware and software technologies developed for wireless communication. John Mittola proposed the first concepts for software defined radio in 1991, with the idea that radios may be set and programmed in software. Other hardware devices perform actions such as signal lowering/amplifying, modulation/demodulation, and filtering in a traditional hardware radio system. In the software radio, on the other hand, there is a programmable system that the user can change the location of these elements at any time. The development of SDR systems has offered benefits such as cheaper costs and modifying the functionality of hardware-based radios [1-4].

FM radios are commonly used to broadcast audio signals. For digital communication systems with a narrow bandwidth and a correspondingly low receiver sensitivity, they are also an option [5].

Software defined modems are becoming increasingly popular as an alternative to traditional analog modems due to a number of significant advantages, including the ability to be reprogrammed, flexibility, and a cheap cost [6].

FM implementation using analog circuits suffers from a number of deficiencies, the most notable of which are non-linearity caused by the voltage-controlled oscillator (VCO), as well as poor stability performance [7].

As a result of recent advancements, the growing availability of low-cost digital signal processing integrated circuits has acquired a significant amount of importance for the design of digital FM. In addition to that, their noise figure performance is remarkable, and they have an amazing voice clarity.

For software-defined FM modulation and demodulation, FPGAs may create digital Numerically Controlled Oscillators (NCOs) and high order filters.

Based on the signal that is emitted by the FM radio, two observations are possible.

The FM signal's amplitude, independent of the message signal, is maintained at a constant level, which results in a characteristic called a constant envelope. Furthermore, the frequency-modulated output is based nonlinearly on the message signal. As a result, the FM signal qualities are difficult to examine. The FM signal bandwidth, on the other hand, may be approximated using a tone message signal that represents the number of efficient sidebands. A message signal can be extracted from an FM transmission via frequency demodulation. It has a frequency discriminator that works as a differentiator with a specialized envelope detector [7-8].

1.1. LITERATURE REVIEW

The first wireless communication was found in the late 1980s, and since then, various breakthroughs in radio communication technology have evolved to ensure radio users' connectivity. The Triumphant radio, which was created in the 1930s and employed voice communication because of bandwidth limitations at the time, is the earliest sort of transmission. Then, in the 1950s, broadcast communication became mainstream, with analog television communication using a large amount of bandwidth and providing exceptional customer service. In the 1960s, computers were more widely used, and they were able to transport data across large distances through cable and wireless connections. Following the introduction of cell phones, wireless voice communications were found, allowing transmission from any location. Nonetheless, because the mobile gadgets were not portable, they were difficult to utilize [9].

Ali HANDER designed an AM receiver using SDR methods and implemented it on an FPGA. The study's goal was to create a simple and inexpensive FPGA-based platform for teaching the fundamentals of SDR. To create the simulation environment, the researchers employed MATLAB programs. A signal was utilized in the simulation to evaluate the FPGA implementation. In addition, the simulation code serves as a framework for the FPGA-based SDR system's VHDL architecture. Another MATLAB script was written by the researcher to examine the simulation and test findings and compare them. Because the greater the SNR ratio, the better, the test results on the two signals revealed that A1 tests signals are better than A2 tests signals. When the actual real-world values were compared to the simulations of each test signal, it was discovered that the real-world SNR findings were somewhat lower than the simulations SNR. SNR value greater than 20dB is regarded as an acceptable level for an AM receiver. The FPGA AM RX system has shown to be a promising contender for AM demodulation and reception based on test and simulation findings. Furthermore, the planned and implemented FPGA AM RX was effective in teaching the fundamentals of basic SDR, which was the study's major focus [10].

In [11] authors utilized MATLAB algorithms to build and implement an AM radio transmitter simulation in an FPGA. Later, using ISE Design Suite 14.7, VHDL code was written and an amplitude modulated transmitter was constructed on the FPGA board (Mimas Spartan 6). The sound card was utilized to send the sample sound that was used in modulation using the Audacity programs to the FPGA card. The ADC (LM4550) card provides an analog signal to the FPGA card, which is then received, demodulated, and recorded using the HDSDR application. The transmitter signal is created in analog form by the FPGA card and delivered to the microphone input of the laptop's sound card through the DAC (LM4550) card. Finally, the recorded transmitter is demodulated offline using the MATLAB code, and the output is stored to the hard disk. The findings of the investigation revealed that there is only a little variation between the simulation and real test results for the same test signal. Because the signal in the genuine test result has been subjected to noise, this is seen to be extremely plausible. Similarly, according to the SNR values obtained, the average value is approximately 20dB, which can be regarded as an acceptable value for an AM receiver. Furthermore, the system is thought to be an excellent platform for implementing and training FPGA SDR systems [11].

Hikmat N. Abdullah created a design approach and the implementation suggested an SDR system using an Altera Cyclone II family board, as well as Embedded MATLAB blocks and MATLAB/Simulink. The design was originally implemented in the

MATLAB/Simulink environment, and then translated to VHDL using the Simulink HDL coder. The design has been synthesized and loaded onto an Altera Cyclone II FPGA board using Quartus II 9.0 Web Edition® software. The findings of the study revealed that using programmable logic tools, the implementation of SDR may be readily produced and understood. In addition, the research revealed an efficient design method for obtaining VHDL netlists that may be downloaded to FPGA boards [12].

CHAPTER 2

SOTWARE DEFINED RADIO (SDR)

The main goal of software defined radio (SDR) systems is to process radio beacons entirely digitally using a digital beacon processor. The use of a digital signal processor (DSP) that can handle digital signals and a processor such as field programmable gate arrays (FPGA) to execute operations like as modulation, demodulation, signal creation, and line coding in these systems minimizes the requirement for analog circuit-based hardware [13].

First, in the early 1970s, work on software-based devices that could be programmed by communications engineers began. During these years, the United States produced a system called ICNIA. This system was one of the first to use DSP-based programmable modems.

In the 1980s, prototyping of programmable digital baseband radios began.

The term 'software radio' was first coined in 1984 by a team working at E-Systems. This expression was used for a digital baseband receiver that the team produced. By 1992, Joseph Mittola coined the term 'soft radio' to refer to his GSM base station project. Unlike the expression put forward in 1984, Joseph Mittola used the term software radio for an entire system including a transmitter and a receiver [1].

In 1996, the first association was established on behalf of SDR. Although it was originally called MMITS Forum, its name was changed to SDR Forum in 1998.

SDR is structurally composed of two subsystems, analog and digital. The analog system consists of an RF bandpass filter, a microwave switch that positions the antenna as a transmitter and a receiver, a low-noise amplifier, an RF power amplifier and a reference frequency generator. The analog system includes modules that cannot be implemented digitally [3].

The digital system, on the other hand, consists of software running on hardware. The software is layered so that the hardware can be separated from the software. Special software is created to create the layered structure. The software operating system, on the other hand, includes software such as hardware drivers, resource management. The software defined radio architecture is shown in Figure 2.1 [4].

Considering the working process of the system, firstly, the digital data is coded and modulated in the transmitter. The data is then inserted into the digital up converter (DUC) and the digital baseband samples are converted to mid-frequency (IF) samples. These samples are transferred to a digital to analog converter (DAC) and an analog IF signal is obtained. Then, this signal is inserted into the RF up converter and the RF signal is obtained.

At the receiver, the incoming RF signal is amplified first and then converted to an analog IF signal. An analog to digital converter (ADC) converts the IF signal into digital samples. The signal is then inserted into the digital down converter (DDC) and the incoming signal is converted to a baseband signal. Finally, the bandwidth of the signal is limited by the finite impulse response (FIR) filter [17].

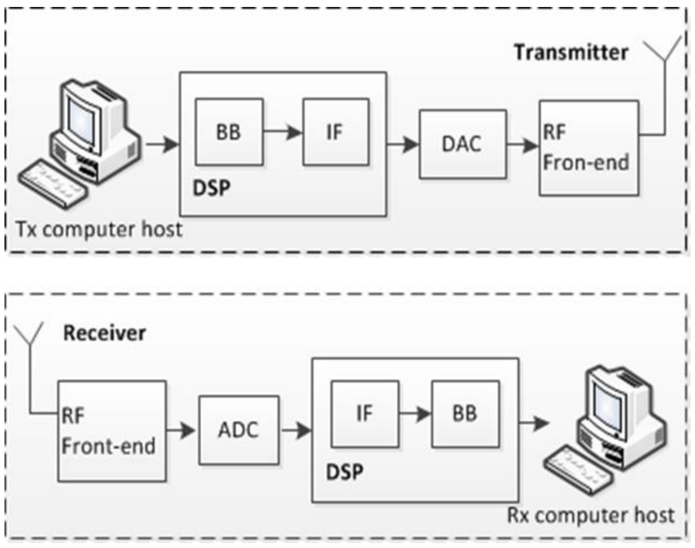


Figure 2.1. Software Defined Radio architecture transmitter and receiver.

2.1. SDR ADVANTAGES

- Enables reconfigurability of communications systems (adapting to new standards).
- Flexibility.
- Adaptable & upgradeable.
- Filters (other hardware).
- Cognitive (Smart Radio).

2.2. SDR DISADVANTAGES

- The RF hardware might need to be integrated into various sections of circuitry meant for different frequency ranges in order to provide very broad frequency coverage.
- The RF performances that can be achieved are constrained by technology.
- Instead of hardware constraints, software reliability may be what determines total radio dependability.
- Reliant on a computer.
- Limits of Software.

2.3. APPLICATIONS OF SDR

A communication link between the Software Defined Radio (SDR) and the computer must be established before other applications of SDR technology may be discovered. The technology's capacity to fix mistakes in real time has become the norm. However, the device is also related to other applications, such as dynamic spectrum positioning, cost minimization and spectrum control, and opportunity driven multiple access. SDR implementations are cheaper than conventional analogue devices, according to research [18].

Particularly in telecommunications applications like GPS signal reception, HF propagation, and the interpretation of cellular technology emissions, SDR technology has raised significant concer.

CHAPTER 3

FIELD PROGRAMMABLE GATE ARRAYS (FPGA)

Field Programmable Logic Arrays (FPGAs) are digital circuits that connect programmable logic blocks and have a wide range of applications.

It was created with the intention of implementing logic functions for the user's needs. As a result, the designer can modify the functionalities of the logic blocks. By using FPGA, the functionality of basic logic gates and complex circuit elements is noticeably increased [19].

Looking at its historical process, FPGAs were first used in the 1980s, usually for intermediate adhesive logic and constrained data processing tasks. In the 1990s, it started to be used in network and communication environments that required extensive data processing thanks to its increased capacity. At the end of the 90s, its use in the automotive and industrial sectors showed a great growth. In the early 2000s, high-performance FPGAs containing millions of gates entered the market and today they find a wide place in more than one market branch [20, 21].

3.1. FPGA INTERNAL STRUCTURE

The internal structure of the FPGA consists of three parts, namely the input-output blocks, the interconnects, and the logic cell. Figure 3.1 shows the internal structure of the FPGA.

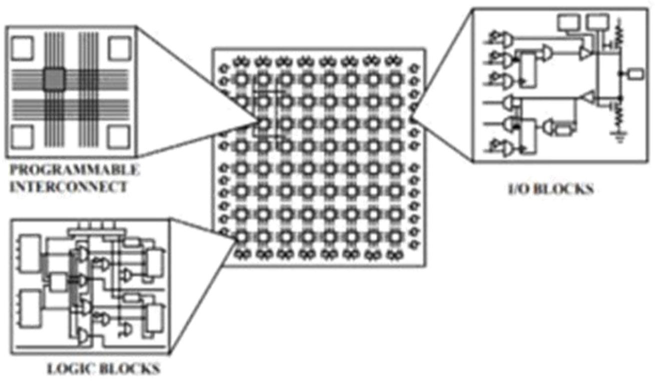


Figure 3.1. Internal structure of FPGA.

3.1.1. Logic Cell

Logic cell is the basic structure of FPGA. It consists of one D-type FF and LUT and one 2x1 Mux. The logic cell is shown in Figure 3.2.

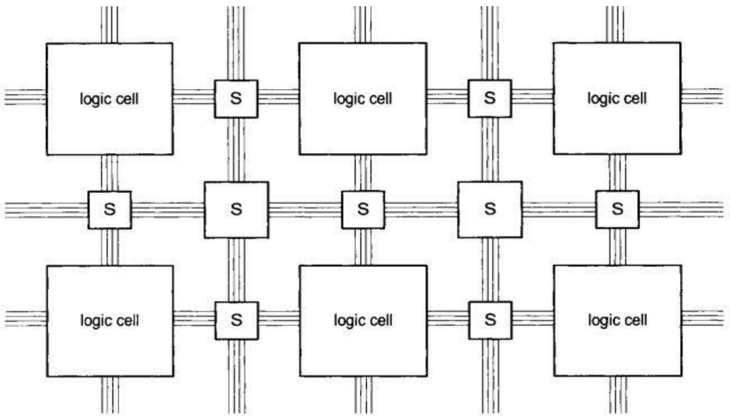


Figure 3.2. Logic cell.

LUTs are mini memories that perform logic operations. A LUT with N inputs creates a memory with 2^N. Interconnections in the logic cells are realized with matrix-shaped data buses and programmable switches.

3.1.2. FPGA Pins

FPGA pins are often classified as dedicated or user pins. Pins are separated into three groups according on their functions: power pins, configuration pins, and clock pins.

The power pins supply the FPGA's energy requirements. The software may be loaded into the FPGA via configuration pins. Clock pins are used to transmit clock signals. The designer specifies the conventional input/output pins as user pins [18].

3.2. FPGA PROGRAMMING

In order to program the FPGA, graphic design and HDL are utilized. The compiler program library's parts and logic are used to program the graphical design. Design in HDL; VHDL or Verilog are used for programming [19, 20].

Very High Speed Integrated Circuit Hardware Description Language is another name for VHDL. Since the 1980s, it has undergone continual development and is recognized as a standard by the IEEE [21].

The use of VHDL has two main purposes;

Synthesis: It is used to generate the codes to be loaded into the FPGA.

Simulation: It is used to simulate the codes to be loaded into the FPGA.

3.3. FPGA FLOW DIAGRAM

The steps to be followed while creating the design are given in Figure 3.3.

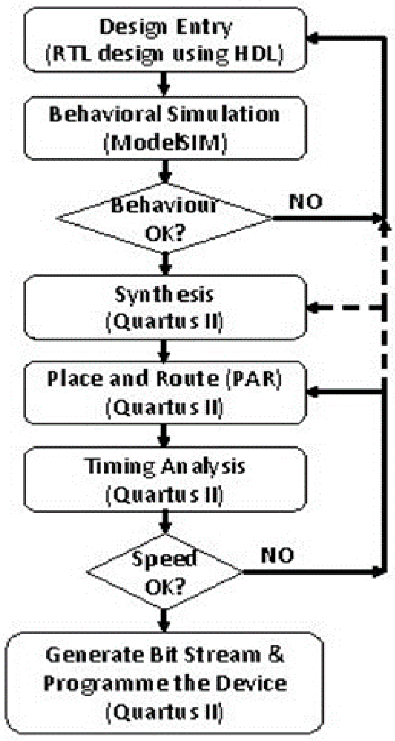


Figure 3.3. FPGA design flow.

3.4. FPGA MANUFACTURERS

Altera and Xilinx are FPGA manufacturers that are in high demand in the market. Actel, Latice and Quicklogic companies can be counted as other important manufacturers.

3.4.1. Xilinx

It is the biggest corporation in the international market and the first company to create FPGA. As a compiler, it provides the ISE Design Suite software. It is a producer of FPGA and CPLD devices with several uses and products in industries including communications, defense, automotive, and consumer goods. PROM devices, interface circuits (CoolRunner), low-cost circuits (Spartan), and high-performance FPGA chips are all produced by Xilinx (Virtex). Ross Freeman, Bernard Vonderschmitt, and James Barnett started it in the USA in 1984.

3.4.2. Altera

The greatest rival of Xilinx, the company that created the FPGA, is Altera. In 1984, they initially hit the market. The Intel corporation bought Altera in 2015. They gave the designers the Quartus II software as a compiler. The business offers the Stratix, Cyclone, and Aria ranges.

CHAPTER 4

FREQUENCY MODULATION (FM)

Adjusting the Frequency Modulation through frequency shift (FM) is a method of encoding data onto a carrier wave. These applications include computer systems, signal processing, radio transmission, and telecommunications.

In analog frequency modulation, such as radio transmission, an audio signal representing voice or music has a functional connection between the modulating signal amplitude and the instantaneous frequency deviation, or difference between the carrier frequency and its center frequency.

Frequency-shift keying (FSK) is a method for encoding and transmitting digital data that uses a type of frequency modulation in which the carrier's instantaneous frequency is shifted among a range of frequencies. The frequencies might be used as a substitute for the digits 0 and 1. FSK is used in a wide variety of computer modems, such as fax modems, caller ID telephone systems, garage door openers, and other low-frequency devices, and it is also used in radio teletype.

Frequency modulation is widely used in FM radio broadcasts. Other applications include the monitoring of infants for convulsions, telemetry, radar, seismic prospecting, EEG, two-way radio, sound synthesis, magnetic tape-recording, and specific video transmission systems. Due to its greater signal-to-noise ratio, frequency modulation is superior to an amplitude modulation (AM) signal of the same intensity when it comes to rejecting radio frequency interference during transmission. Because of this, most radio stations only play music on FM.

Angle modulation can be accomplished using either frequency modulation or phase modulation, with the latter typically being used as a prelude to the former. Amplitude modulation, in contrast, maintains a constant frequency and phase but varies the carrier wave's amplitude.

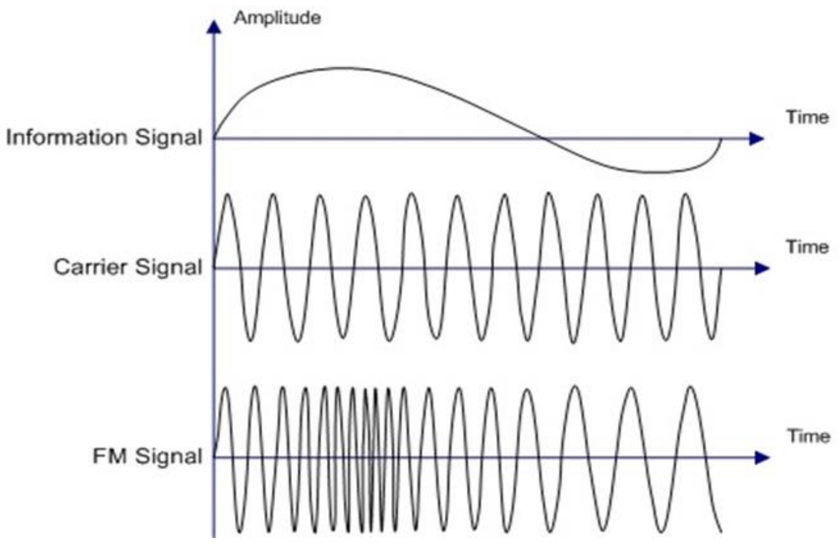


Figure 4.1. Information signal, carrier signal and frequency modulation signal.

4.1. OVERVIEW AND THEORY OF FM SIGNALS

In reality, FM signals differ from AM signals in a number of ways. Figure 2.4 below illustrates how the carrier wave frequency of an FM signal changes in response to the signal strength while maintaining a constant modulated wave amplitude. As can be seen in the following diagram, the carrier frequency doesn't change whether the signal's voltage is equal to zero, A, C, E, or G points. The carrier frequency, on the other hand, rises to its highest value at the positive peaks, B and F, while falling to its lowest value at the negative peaks [22].

Both the FM signal's carrier frequency and its amplitude can be adjusted according to the strength of the modulating signal. This suggests that the magnitude of the instantaneous modulating signal is linked to the frequency variation. The rate of change is proportional to the frequency of the underlying signal. In reality, the peak voltage of the modulating signal is where the frequency fluctuation is the largest [22].

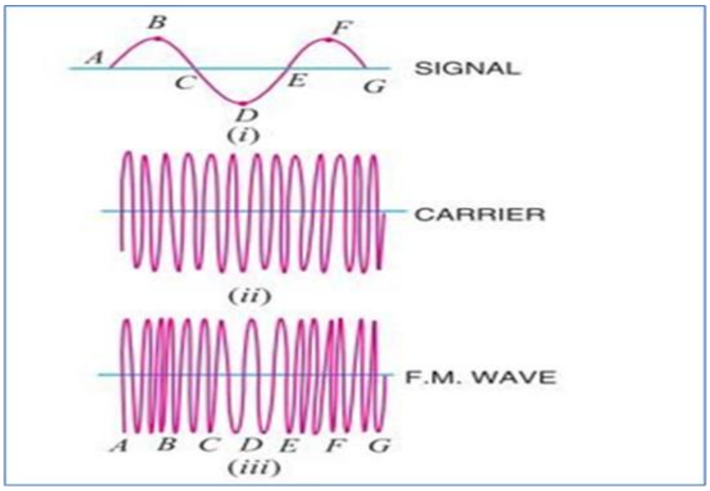


Figure 4.2. Illustration of FM signal.

4.1.1. Information Signal

The information (message) signal is of low frequency. The information signal is shown in Figure 4.3.

Mathematically:

Vm = vmsin2πFmt

Vm = Message signal instant value

vm = Message signal maximum value

Fm = Message signal frequency

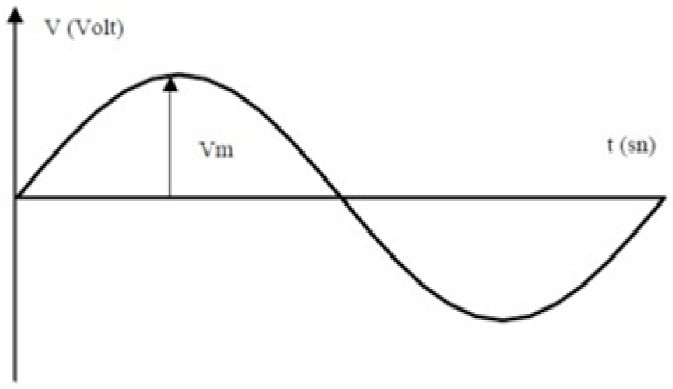


Figure 4.3. Information signal.

4.1.2. Carrier Signal

The carrier signal is a high frequency sin/cos signal. Shown in Figure 4.4.

Mathematically:

$Vc = vc \sin 2\pi Fc t$

Vc = Instantaneous value of carrier signal

vc = Maximum value of carrier signal

Fc = Frequency of carrier signal

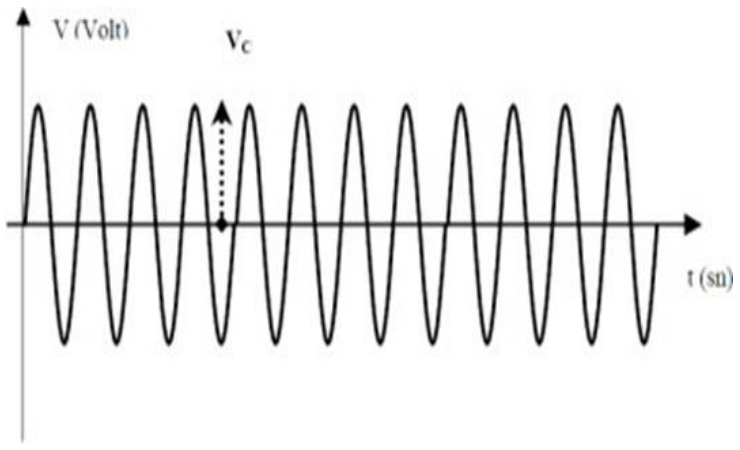


Figure 4.4. Carrier signal.

4.1.3. Frequency Modulated Signal

.

Figure 4.5 shows the frequency modulated signal.

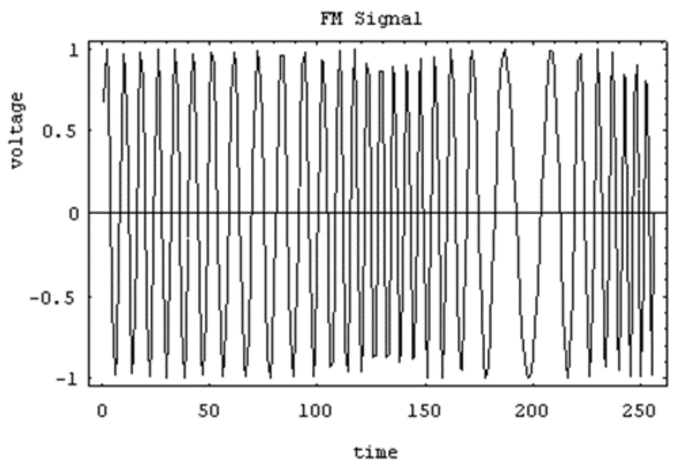


Figure 4.5. Frequency modulated signal.

4.2. COMPARISON BETWEEN FM AND AM SIGNALS

Table 4.1. Comparison among FM and AM.

FM signals	AM signals
Throughout the whole modulation process, the carrier amplitude remains constant	Throughout the whole modulation process, the carrier amplitude fluctuates
The value of the modulation index may surpass	The value of the modulation index should not
The modulating signal intensity determines how much the carrier frequency changes	The modulating signal intensity determines the amplitude fluctuation

4.3. BENEFIT SIGNALS OF FM OVER SIGNALS OF AM

Compared to AM signals, FM transmissions have a number of benefits, some of which are shown below: [22]

- FM signals operate across a much wider area.
- High-fidelity reception is provided by FM signals.

- FM signals are more effective in transmission.

An FM receiver may reject noiseless FM signals by filtering out noises, which are amplitude changes.

4.4. FM SIGNALS MODULATION AND DEMODULATION

4.4.1. FM Modulation

The carrier frequency is altered by the modulating signal's amplitude in frequency modulation (FM) (i.e., intelligence signal). The FM signal may be described by the following equation:

$$xFM\ (t) = Ac \cos\ (\theta t) = Ac \cos\ (2\pi fct + 2\pi f\Delta \int x\ (\lambda)\ d\lambda) \tag{4.1}$$

If $x\ (\lambda) = Am \cos\ (2\pi fm\lambda)$, then:

$$xFM\ (t) = Ac \cos\ (2\pi fct + \beta \sin\ (2\pi fmt)) \tag{4.2}$$

Where;

θ (t) = instantaneous modulated frequency
fc= carrier frequency
fm= modulating frequency
β = modulation index = $(f\Delta/fm)$
The frequency of FM signal (t) may be expressed as:

$$f\ =\ 1d\ \theta(t)\ /\ 2\pi\ dt\ =\ f_c - f_m\ \ \beta\ cos\ (2\ \pi\ f_m\ t) \tag{4.3}$$

The frequency of a frequency-modulated signal deviates from the carrier frequency due to variations in intelligence amplitude, as shown in (4.3).

4.4.2. FM Demodulation

- *The zero-crossing-counter demodulator*

By averaging the times at which zero crossings occur in the FM signal, we may construct a simple yet effective FM demodulator. An example of this concept is shown in Figure 4.6.

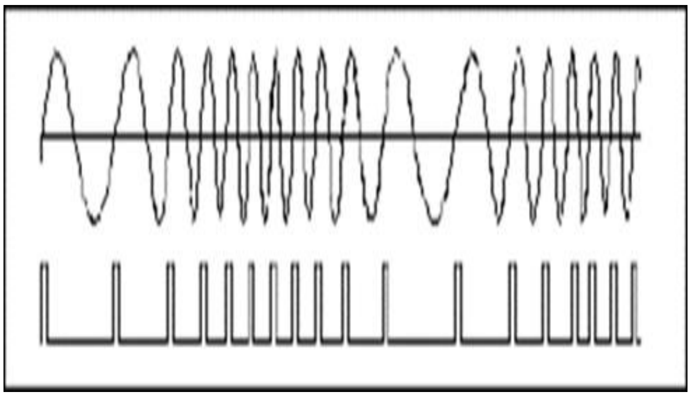


Figure 4.6. An FM signal and a series of zero-crossing pulses.

Each pulse in the train occurs at the precise instant when the FM signal goes through zero. It is the pulse repetition rate that alters this signal. When processing the pulse train with a low pass filter, the data will be averaged. The message's repetition rate affects the pace at which this average value shifts, while the generator's depth of modulation establishes the size of this shift.

In the next stages of the experiment, a model of this zero-crossing-counter demodulator will be utilized to demodulate low frequency FM. Following this, we'll have a look at the phase locked loop (PLL) and its role as a demodulator.

- *FM Demodulation with the PLL*

A non-linear feedback loop is the phase locked loop. It is a challenging activity to accurately assess its performance. It is easy to depict it in simpler block diagram form. As show Figure 4.7.

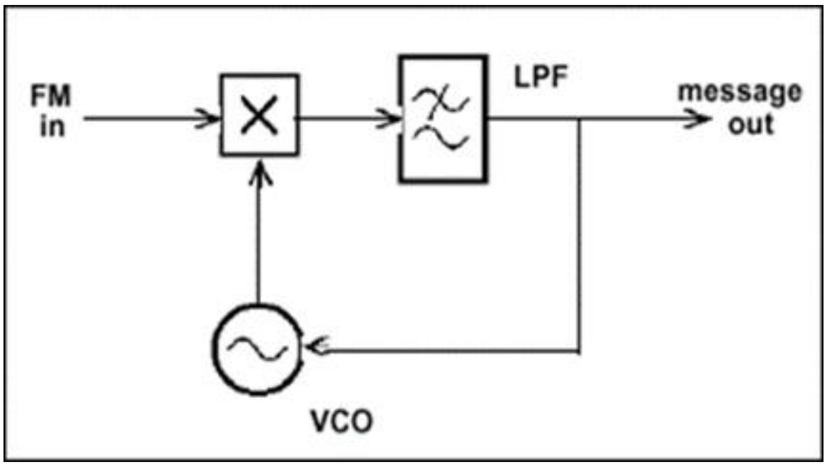


Figure 4.7. The basic PLL.

There are several uses for this configuration. The output was collected from the VCO in order to acquire the carrier. The output is collected from the LPF and used as an FM demodulator, as illustrated. It is easy to explain the PLL's basic mode of operation as a demodulator, but it is more difficult to conduct a thorough investigation of its performance. Its performance is governed by non-linear equations, whose solutions typically involve approximation and compromise. This complicates the situation.

It would seem that the operating concept is straightforward. Consider about the open-loop configuration of Figure 4.7 What this means is that the connection between the control voltage input to the VCO and the filter output has been severed.

Let's pretend the input is an unmodified carrier.

This setup seems like a product or multiplier-type demodulator. In a perfectly tuned VCO, the incoming carrier's frequency would be precisely matched, and the VCO's output would be a DC voltage whose amplitude is proportional to the phase difference between the VCO and the carrier.

The voltage would be exactly zero volts DC for two of the possible 360-degree angles. Suppose the VCO's frequency started to drop down over time. The resulting voltage would be a slowly changing alternating current, which, if sufficiently slow, might be misunderstood for a DC with varying amplitude. This DC voltage will take on a positive or negative value depending on the drift direction.

Assume the loop in Figure 4.7 has closed. By changing the sign of the slowly varying DC voltage, now a VCO control voltage, the VCO may be "locked on" to the incoming carrier frequency w0. This is the procedure for acquiring a carrier.

Let's pretend for a moment that the frequency of the incoming wave is being modulated. For a low-frequency message, it is acceptable to expect the VCO to make some effort to track the incoming carrier frequency with little change. Consider the possibility of using wideband FM. With "correct design" of the low pass filter and VCO circuits, the VCO will also track the incoming carrier in this case.

The VCO's control voltage will seek to generate a message that is a perfect match for the incoming carrier so that the VCO can retain its frequency lock on it.

4.5. QUADRATURE DEMODULATOR

It is the most extensively used single "FM" demodulator, with a phase shift circuit producing a "90 degree" phase shift at the unmodulated carrier frequency. This detector is mostly employed in television demodulation, although it is also utilized in some "FM" radio stations.

A relatively small number of components were needed to construct the quadrature detector; nonetheless, a coil was essential. Nevertheless, due to the fact that the inductor is merely a coil and not a transformer, the cost impact may be handled for many different designs of radio receivers. Quadrature detectors, like its related coincidence detectors, provide great performance with good linearity.

The signal application mode application employs the "FM" signal mode, where the value of the center frequency is decided by the "Q" of the tuned circuit, and the output longitude is determined by altering the current in the multiplier as a function of the "IF" signal deviation.

In this study, used in demodulating the FM signal in the MATLAB code.

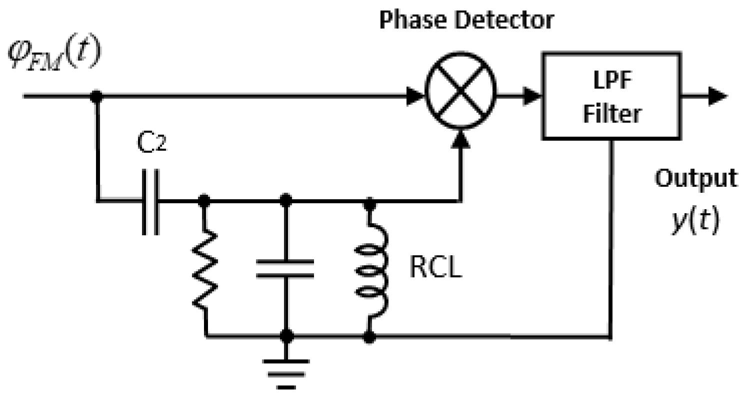


Figure 4.8. Quadrature demodulator.

.

CHAPTER 5

VHDL – HARDWARE DESCRIPTION LANGUAGE

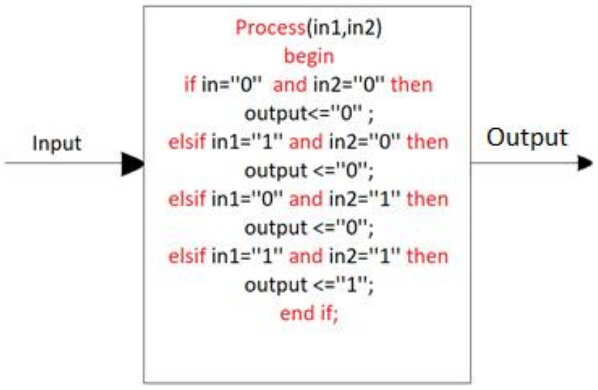
Very High-Speed Integrated Circuit Hardware Description Language is the literal meaning of VHDL. It might be thought of as a language for use in very fast computer components. There has been steady growth in this area since the 1980s, and it has been recognized by the IEEE as well.

5.1. VHDL TERMINOLOGY

Hardware description language (HDL) refers to the software used to simulate hardware resources. The HDL language offers the ability to use software to configure hardware and describe hardware behavior. The HDL language most frequently used in FPGA programming is VHDL.

5.1.1. Behavioral Modeling

The input-output responses in the model are defined behaviorally. It is not concerned with its internal structure. The function and function of the circuit is important. Behavioral modeling is shown in Figure 5.1.



```
Process(in1,in2)
begin
if in="0"  and in2="0" then
output<="0" ;
elsif in1="1" and in2="0" then
output <="0";
elsif in1="0" and in2="1" then
output <="0";
elsif in1="1" and in2="1" then
output <="1";
end if;
```

Figure 5.1. Behavioral modeling.

5.1.2. Structural Modeling

Shows the relationships between a component and its subordinate components. Structural modeling is based on the designer constructing the structure of the model. Most of the time, the preferred technique in VHDL designs is to create the modules at the bottom, which are modulated differently from each other, by structural modeling, and connect them to the modules at the top with structural modeling. Structural modeling is shown in Figure 5.2.

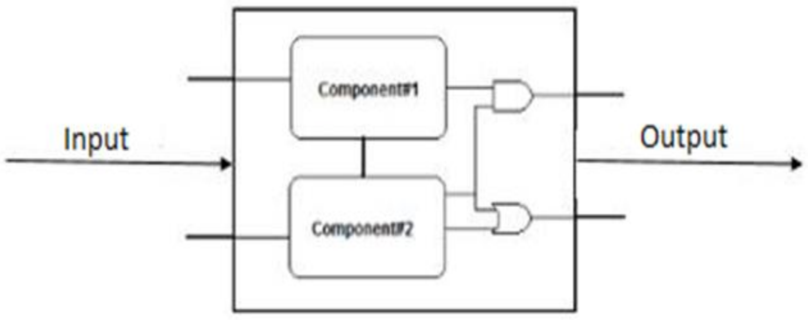
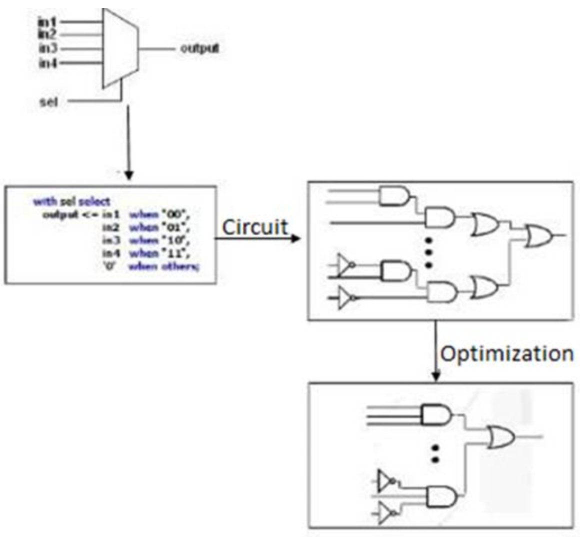


Figure 5.2. Structural modeling.

5.1.3. Register Transfer Level

RTL is a synthesis-oriented abstraction approach. RTL is the design of a generated piece of code represented in registers. Simply expressed, they are logic gate-based circuits that correlate to our VHDL code.

In order to perform RTL synthesis, the VHDL code must first be translated into a digital circuit. Then, the corresponding VHDL-coded circuit is optimized by making the most of the available FPGA resources. Specifically, as seen in Fig. 5.3, an example RTL model is displayed. In this example, creating a 4-input, 1-output MUX is requested. The VHDL code is produced first. The compiler then transforms the code into the matching digital circuit. Finally, the RTL flow is finished and this code, which transforms into a digital circuit, is optimized.



```
with sel select
  output <= in1  when "00",
            in2  when "01",
            in3  when "10",
            in4  when "11",
            '0'  when others;
```

Figure 5.3. Example RTL modeling.

5.2. VHDL DESIGN

VHDL design consists of three parts: coding, simulation and synthesis. The coding part is the part of the program where the VHDL code is generated. In the simulation part, the VHDL code is simulated and it is observed whether the program is correct. In the synthesis part, the written VHDL code is translated into hardware language and the RTL scheme is extracted. The code is then converted by the compiler into a configuration file that will be loaded into the FPGA. The VHDL design flow as seen in Fig. 5.4.

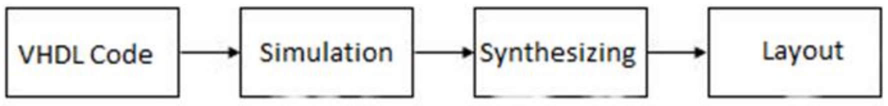


Figure 5.4. VHDL design flow.

5.3. VHDL DESIGN SETIONS

A VHDL design; the entity consists of 5 parts: architecture, package, component and process.

5.3.1. Entity

It's the smallest building block possible for the overall layout. In other words, it describes how the design interacts with the outside world. The input and output channels are specified here.

5.3.2. Architecture

It is used to specify the model's intended purpose. An entity can have more than one architecture. An architecture can be used in three different ways: behavioral modelling, structural modeling, and data flow.

5.3.3. Package

The package groups the definitions used by the entity and also serves to group them for use in different designs.

5.3.4. Component

The component structurally defines the name and interface of the component used as a subcircuit in the circuit description.

5.3.5. Process

The transaction block contains states that will occur sequentially. In an architecture, multiple transaction blocks are executed instantaneously. Transaction blocks start at the same time and each transaction block is executed in line with itself.

5.4. VHDL MODELING BASICS

5.4.1. Constant

Objects that cannot be changed after their initial value is determined. It is often used to increase the understandability of the code.

Form of display; Constant name:data type:=value;

5.4.2. Signal

Signals provide communication between processes within the architecture.
Signal identification; Signal name: type: =initial value;

If the signal is assigning a value <= symbol is assigned.

C/C++, equals in programming languages (=) performs the same function as the expression.

For all values;
Reg<="1100";
Reg<=x"C";(hexadecimal)
For single bit assignment;
Reg (2)<='1';
for bit slicing;
Reg (1 to 2)<="20";

5.4.3. VHDL Operators

All arithmetic and boolean operations in VHDL are restricted to the standard package of data types.

Arithmetic operators (+, -, <, >, <= ,>=) integer types also apply. Boolean operators (And, Or, Not) is applied for BIT types.

In other data types, special functions in the IEEE library are used for arithmetic and boolean operations.

Packages with special functions in the IEEE library;

std_logic_arith (arithmetic functions)
std_logic_signed (signed arithmetic functions)
std_logic_unsigned (unsigned arithmetic functions)
In the functions mentioned above, since the operator and the operator's name are the same, the function name is indicated with quotation marks.

5.4.4. Simultaneous Signal Assignments

There are 3 different ways for simultaneous signal assignments:

- Simple signal assignments.
- Conditional signal assignments.
- Selected signal assignments.

Simple signal assignments;

Display format: name of signal <= expression
Conditional signal assignments;
Display format: name of signal <= expression 'when' condition 'else',
Notation: name of signal <= expression 'when' condition 'else’,
expression 'when' conditional 'else', expression;
Selected signal assignments;
Notation: value <= expression 'when' choice,
expression 'when' choice, expression 'when' others;

5.4.5. Sequential Commands

The commands used in process, function, and procedure operations. Sequential commands are also used for simple signal assignments. The commands are as follows:

- If-then command
- Case command
- Loop command
- Wait command

CHAPTER 6

MATERIALS AND METHODS

6.1. DESIGN AND IMPLEMENTATION OF FPGA BASED FM TRANSMITTER

The implementation of the FM transmitter in FPGA has two stages: Hardware design and software design in VHDL.

6.1.1. Hardware Component

Hardware design part of the study incorporates PC, FPGA card, ADC and DAC cards. Figure 6.1 is a system block diagram depicting the components involved.

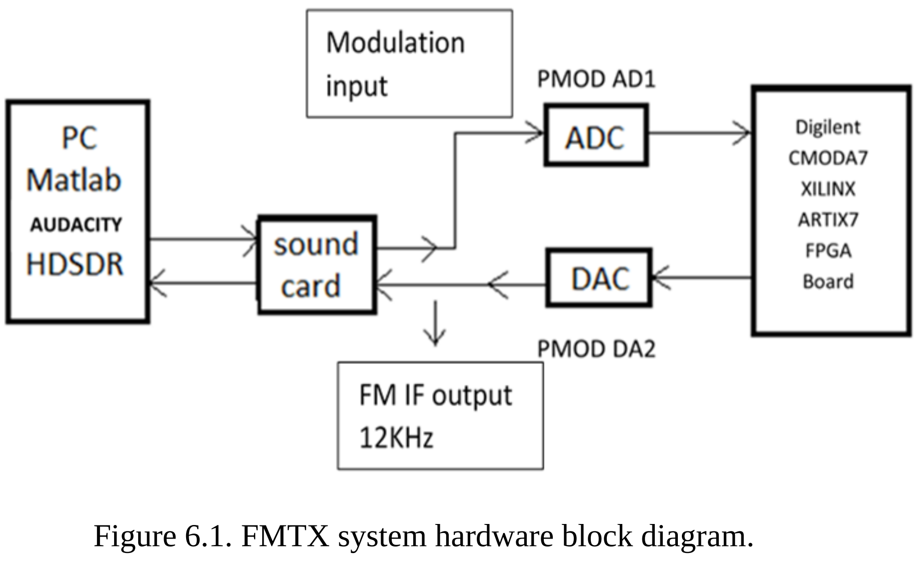


Figure 6.1. FMTX system hardware block diagram.

FPGA module in the system is Digilent CMODA7-35T which incorporates Xilinx Artix7 FPGA on it. The FPGA chip is XC7A35T in 1CPG236C package whose capacity is 20K-LUT with 225KB Block-RAM. The board has 512KB SRAM with 8-bit bus and 8ns access time, 4MB Quad-SPI Flash to hold FPGA programs and USB-JTAG programming facility which also supplies necessary power from the connected USB bus. The board is in DIP form where total of 48-pins provided at each side. The board provides 44 Digital GPIO pins with 3.3V logic capability. All the connection to the board is provided through a solder less breadboard.

ADC card is Digilent's PMOD-AD1 which uses a dual channel 12-bit, 1MSPs/channel sampling rate A/D converter chip AD7476 from Analog Devices. DAC card is Digilent's PMOD-DA2 which incorporates two 12-bit, 1MSPs DAC chip DAC121S101 from Texas Instruments. The interface to both cards is through a standardized PMOD connector which encapsulates a standard multi-channel Serial Peripheral Interface (SPI).

There are also an external USB-soundcard and connecting audio cables to carry analog test signals between PC and FMTX system where built-in soundcard of the PC is reserved for listening of the results.

The recorded test signal (modulation signal) is played back through the soundcard speaker output at a 48KSps rate repeatedly using Audacity. This analog signal is digitized through the ADC and then sent to FPGA board for processing. After processing and frequency modulating the signal by the internal structure of the FPGA, the digital FM IF output is sent to D/A card for conversion to analog at 48KSps rate. The internal structure of FMTX system is constructed through programming by the VHDL code whose details are given in the software development part. The VHDL code listing is provided on Appendix B. This analog output signal is taken through the microphone input of the soundcard and sent to PC for monitoring and recording. Demodulation, monitoring and recording of the modulated signal is through HDSDR SDR software environment on the PC. Fig 6.2 is a photo of the FMTX system.

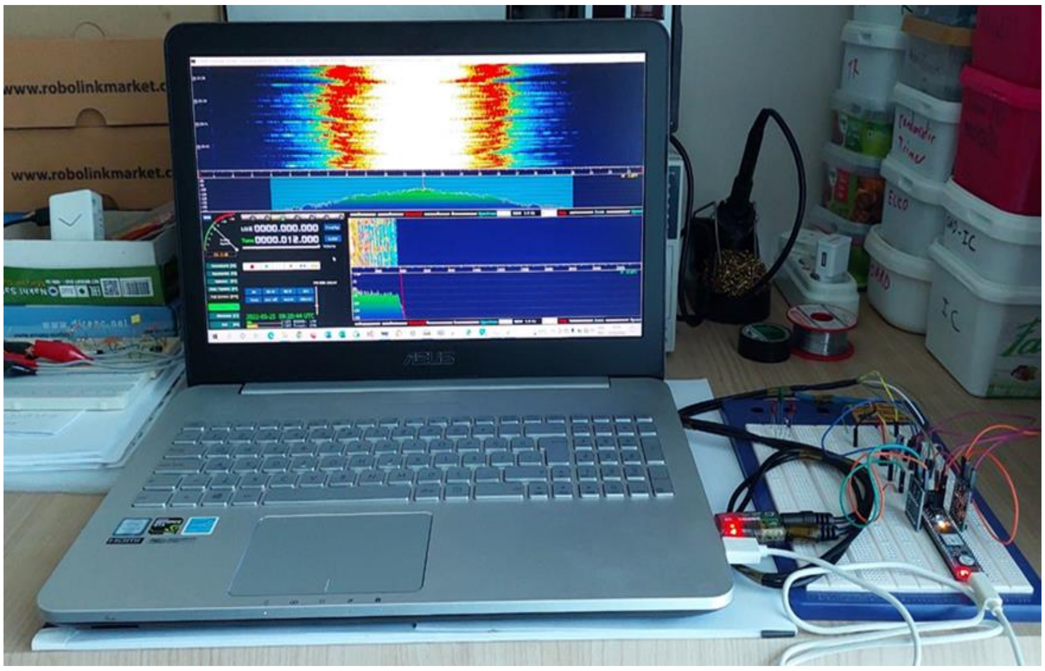


Figure 6.2. Photo of the FMTX system in operation.

6.1.1.1. Digilent CMODA7 FPGA Card

The Digilent Cmod A7 is a 48-pin DIP form factor board powered by a Xilinx Artix 7 FPGA. Furthermore, the board is equipped with standard input/output (I/O) pins, a USB-JTAG programming circuit, a USB-UART bridge, a clock source, a Pmod host connection, SRAM, and Quad SPI Flash. With these additions, it becomes a compact yet potent platform for digital logic circuits and Microblaze embedded softcore processor designs. With 44 Digital FPGA I/O signals and 2 FPGA Analog inputs connected to 100-mil through-hole pins, your programmable logic design may be added to a circuit without the need for any soldering. It may also be inserted into a regular socket and utilized in embedded systems because it is only.7" by 2.75" in size [23].

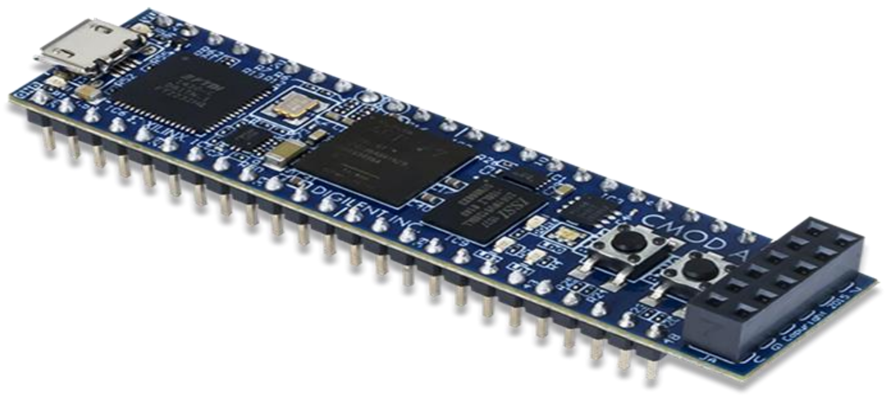


Figure 6.3. Digilent Cmod A7 FPGA board.

6.1.1.2. Digilent PMOD AD1

The Analog Devices AD7476A is used in the Digilent Pmod AD1 (Revision G), which is a two-channel, 12-bit analog-to-digital converter. With a maximum sampling rate of 1 million samples per second [24], this PmodTM can handle even the most taxing audio applications with ease.

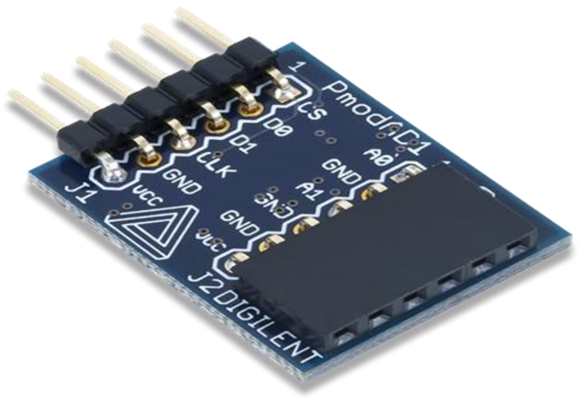


Figure 6.4. Digilent Pmod AD1.

6.1.1.3. Digilent PMOD DA2

There is a maximum output voltage of 16.5 MSa [25] from the two channels of the Digilent Pmod DA2 Digital-to-Analog Converter module.

Figure 6.5. Digilent Pmod DA2.

6.2. PROGRAMS

On the server PC machine, numerous third-party software was employed in this study. Brief information on them is provided in the following sections.

6.2.1. Audacity

Audacity is a free digital audio editing and recording software that can run on many platforms such as Windows, Mac OS and Linux. It was developed in 1999 by Dominic Mazzoni and Roger Dannenberg. If some of its features are mentioned, editing operations such as cut, paste and merge can be done on audio files. It supports Ogg Vorbis, WAV, MP3 file formats. Using some sound cards and Windows Vista, 7, 8 operating system, it can also record audio being played on the computer. Unlimited number of transactions can be reversed and forwarded. In this study, an audio file that had been edited and tested in the audacity application was then played over the computer's audio output. In addition, audacity was used to present test and simulation results for the FPGA FM transmitter system using the A1 modulating signal [26].

6.2.2. HDSDR

HDSDR is a free SDR tool that runs on Microsoft Windows.

Developed by Alberto Di Bene.

Some general features are;

- AM, ECSS, FM, SSB and CW modulation.
- The Tx output creates an I/Q modulated signal pair for the Tx input.
- The aliasing filter, bandpass filter, and squelch may all be adjusted.
- Timed recording and playback of RF, IF, and AF WAV files.

The modulated signal received in this study was visualized in the HDSDR program and also saved to the computer for later analysis. HDSDR program also made it possible to demodulate the modulated signal and listen via a second sound card [27].

6.2.3. Xilinx Vivado

Vivado Design Suite is a Xilinx software package for synthesis and analysis of hardware description language (HDL) designs that replaces Xilinx ISE with new capabilities for system on a chip development and high-level synthesis.

In this study, all the coding required to create the FPGA FM transmitter was done with VHDL using Vivado Design Suite.

6.2.4. Matlab

With hardware like a fast A/D converter and a powerful signal processor being necessary for a real-world SDR implementation, the usage of MATLAB in our research is crucial. This hardware platform is too expensive for students majoring in radio communication. Because of this, we used Matlab for our analysis, and the audio frequency range is the only one used for wireless signals. In a single Matlab session, we were able to set up the transmitter. All of the modulation and demodulation experiments are carried out in Matlab. The user must just select the desired modulation, demodulation, and related parameters to begin utilizing the system. [28]

6.3. SIMULATION STUDIES

Firstly, simulations are done in the study. Simulation of the FMTX system is done using suitable Matlab scripts which are listed on Appendix A. Simulation of the FM transmitter is realized with code listed on Appendix A.1. Then FM receiver simulation is carried over by the code listed on Appendix A.2. Analysis of the results are done using the code listed on Appendix A.3.

The transmitter code uses a 10s sample recording sampled at 8KSPs which is a 4KHz music recorded in wav format. This sample wav file is upsampled to 48KSPs to provide a 24KHz frequency modulated intermediate frequency waveform whose center frequency is 12KHz. This IF signal is then normalized and recorded as a wav file (FM.wav). The modulating signal is normalized and integrated before modulation. A modulation coefficient determines the maximum frequency deviation which sets the FM bandwidth according to Carson's rule. Maximum frequency deviation Fdmax is derived using the formula below:

$$F_{dmax} = \frac{A_{max}}{2.\pi.T_s} \tag{6.1}$$

Here,
Fdmax: Maximum frequency deviation (Hz),
Amax: Maximum amplitude (rad),
Ts: Sampling period (s)

Amax given in the code sets Fdmax as 4KHz. The integral of the signal is added into the carrier signal phase argument. Thus, an indirectly frequency modulated signal is derived through phase modulation which is a frequently used method in deriving FM in sampled systems. The modulated carrier is filtered through a bandpass filter whose center frequency is 12KHz and bandwidth is + 8KHz. The bandwidth is selected as 16KHz because Carson's rule gives us so as below equation suggests:

$$BW_{FM} = 2.(F_{dmax} + BW_m) = 2.(4KHz + 4KHz) = 16KH \tag{6.2}$$

So, the modulation index β, of the frequency modulated carrier is:

$$\beta = \frac{F_{dmax}}{BW_m} = \frac{4KHz}{4KHz} = 1 \tag{6.3}$$

Since β is equal to 1, the FM signal is said to be wideband. The modulated carrier, FM IF, being an infinite bandwidth signal is further bandpass filtered through a FIR filter in the code which limits signal to 4-20KHz range whose bandwidth is now limited but since it has 98% of its energy reside in the Carson bandwidth it continues to represent the modulating signal. Carson bandwidth is sufficient for a successful demodulation and reproduction of the original modulating signal. This filtering is mandatory to eliminate residual spectral components by bandpass filtering before transmitting to prevent interference to neighbouring stations.

So, a 16KHz wideband FM IF is obtained which is normalized and recorded as a wav file for further use in simulations and tests. In the simulations and tests two different music recordings are used which will be called as A1 and A2 and their FM results are FM1 and FM2 respectively.

To test the transmitter, receiver simulation is done through the receiver code (Appendix A.2). Firstly FM.wav file at 48KSPs is loaded. It is normalized and then sent for demodulation. Demodulation method used is quadrature FM demodulation. It is achieved by delaying the input signal by one sample and then multiplying itself. Selection of 12KHz as IF signal center frequency is not arbitrary. Phase interval between two samples corresponds exactly 90 degrees for an unmodulated 12KHz carrier sampled at 48KSPs. So, one sample delay represents 90 degrees phase delay. if we multiply delayed signal by non-delayed signal, we get zero DC level other than a high frequency component which is filtered after demodulation. This process provides a changing level if instantaneous frequency of the carrier is slightly changed by time. To get a demodulated signal, the strength and polarity of the output signal must be proportionate to the sign and magnitude of the frequency shift. Since the bandwidth of the modulating signal is 4 Kilohertz, the filter at the output is a lowpass FIR filter with a cut-off frequency of 4 Kilohertz. The replicated signal is standardised before being captured as a wav file for further examination.

Analysis of the results done using the code listed on Appendix.A3, provides us a comparison between original test signal with the signal resulted from demodulation. The result is pre-processed before the analysis operation. It is very important to time synchronize each signal that will be compared to obtain consistent results. Gain errors are corrected using a suitable amplitude scaling. It is possible to see the distortion and noise effects of demodulation over the signal after this pre-processing. Pre-processing of test results is done using the software tool called Audacity which is a easy to use and free audio processing software. It provides many functions for processing of audio files in wav format. The Audacity audio processing software is shown in Fig 6.6.

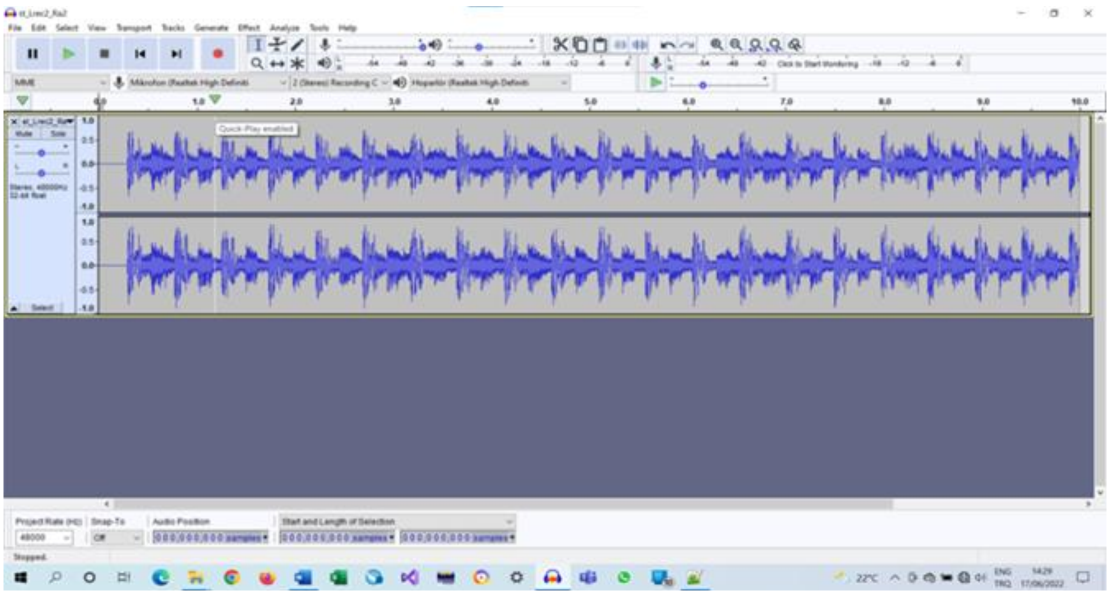


Figure 6.6 Audacity audio processing tool.

The analysis code uses a dual channel wav file. The input wav file's right channel contains the original signal while the left channel contains the resultant signal. The recording is prepared in the Audacity program. Both signals are added to the project file as stereo. After this both signals are normalized to same level (-1dB). As a last step, result signal is time synchronized to the original by setting a zero crossing as reference which is at the same time point. The synchronization is achieved by discarding enough samples at the beginning of the result signal. The unused parts at the end of the signals are also discarded to set the record length to ten seconds. The test result recordings usually lasts longer than ten seconds to ensure that one full ten second signal is captured in the recording. Recordings are made using the SDR software HDSDR. HDSDR program is also useful in visual and spectral monitoring of the result in real time. HDSDR SDR program in operation is shown in Figure 6.7.

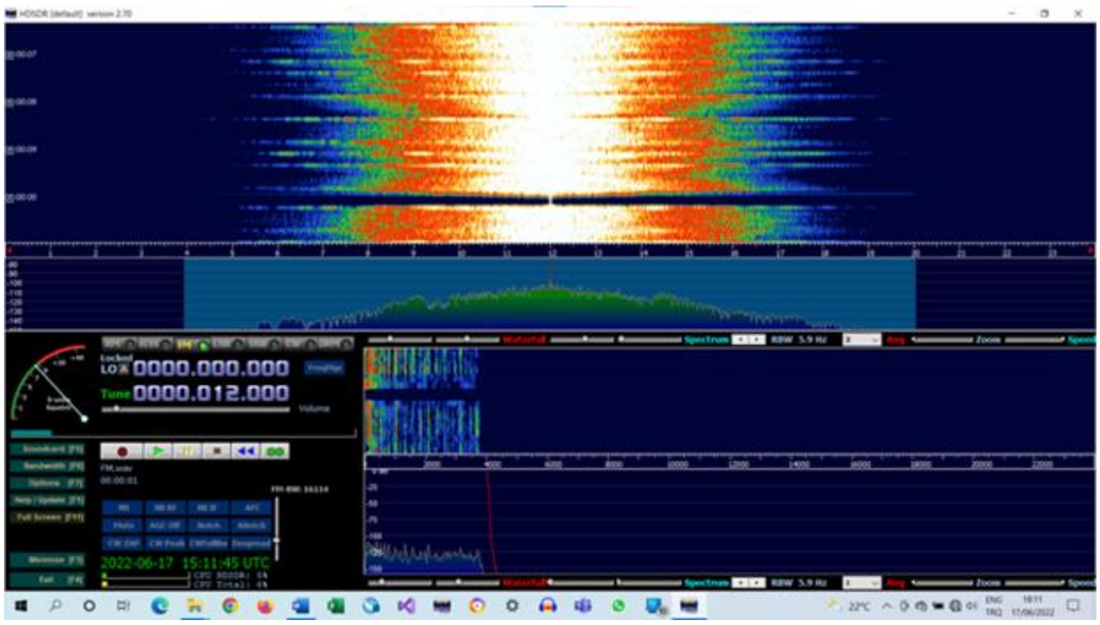


Figure 6.7. HDSDR SDR software.

After analysis code takes this prepared dual channel recording as input, the channels are separated, and then a suitable scaling factor is applied to correct for gain errors. The scaling factor is determined by trial and error. At the conclusion of the code's execution, the Signal-to-Noise ratio (SNR) is calculated and made available; a good value will maximize this ratio. SNR in dB is calculated by dividing rms original signal level to rms error signal level and then this ratio is converted to deciBells. Error is calculated by taking sample by sample difference of original and result signals. The rms level then calculated by squaring and adding each sample and then taking the square root of the average.

6.4. BLOCK SCHEMA AND VHDL CODE OF THE SYSTEM

Software for the FMTX system on FPGA is developed under the XILINX VIVADO integrated development environment (IDE). It is the standard development environment for XILINX Artix-7 series FPGAs. The language used is VHDL which is a standard language for implementation of hardware logic circuits in FPGAs. The code listings for the FMTX system are provided in Appendix B.

The top module code FMTX listed on Appendix B.1 provides a main body for the other functional modules. The internal workings of the FM transmitter are seen in Figure 6.8.

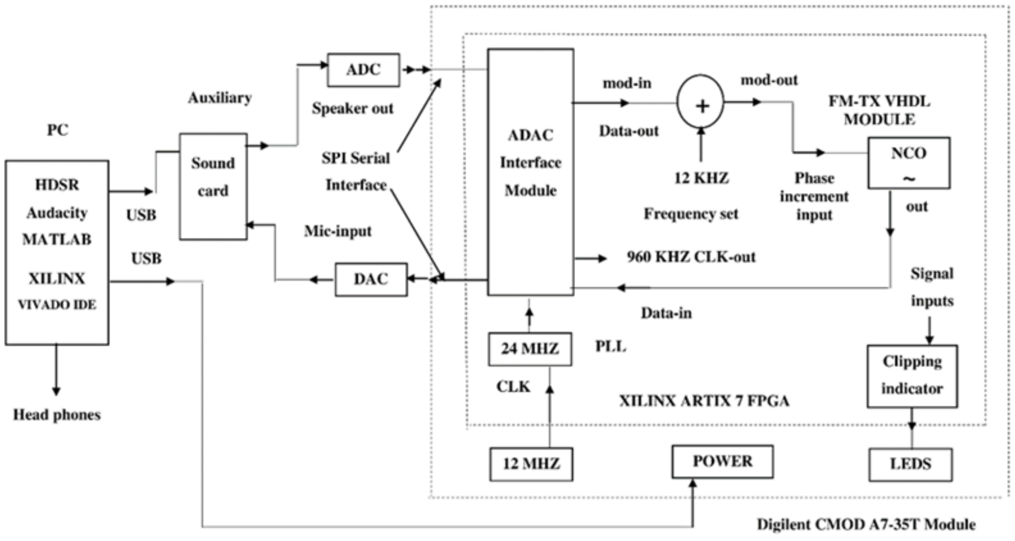


Figure 6.8. Basic block diagram of the FMTX system realized on the FPGA fabric.

PC connection for programming of the module is provided through a spare USB port. This USB port also provides power to the module. Signal flow to the FPGA is analog through an external USB soundcard interface. This external soundcard provides modulating signal through the speaker output and inputs frequency modulated IF signal through microphone input. Both input and output are single channel (mono) ports with a sampling rate of 48KSPs. The input to the FPGA is through the PMOD-AD1 A/D module. The digital output from this module is through a high-speed serial data link in SPI format. This is a synchronous serial interface as can be seen from the datasheet of the ADC chip AD7476. The clock provided by the FPGA is 24MHz which is an integer multiple of sampling rate (960 KHz). So, 12-bit samples are provided at a rate 960 KHz by the ADC. This being an integer multiple of the sampling rate of the soundcard interface at 48 KHz, is a very high rate than the required signal bandwidth of 24 KHz. However, it does not make any harm because the processed signals remain in the 24 KHz bandwidth limit. Such a high sampling rate as 960KSPs is necessary because stable operation of ADC and DAC cards used can only be possible at such high rates.

The PMOD-DA2 D/A module receives serial digital data from the FPGA and transforms it into analog form. The interface is again SPI where the clock rate is 24MHz which must be in conformance to the A/D converter and sampling rate of the FPGA. This clock frequency is derived from an on board crystal clock module running at 12MHz by a Digital Clock Management (DCM) IP module. The other necessary clocks are derived from 24MHz master clock using suitable divider module which ensure synchronicity through all the FPGA fabric.

Handling of the data acquisition to and from DAC and ADC modules is carried over by the ADAC module whose listing is given on Appendix B.2. It provides the data to FM modulator and sends the modulator output to the DAC. It also derives the necessary sampling clock of 960 KHz which is used by the modulator.

Frequency modulator is a simple direct modulator. A Numerically Controlled Oscillator (NCO) is used as a VCO in this case. Phase increment value is controlled by the digital input samples. A higher phase step means an increase in the instantaneous frequency of the NCO. So, changing phase increment input of the NCO causes a frequency modulation at the output. The output of the ADC is first normalized by adjusting the bit length and then factorized and put as a modulator input. Second input to the modulator, which is a simple 32-bit adder, is an offset value which is factorized to give out a frequency offset adjusted to 12-KHz. So, without a modulation input NCO provides a 12-KHz smooth sinusoid. The phase increment input of the NCO is 32-bit while the output is 16-bits, as the code on listing Appendix B.1 reveals. 32-bit is standard to obtain a suitable frequency resolution in NCOs. The NCO is operated at 960KHz, so the input and output sample rates are 960KSps. The frequency output of the NCO, fo is calculated using the equation below:

$$f_o = \frac{f_{clk}.\Delta\theta}{2^{B_{\theta(n)}}} \tag{6.4}$$

And for calculating the phase increment value (Δθ) necessary to generate an output frequency is:

$$\Delta\theta = \frac{2^{B_{\theta(n)}}.f_o}{f_{clk}} \tag{6.5}$$

Here,

fo: Output frequency in Hz,

$B\theta(n)$: Phase increment bit length in number of bits,

$fclk$: Clock speed in Hz,

Δθ: Phase increment value.

So, for 32-bit phase increment register, 960 KHz clock rate and 12-KHz output the phase increment value must be 0x03333333. And for the same parameters the frequency resolution will be 0.0002235174Hz which is a very small value and hence the error in the actual frequency output will be very low.

Another useful feature of the FMTX system implemented in the FPGA is the clipping indicator (clip indicator) whose listing is given on Appendix B.3. For the purpose of checking the output level, this module has two LEDs built right in. If output or input of any module is overloaded (the level crosses a determined threshold) the corresponding led is lit for approximately 1 seconds. One second time delay is necessary to see even a one-time event since human eye cannot follow an event that lasts only 1/960000 of a second. Use of this feature ensures that the analog input of the ADC and digital output of DAC is not overloaded which leads to clipping distortion. Overloading of input of any module in the signal chain can lead to unexpected results and hard to determine faults.

CHAPTER 7

RESULTS AND DISCUSSION

The experiments in the study are obtained in two steps: simulation results and test results. Simulation results come from ideal simulation efforts using MATLAB codes listed on Appendix A. Test results come from as recordings from actual operational tests of FMTX system implemented on FPGA.

Two modulation test signals used in each stage are A1 and A2. Each test signal lasts 10 seconds and sampled at a rate 48KSps. Results from simulation and tests, which are demodulated audio, are recorded as a separate wave file. These results are also 48KSps wave files. A post-processing is applied to these results under audio processing program Audacity. These post-processes are mainly normalization and synchronization processes which after demodulation results and original modulating signals are combined into a single stereo (2-channel) recording which lasts exactly 10 seconds. These recordings hold demodulation product on left channel (upper signal in the stereo track), while original is held on right channel (bottom signal in the stereo track). Figure 7.1 shows the result for the simulation A1, while Figure 7.2 shows the result for the simulation A2. Results for the test A1 and A2 are provided on Figure 7.3 and Figure 7.4 respectively.

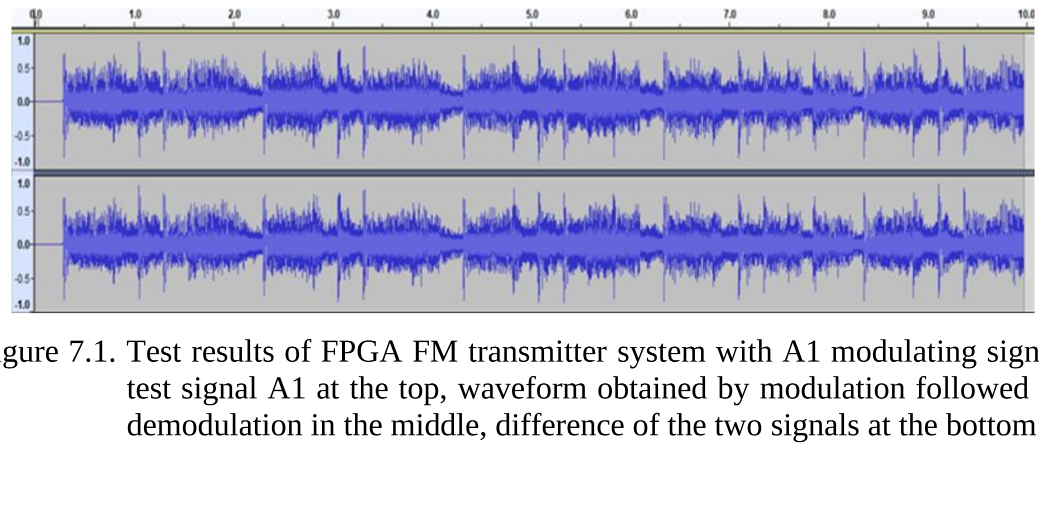


Figure 7.1. Test results of FPGA FM transmitter system with A1 modulating signal: test signal A1 at the top, waveform obtained by modulation followed by demodulation in the middle, difference of the two signals at the bottom.

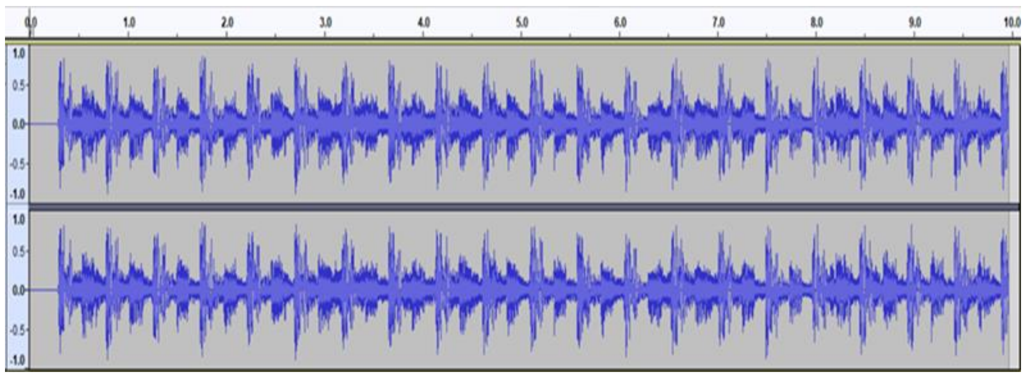


Figure 7.2. Simulation results of FPGA FM transmitter system with A1 modulating signal: test signal A1 at the top, waveform obtained by modulation followed by demodulation in the middle, difference of the two signals at the bottom.

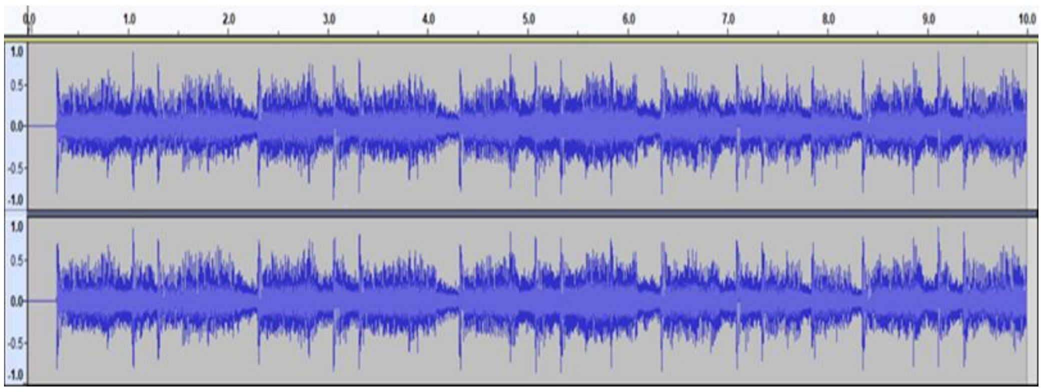


Figure 7.3. Test results of the FPGA FM transmitter system with the A2 modulating signal: test signal A2 at the top, the waveform obtained by modulation followed by demodulation in the middle, the difference of the two signals at the bottom.

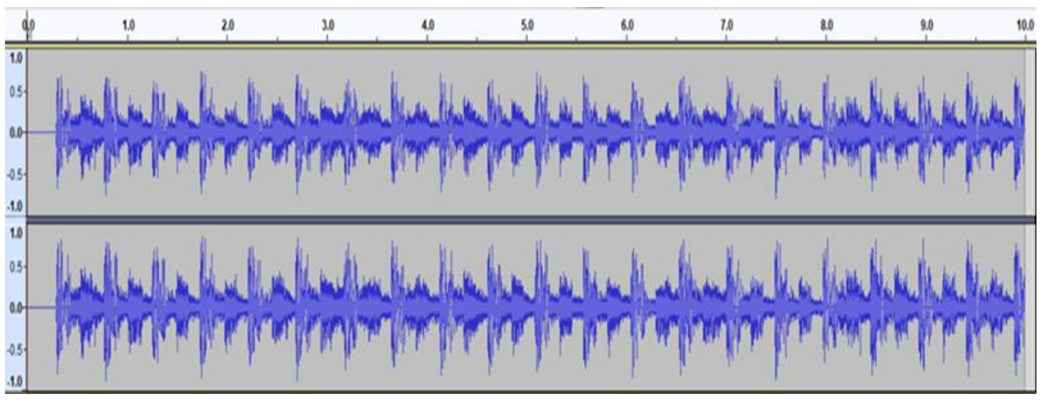


Figure 7.4. Simulation results of the FPGA FM transmitter system with the A2 modulating signal: test signal A2 at the top, waveform obtained by modulation followed by demodulation in the middle, difference of the two signals at the bottom.

These results are analysed using the code listed on Appendix A.3. This analysis code compares two signals by subtracting from each other. From this difference (error) signal rms error is calculated. Then S/N ratio is calculated in dB using rms error and rms original signal level whose definition is given on Equation 1.

$$S/N_{dB} = 20\log\left(\frac{S_{rms}}{E_{rms}}\right) \tag{7.1}$$

Signal-to-Noise ratio and rms error results for each simulation and test is given on Table 7.1.

Table 7.1. SNR and rms error for each simulation and test.

Test or simulation	**rms error (x10^{-3})**	**SNR (dB)**
A1 simulation	1.048	44.18
A1 test	15.597	20.73
A2 simulation	1.181	43.09
A2 test	17.144	19.86

For the comparison of results, higher SNR and lower rms error level is better. So, when we compare the performances, the best results are obtained from the simulations with A1 test signal. Simulating with test signal A2 produced somewhat less favorable results. The worst result belongs to experiments with test signal A2. It is slightly below the 20dB acceptable performance threshold, while the experiments with test signal A1 provided slightly higher performance above the 20dB threshold. The results prove the FMTX system as a good candidate for generating quality FM signals for broadcast purposes.

CHAPTER 8

CONCLUSION

In this study, an FM transmitter (FMTX) system utilizes SDR methods in its design and implementation on an FPGA platform. The aim to primarily a platform for education of practical SDR systems. The PC is used as a data administration and control central. The obtained results are evaluated using MATLAB scripts. Suitable ADC and DAC modules are used to process analog signals on the FPGA side and an external soundcard is used for the same purposes on the PC side. The collected signals are processed by the FPGA fabric. A signal chain is developed using VHDL hardware description language under XILINX VIVADO IDE for this purpose. The FMTX system generates a 12 KHz IF signal as a frequency modulated signal. The frequency modulator depends on a wideband direct modulation method which is a very simple technique to encode frequency modulated signals. An NCO IP is used to provide this facility. Minimal use of filters is preferred in the design of the system to keep it as simple as possible. That the system's price, complexity, and energy usage be kept to a minimum.

The study consists of two stages: first one is the simulation and design of the FMTX system and the second stage is the implementation on FPGA and verification of the actual results. The first stage is accomplished through suitable MATLAB codes which simulates a FM transmitter using a 10 second music recording. Two different samples, A1 and A2 are derived using different audio recordings in wav format. A demodulator code provides an ideal demodulation result used for comparison purposes in the design verification stage. So, a suitable MATLAB code is also used to analysis the experiment and simulation results.

Design of the FMTX system is made on XILINX VIVADO IDE. The codes are written using VHDL. The design is based on the FM transmitter code written under MATLAB.

The modulating signals are continuously played back using Audacity through the PC soundcard speaker output and modulation signal is recorded through microphone input of the same soundcard interface using HDSDR SDR software. HDSDR provides both a visual means for monitoring the results in frequency domain using its waterfall and spectrogram displays and recording them in the hard disk while listening the demodulation products through another spare soundcard output. The recorded results from the experiments are then analysed and compared to the results from the simulation stages. The analysis results include rms error level which shows the level of noise from the original and calculation of SNR in dB from the rms error and original rms signal level.

Analysis of the results show that FMTX system implemented on FPGA is successful in the generation of FM signals. So, it can be accepted as a good candidate in the training and studying of the practical SDR principles of FM signals. The designed FMTX system can be utilized as a practical FM backend for a SDR equipment such as Softrock Ensemble TX which can be set to accept a 12KHz IF. Thus, FM communication can be possible on Civilization Band (CB) which is located at 27MHz. The FMTX system can be integrated with a suitable FM receiver built on the same FPGA and can be used as a transceiver on HF or VHF band if combined with a suitable up-down converter which can operate with a 12-KHz IF.

REFERENCES

1. Mady Z.G.A. (2016). Transmit and Receive of Quadrature Phase-Shift Keying (QPSK) Signal Using Softrock SDR and Matlab, ***Natural and Appliance Science of Karabuk University,*** Turkey.

2. Gareane A.G.A (2016). Transmit and Receive of FM Signals Using Softrock SDR and Matlab, ***Natural and Appliance Science of Karabuk University,*** Turkey.

3. Eame M.A.M (2016). Transmit and Receive of FSK Signals Using Softrock SDR and Matlab***, Natural and Appliance Science of Karabuk University***, Turkey.

4. Feng Z. (2013). A Software Defined Radio Implementation Using Matlab, ***Vaasan Ammattikorkeakoulu University of Applied Sciences,*** Finland.

5. N.H. Sephus, A.D. Lanterman and D.V. Anderson, "Exploring frequency modulation features and resolution in the modulation spectrum," in Digital Signal Processing and Signal Processing Education Meeting (DSP/SPE), 2013 IEEE , **vol., no., pp.169-174, 11-14 Aug. 2013.**

6. D.V I. Vitas, D. Šimuniand P. Kneževi, "Evaluation of software defined radio systems for smart home Environments," ***in MIPRO, Opatija***, Croatia, 2015, p 562.

7. I. Hatai and I. Chakrabarti "A new high performance digital FM modulator and demodulator for software defined radio and its FPGA implementation," ***International Journal of Reconfigurable Computing,*** New York, United States, 2011.

8. I. Hatai and I. Chakrabarti "FPGA Implemantation of a Digital FM Modem," ***International Conference on Information and Multimedia Technology***, 2009, pp 1-4

9. D. V. W. L. and f. h., "How to pack a room of analog FM-modulators into a Xilinx FPGA," ***Xilinx DSP Magazine***, April 2007.

10. Ali Ibrahim Khalifa HANDER, 2021 "Design and Implementation of an Educational AM Receiver With FPGA Using SDR Techniques", ***Karabük University, Department of Electrical and Electronic Engineering.***

11. C. Kiremitci and B. Erkal, "Eğitim Amaçlı SDR Tekniklerine Dayalı FPGA Tabanlı Genlik Modüleli Radyo Vericisi Tasarımı ve Uygulaması", Avrupa Bilim ve Teknoloji Dergisi, **pp. 184-189, Oct. 2020, doi:10.31590/ejosat.803492.**

12. Hikmat N. Abdullah., “SOFTWARE DEFINED RADIO USING SIMULINK HDL CODER”, ***University of Al-Mustansiryah, College of Engineering, Electrical-Engineering Department,*** Baghdad-Iraq, (2020).

13. Collins T.F., Getz R., Pu D. and Wyglinski A.M. “Software Defined Radio for Engineers”, ***John Gomes,*** US (2018).

14. Akpolat A.N. “FPGA Tabanlı Nesne Algılama”, Yüksek Lisans Tezi, Fırat Üniversitesi Fen Bilimleri Enstitüsü, Şanlıurfa (2015)

15. Wang Lin He Jun Li, F. and Yuliang, L., "Ultra low frequency phase generated carrier demodulation technique for fiber sensors [J]", Chinese Journal of Lasers, 4: 025 (2011).

16. Yılmaz N. “Designing an ANN on Field Programmable Gate Arrays (FPGA) and Implementation as Hardware”, M.Sc., ***Selçuk University Institute of Science and Technology,*** Şanlıurfa (2008).

17. Kibar A.E. “Harmonic Analysis of Power Signals Using Amplitude Modulation”, Master Thesis, ***Gazi University Institute of Science and Technology,*** Ankara (2019).

18. Alghiryani S.G.S . “Transmit and Receive of SSB and DSB-AM Signals Using Softrock SDR and Matlab”, ***Yüksek Lisans Tezi, Karabük Üniversitesi Fen Bilimleri Enstitüsü,*** Karabük (2016).

19. Dikmen O., “Investigation of Amplitude Modulation-2”, Düzce University ***Department of Electrical and Electronics Engineering,*** Düzce (2016).

20. Internet: Michigan Technological University, “Am Transmitter-Prelab”, **http://www.ece.mtu.edu/labs/EElabs/EE3305/AM_transmitter pdf**

21. Internet: Amateur Radio Station N0GSG, “Chapter 4: AM Transmitters”, **http://n0gsg.com/ecfp/ch4_sample.pdf**

22. Mehta, V. K. and Mehta, R., “Chapter 16: Modulation and Demodulation, Principles of Electronics”, S. Chand & Company Ram Nagar, New Delhi (2014).

23. Internet: Digilent, “CMOD-A7 FPGA board”, **https://digilent.com/reference/_media/cmod_a7/cmod_a7_rm.pdf**

24. Internet: Digilent, “PMOD-AD1”, **https://digilent.com/reference/_media/pmod:pmod:pmodAD1_rm.pdf**

25. Internet: Digilent, “PMOD-DA2”, **https://reference.digilentinc.com/reference/pmod/pmodda2/reference-manual**

26. Internet: Audacity Audio Software Systems: **https://www.audacityteam.org/**

27. Internet: High Definition Software Defined Radio: **https://hdsdr.software.informer.com/**

28. Internet: Mathworks Systems (MATLAB): **https://www.mathworks.com/products/matlab.html/**

APPENDIX A.

APPENDIX A. MATLAB CODE LISTINGS

Appendix A.1 Transmitter (Modulation) code (fmtx.m)

```
% (WBFM) Mod. with MUSIC by B. ERKAL 2021
% FM transmitter simulation code by Bilgehan ERKAL
% Karabuk 2022
clear all;

% sound file 1 loading (4Khz mono (8KSps))
[iff1 , afs]=audioread('a1.wav');
[y1,~]=size(iff1);
% upsample x6 (8x6=48Khz)
yu1=upsample(iff1,6);
% Baseband signal is filtered and normalized
yu1=filter(fir1(128,4e3/24e3),1,yu1);
yu1=yu1./(1.01*max(abs(yu1)));
audiowrite('a1_48k.wav', yu1, 48e3);

fs=48e+3;       % sampling frequency
ts=1/fs;        % sampling interval
t=0:ts:10-ts;   % time axis

% FM modulation
% message signal integral
ati=0;
% Deltafmax = Amax / (2*pi*ts) = 0.524 / 1.31*10^-4 = 4KHz
yu1=0.524*yu1;
[y,~]=size(yu1);
at(1:y,1)=0;
for i=1:1:y
    ati=ati+yu1(i,1);
    at(i,1)=ati;
end

% (BTFM=2*(Deltafmax+BWm)= 2*(4K+4K)=16KHz and Beta=deltafmax/BWm=1)

% carrier parameters
C=22938;fct=12e+3;tetac=0*(pi/180);
% FM IF signal
m=int16(C*cos((2*pi*fct*t'+tetac+at)));
m=int16(filter(fir1(2048,[(fct-8e3)/24e3 (fct+8e3)/24e+3],
'bandpass'),1,m));

% IF signal is recorded in wav file
audiowrite('FM.wav', m, fs);
```

Appendix A.2 Receiver (Demodulation) code (fmrx.m)

```
% (WBFM) Demod. with MUSIC by B. ERKAL 2021
% FM receiver code by Bilgehan ERKAL
% Karabuk 2022
clear all;

% IF file loading (48Khz stereo)
[yu1 , afs]=audioread('FM.wav');
[y1,~]=size(yu1);

% IF signal is normalized
yu1=yu1./(1.01*max(abs(yu1)));

fs=afs;          % sampling frequency
ts=1/fs;         % sampling interval
t=0:ts:10-ts;    % time axis

% Quadrature FM Demodulation
dem=yu1(1:end-1,1).*yu1(2:end,1);
dem(end+1,1)=dem(end,1);

% Final filtering (lowpass fc=4KHz)
dem=filter(fir1(256,4e3/24e3),1,dem);

% Demodulated signal normalized and written to file
dem=dem./(1.1*max(abs(dem)));
audiowrite('DEM.wav', dem, fs);
```

Appendix A.3 Signal Analysis code (an.m)

```
% FM Demod. performance analysis
% FM receiver analysis by Bilgehan ERKAL
% Karabuk 2022
clear all;

% stereo comparison file loading (48Khz stereo)
[iff1 , afs]=audioread('st_Ldem_Ra1.wav');
[y1,~]=size(iff1);
% channel seperation and gain error correction
rec=0.9566*iff1(1:y1,1)';
a1=1*iff1(1:y1,2)';

% Calculate rms error and rms signal
diff=(a1-rec)/2;
err=(mean(diff.^2))^0.5;
a1_rms=(mean(a1.^2))^0.5;
fprintf('rms error: %d \nSNR(dB): %d \n', err,
20*log10(a1_rms/err));

audiowrite('diff.wav', diff, afs);
```

APPENDIX B.

APPENDIX B. VHDL CODE LISTINGS

Appendix B.1 (FMTX.VHD – top module)

```
--------------------------------------------------------------------
--------------
-- Company: KARABUK UNIVERSITY
-- Engineer: Bilgehan ERKAL
--
-- Create Date: 19.05.2022 16:23:24
-- Design Name: FMTX FM XMITTER
-- Module Name: fmtx - Behavioral

--------------------------------------------------------------------
--------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

entity fmtx is
    Port (
    led : out std_logic_vector(1 downto 0); -- LED Indicators

    -- adc module connections
    sync_n1 : out std_logic;    -- chip select          1 - 43
    sdi0 : in std_logic;        -- serial data out 0    2 - 44
    sdi1 : in std_logic;        -- serial data out 1    3 - 45
    sclk1 : out std_logic;      -- serial clock         4 - 46
    gnd1  : out std_logic;      -- negative supply      5 - 47
    vcc1  : out std_logic;      -- positive supply      6 - 48

    -- dac module connections
    sync_n2 : out std_logic;    -- chip select          1 - 6
    sdo0 : out std_logic;       -- serial data in 0     2 - 5
    sdo1 : out std_logic;       -- serial data in 1     3 - 4
    sclk2 : out std_logic;      -- serial clock         4 - 3
    gnd2  : out std_logic;      -- negative supply      5 - 2
    vcc2  : out std_logic;      -- positive supply      6 - 1

    clk : in std_logic          -- master clock 12MHz
    );
end fmtx;

architecture Behavioral of fmtx is
component clk_wiz_1
port
 (-- Clock in ports
  -- Clock out ports
  clk_out1          : out    std_logic;
  clk_out2          : out    std_logic;
  -- Status and control signals
  reset             : in     std_logic;
  locked            : out    std_logic;
  clk_in1           : in     std_logic
 );
end component;

COMPONENT clip_indicator
```

```
  PORT (
       clip_in          : in std_logic_vector(11 downto 0);
       clip_out         : out STD_LOGIC;
       clk              : in  STD_LOGIC --960KHZ

  );
END COMPONENT;

COMPONENT c_addsub_0
  PORT (
    A   : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    B   : IN STD_LOGIC_VECTOR(31 DOWNTO 0);
    CLK : IN STD_LOGIC;
    S   : OUT STD_LOGIC_VECTOR(31 DOWNTO 0)
  );
END COMPONENT;

COMPONENT dds_compiler_0
  PORT (
    aclk                : IN STD_LOGIC;
    s_axis_phase_tvalid : IN STD_LOGIC;
    s_axis_phase_tdata  : IN STD_LOGIC_VECTOR(63 DOWNTO 0);
    m_axis_data_tvalid  : OUT STD_LOGIC;
    m_axis_data_tdata   : OUT STD_LOGIC_VECTOR(15 DOWNTO 0)
  );
END COMPONENT;

component adac
Port (     clk_24MHz    : in STD_LOGIC;
           reset_n      : in STD_LOGIC;
            -- adc module connections
            sync_n1     : out std_logic;-- chip select          1 -
43
            sdi0        : in std_logic;-- serial data out 0     2 -
44
            sdi1        : in std_logic;-- serial data out 1     3 -
45
            -- dac module connections
            sync_n2     : out std_logic;-- chip select          1 -
6
            sdo0        : out std_logic;-- serial data in 0     2 -
5
            sdo1        : out std_logic;-- serial data in 1     3 -
4

            -- dac signals
            dac_input1  : in std_logic_vector(15 downto 0);
            dac_input2  : in std_logic_vector(15 downto 0);

            -- adc signals
            adc_reg1    : out std_logic_vector(15 downto 0);
            adc_reg2    : out std_logic_vector(15 downto 0);

           clk_960KHz_out : out STD_LOGIC -- 960KHz sampler clock
output
           );
end component;

-- Reset and clock signals
```

```
signal res_count    : std_logic_vector(26 downto 0) := (others =>
'0');
signal reset_n      : std_logic := '1';
signal clk_48MHz    : std_logic;
signal clk_960KHz   : std_logic;
signal clk_24MHz    : std_logic := '0';

-- DDS and modulator signals
signal mod_in   : std_logic_vector(31 downto 0) := (others => '0');
signal s        : std_logic_vector(6 downto 0) := (others => '0');
signal mod_out  : std_logic_vector(31 downto 0) := (others => '0');
signal freq     : std_logic_vector(31 downto 0) := (others => '0');
signal phi      : std_logic_vector(63 downto 0) := (others => '0');
signal dds_out  : std_logic_vector(15 downto 0) := (others => '0');

-- Dual channel ADC-DAC output/ input signals (ADAC MODULE)
signal adc_out1     : std_logic_vector(15 downto 0) := (others =>
'0');
signal dac_input1   : std_logic_vector(15 downto 0) := (others =>
'0');
signal adc_out2     : std_logic_vector(15 downto 0) := (others =>
'0');
signal dac_input2   : std_logic_vector(15 downto 0) := (others =>
'0');

-- 2-channel clipping indicator signals
signal clip_in1     : std_logic_vector(11 downto 0) := (others =>
'0');
signal clip_in2     : std_logic_vector(11 downto 0) := (others =>
'0');

begin

-- module connectors
freq <= X"03333333"; -- 12KHz
s <= (others => not adc_out1(11)); -- sign of adc output
mod_in <= s & adc_out1(6 downto 0) & "000000000000000000";--
modulator input from ADC
phi <= X"00000000" & mod_out; -- modulation input of DDS (phase
increment input)

dac_input1 <= "0000" & not(dds_out(15)) & dds_out(11 downto 1);-- FM
signal output from DDS
--dac_input1 <= "0000" & adc_out1(11) & adc_out1(6 downto 0) &
"0000";--loopback data
--dac_input1 <= X"0FFF";--fixed data
dac_input2 <= X"0000";--fixed data
--dac_input2 <= adc_out2;--loopback data

-- clock and power supply connections
sclk1 <= clk_24MHz;
sclk2 <= clk_24MHz;
gnd1 <= '0';
gnd2 <= '0';
vcc1 <= '1';
vcc2 <= '1';

-- clipping indicators
--clip_in1 <= X"59B";-- 1435 threshold value
clip_in1 <= not adc_out1(11) & adc_out1(10 downto 0);
```

```
clip_in2 <= not dac_input1(11) & dac_input1(10 downto 0);
--clip_in2 <= x"A65";--  -1435 negative threshold

inst_c_ind1 : clip_indicator
    port map (
    clip_in => clip_in1,
    clip_out => led(0),
    clk => clk_960KHz

    );

inst_c_ind2 : clip_indicator
    port map (
    clip_in => clip_in2,
    clip_out => led(1),
    clk => clk_960KHz

    );

-- Clock module
pll1 : clk_wiz_1
   port map (
  -- Clock out ports
   clk_out1 => clk_48MHz,
   clk_out2 => clk_24MHz,
  -- Status and control signals
   reset => '0',
   locked => open,
   -- Clock in ports
   clk_in1 => clk -- 12MHz master clock
 );

-- Master reset of FPGA fabric
      reset_proc: process(clk_24MHz)
            begin
                  if rising_edge(clk_24MHz) then
                        if res_count(26) = '1' then
                              res_count <= res_count;
                        else
                              res_count <= res_count + 1;
                        end if;
                  end if;
            end process;

      reset_n <= res_count(26);

-- Frequency modulator
fm_modulator : c_addsub_0
  PORT MAP (
    A => freq,          -- frequency input
    B => mod_in,        -- modulator input
    CLK => clk_960KHz,  -- sampling clock input (960KHz)
    S => mod_out        -- modulator output
  );

-- DDS (Direct Digital Synthesis) or NCO (Numerically Controlled
Oscillator) module
dds1 : dds_compiler_0
  PORT MAP (
    aclk => clk_960KHz,
    s_axis_phase_tvalid => '1',
    s_axis_phase_tdata => phi,
    m_axis_data_tvalid => open,
    m_axis_data_tdata => dds_out
  );

-- ADAC mdoule
inst_adac : adac
    port map (
    clk_24MHz => clk_24MHz,
    reset_n => reset_n,
    -- adc module connections
     sync_n1  => sync_n1,
     sdi0 => sdi0,
     sdi1 => sdi1,
     -- dac module connections
     sync_n2 => sync_n2,
     sdo0  => sdo0,
     sdo1 => sdo1,

     -- dac signals
     dac_input1 => dac_input1,
     dac_input2 => dac_input2,

     -- adc signals
     adc_reg1 => adc_out1,
     adc_reg2 => adc_out2,

     clk_960KHz_out => clk_960KHz
           );

end Behavioral;
```

Appendix B.2 (ADAC.VHD)

```
--------------------------------------------------------------------
--------------
-- Company: KARABUK UNIVERSITY
-- Engineer: BILGEHAN ERKAL
--
-- Create Date: 09.10.2021 12:39:05
-- Design Name: High speed version (960KSps)
-- Module Name: adac - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
--------------------------------------------------------------------
--------------


library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx leaf cells in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity adac is
    Port ( clk_24MHz    : in STD_LOGIC;
           reset_n      : in STD_LOGIC;
            -- adc module connections
            sync_n1     : out std_logic;-- chip select          1 -
43
            sdi0        : in std_logic;-- serial data out 0     2 -
44
            sdi1        : in std_logic;-- serial data out 1     3 -
45
            -- dac module connections
            sync_n2     : out std_logic;-- chip select          1 -
6
            sdo0        : out std_logic;-- serial data in 0     2 -
5
            sdo1        : out std_logic;-- serial data in 1     3 -
4

            -- dac signals
            dac_input1 : in std_logic_vector(15 downto 0);
            dac_input2 : in std_logic_vector(15 downto 0);

            -- adc signals
            adc_reg1 : out std_logic_vector(15 downto 0);
            adc_reg2 : out std_logic_vector(15 downto 0);

           clk_960KHz_out : out STD_LOGIC);
end adac;

architecture Behavioral of adac is

-- ADAC process vars
signal adac_state : std_logic_vector(7 downto 0) := (others => '0');
signal cs_n : std_logic;
signal clk_960KHz : std_logic := '0';
signal sd_reg1 : std_logic_vector(15 downto 0) := (others => '0');
signal adc_out1 : std_logic_vector(15 downto 0)  := (others => '0');
signal dac_in1 : std_logic_vector(15 downto 0) := (others => '0');

signal sd_reg2 : std_logic_vector(15 downto 0) := (others => '0');
signal adc_out2 : std_logic_vector(15 downto 0)  := (others => '0');
signal dac_in2 : std_logic_vector(15 downto 0) := (others => '0');

begin

-- module connectors
sync_n1 <= cs_n;
sync_n2 <= cs_n;
sdo0 <= dac_in1(15);
sdo1 <= dac_in2(15);
clk_960KHz_out <= clk_960KHz;
adc_reg1 <= adc_out1;
adc_reg2 <= adc_out2;

-- ADAC process
adac_proc: process(clk_24MHz)
      begin
            if rising_edge(clk_24MHz) then
              if reset_n = '1' then
                  CASE adac_state(7 downto 0) IS
                        WHEN X"00" => --
                              adac_state <= adac_state + 1;
                              cs_n <= '0';
                              sd_reg1 <= (others => '0');
                              adc_out1 <= adc_out1;
                              dac_in1 <= dac_in1;
                              sd_reg2 <= (others => '0');
                              adc_out2 <= adc_out2;
                              dac_in2 <= dac_in2;
                              clk_960KHz <= '0';

                        WHEN X"01" => --
                              adac_state <= adac_state + 1;
                              cs_n <= '0';
                              sd_reg1 <= sd_reg1(14 downto 0) &
sdi0;
                              adc_out1 <= adc_out1;
                              dac_in1 <= dac_in1(14 downto 0) & '0';
                              sd_reg2 <= sd_reg2(14 downto 0) &
sdi1;
```

```
                              adc_out2 <= adc_out2;
                              dac_in2 <= dac_in2(14 downto 0) & '0';
                              clk_960KHz <= '0';

                        WHEN X"02" => --
                              adac_state <= adac_state + 1;
                              cs_n <= '0';
                              sd_reg1 <= sd_reg1(14 downto 0) &
sdi0;
                              adc_out1 <= adc_out1;
                              dac_in1 <= dac_in1(14 downto 0) & '0';
                              sd_reg2 <= sd_reg2(14 downto 0) &
sdi1;
                              adc_out2 <= adc_out2;
                              dac_in2 <= dac_in2(14 downto 0) & '0';
                              clk_960KHz <= '0';

                        WHEN X"03" => --
                              adac_state <= adac_state + 1;
                              cs_n <= '0';
                              sd_reg1 <= sd_reg1(14 downto 0) &
sdi0;
                              adc_out1 <= adc_out1;
                              dac_in1 <= dac_in1(14 downto 0) & '0';
                              sd_reg2 <= sd_reg2(14 downto 0) &
sdi1;
                              adc_out2 <= adc_out2;
                              dac_in2 <= dac_in2(14 downto 0) & '0';
                              clk_960KHz <= '0';

                        WHEN X"04" => --
                              adac_state <= adac_state + 1;
                              cs_n <= '0';
                              sd_reg1 <= sd_reg1(14 downto 0) &
sdi0;
                              adc_out1 <= adc_out1;
                              dac_in1 <= dac_in1(14 downto 0) & '0';
                              sd_reg2 <= sd_reg2(14 downto 0) &
sdi1;
                              adc_out2 <= adc_out2;
                              dac_in2 <= dac_in2(14 downto 0) & '0';
                              clk_960KHz <= '0';

                        WHEN X"05" => --
                              adac_state <= adac_state + 1;
                              cs_n <= '0';
                              sd_reg1 <= sd_reg1(14 downto 0) &
sdi0;
                              adc_out1 <= adc_out1;
                              dac_in1 <= dac_in1(14 downto 0) & '0';
                              sd_reg2 <= sd_reg2(14 downto 0) &
sdi1;
                              adc_out2 <= adc_out2;
                              dac_in2 <= dac_in2(14 downto 0) & '0';
                              clk_960KHz <= '0';

                        WHEN X"06" => --
                              adac_state <= adac_state + 1;
                              cs_n <= '0';
                              sd_reg1 <= sd_reg1(14 downto 0) &
sdi0;
                              adc_out1 <= adc_out1;
                              dac_in1 <= dac_in1(14 downto 0) & '0';
                              sd_reg2 <= sd_reg2(14 downto 0) &
sdi1;
                              adc_out2 <= adc_out2;
                              dac_in2 <= dac_in2(14 downto 0) & '0';
                              clk_960KHz <= '0';

                        WHEN X"07" => --
                              adac_state <= adac_state + 1;
                              cs_n <= '0';
                              sd_reg1 <= sd_reg1(14 downto 0) &
sdi0;
                              adc_out1 <= adc_out1;
                              dac_in1 <= dac_in1(14 downto 0) & '0';
                              sd_reg2 <= sd_reg2(14 downto 0) &
sdi1;
                              adc_out2 <= adc_out2;
                              dac_in2 <= dac_in2(14 downto 0) & '0';
                              clk_960KHz <= '0';

                        WHEN X"08" => --
                              adac_state <= adac_state + 1;
                              cs_n <= '0';
                              sd_reg1 <= sd_reg1(14 downto 0) &
sdi0;
                              adc_out1 <= adc_out1;
                              dac_in1 <= dac_in1(14 downto 0) & '0';
                              sd_reg2 <= sd_reg2(14 downto 0) &
sdi1;
                              adc_out2 <= adc_out2;
                              dac_in2 <= dac_in2(14 downto 0) & '0';
                              clk_960KHz <= '0';

                        WHEN X"09" => --
                              adac_state <= adac_state + 1;
                              cs_n <= '0';
                              sd_reg1 <= sd_reg1(14 downto 0) &
sdi0;
                              adc_out1 <= adc_out1;
                              dac_in1 <= dac_in1(14 downto 0) & '0';
                              sd_reg2 <= sd_reg2(14 downto 0) &
sdi1;
                              adc_out2 <= adc_out2;
                              dac_in2 <= dac_in2(14 downto 0) & '0';
                              clk_960KHz <= '0';

                        WHEN X"0A" => --
                              adac_state <= adac_state + 1;
                              cs_n <= '0';
                              sd_reg1 <= sd_reg1(14 downto 0) &
sdi0;
                              adc_out1 <= adc_out1;
                              dac_in1 <= dac_in1(14 downto 0) & '0';
                              sd_reg2 <= sd_reg2(14 downto 0) &
sdi1;
                              adc_out2 <= adc_out2;
                              dac_in2 <= dac_in2(14 downto 0) & '0';
                              clk_960KHz <= '0';

                        WHEN X"0B" => --
                              adac_state <= adac_state + 1;
                              cs_n <= '0';
                              sd_reg1 <= sd_reg1(14 downto 0) &
sdi0;
                              adc_out1 <= adc_out1;
                              dac_in1 <= dac_in1(14 downto 0) & '0';
                              sd_reg2 <= sd_reg2(14 downto 0) &
sdi1;
                              adc_out2 <= adc_out2;
                              dac_in2 <= dac_in2(14 downto 0) & '0';
                              clk_960KHz <= '0';

                        WHEN X"0C" => --
                              adac_state <= adac_state + 1;
                              cs_n <= '0';
                              sd_reg1 <= sd_reg1(14 downto 0) &
sdi0;
                              adc_out1 <= adc_out1;
                              dac_in1 <= dac_in1(14 downto 0) & '0';
                              sd_reg2 <= sd_reg2(14 downto 0) &
sdi1;
                              adc_out2 <= adc_out2;
                              dac_in2 <= dac_in2(14 downto 0) & '0';
                              clk_960KHz <= '0';

                        WHEN X"0D" => --
                              adac_state <= adac_state + 1;
                              cs_n <= '0';
                              sd_reg1 <= sd_reg1(14 downto 0) &
sdi0;
                              adc_out1 <= adc_out1;
                              dac_in1 <= dac_in1(14 downto 0) & '0';
                              sd_reg2 <= sd_reg2(14 downto 0) &
sdi1;
                              adc_out2 <= adc_out2;
                              dac_in2 <= dac_in2(14 downto 0) & '0';
                              clk_960KHz <= '0';

                        WHEN X"0E" => --
                              adac_state <= adac_state + 1;
                              cs_n <= '0';
                              sd_reg1 <= sd_reg1(14 downto 0) &
sdi0;
                              adc_out1 <= adc_out1;
                              dac_in1 <= dac_in1(14 downto 0) & '0';
                              sd_reg2 <= sd_reg2(14 downto 0) &
sdi1;
                              adc_out2 <= adc_out2;
                              dac_in2 <= dac_in2(14 downto 0) & '0';
                              clk_960KHz <= '0';

                        WHEN X"0F" => --
                              adac_state <= adac_state + 1;
                              cs_n <= '0';
                              sd_reg1 <= sd_reg1(14 downto 0) &
sdi0;
                              adc_out1 <= adc_out1;
                              dac_in1 <= dac_in1(14 downto 0) & '0';
                              sd_reg2 <= sd_reg2(14 downto 0) &
sdi1;
                              adc_out2 <= adc_out2;
                              dac_in2 <= dac_in2(14 downto 0) & '0';
                              clk_960KHz <= '0';

                        WHEN X"10" => --
                              adac_state <= adac_state + 1;
                              cs_n <= '1';
                              sd_reg1 <= sd_reg1(14 downto 0) &
sdi0;
                              adc_out1 <= adc_out1;
                              dac_in1 <= dac_in1;
                              sd_reg2 <= sd_reg2(14 downto 0) &
sdi1;
                              adc_out2 <= adc_out2;
                              dac_in2 <= dac_in2;
                              clk_960KHz <= '0';

                        WHEN X"11" => --
                              adac_state <= adac_state + 1;
                              cs_n <= '1';
                              sd_reg1 <= sd_reg1;
                              adc_out1 <= sd_reg1;
                              dac_in1 <= dac_in1;
                              sd_reg2 <= sd_reg2;
                              adc_out2 <= sd_reg2;
                              dac_in2 <= dac_in2;
                              clk_960KHz <= '0';

                        WHEN X"12" => --
                              adac_state <= adac_state + 1;
                              cs_n <= '1';
                              sd_reg1 <= sd_reg1;
                              adc_out1 <= adc_out1;
                              dac_in1 <= dac_input1;
                              sd_reg2 <= sd_reg2;
                              adc_out2 <= adc_out2;
                              dac_in2 <= dac_input2;
                              clk_960KHz <= '0';

                        WHEN X"13" => --
                              adac_state <= adac_state + 1;
                              cs_n <= '1';
                              sd_reg1 <= sd_reg1;
                              adc_out1 <= adc_out1;
                              dac_in1 <= dac_in1;
                              sd_reg2 <= sd_reg2;
                              adc_out2 <= adc_out2;
                              dac_in2 <= dac_in2;
                              clk_960KHz <= '0';

                        WHEN X"14" => --
                              adac_state <= adac_state + 1;
                              cs_n <= '1';
                              sd_reg1 <= sd_reg1;
                              adc_out1 <= adc_out1;
                              dac_in1 <= dac_in1;
                              sd_reg2 <= sd_reg2;
                              adc_out2 <= adc_out2;
                              dac_in2 <= dac_in2;
                              clk_960KHz <= '0';

                        WHEN X"15" => --
                              adac_state <= adac_state + 1;
                              cs_n <= '1';
                              sd_reg1 <= sd_reg1;
                              adc_out1 <= adc_out1;
                              dac_in1 <= dac_in1;
                              sd_reg2 <= sd_reg2;
                              adc_out2 <= adc_out2;
                              dac_in2 <= dac_in2;
                              clk_960KHz <= '0';

                        WHEN X"16" => --
                              adac_state <= adac_state + 1;
                              cs_n <= '1';
                              sd_reg1 <= sd_reg1;
                              adc_out1 <= adc_out1;
                              dac_in1 <= dac_in1;
                              sd_reg2 <= sd_reg2;
                              adc_out2 <= adc_out2;
                              dac_in2 <= dac_in2;
                              clk_960KHz <= '0';

                        WHEN X"17" => --
                              adac_state <= adac_state + 1;
                              cs_n <= '1';
                              sd_reg1 <= sd_reg1;
                              adc_out1 <= adc_out1;
                              dac_in1 <= dac_in1;
                              sd_reg2 <= sd_reg2;
                              adc_out2 <= adc_out2;
                              dac_in2 <= dac_in2;
                              clk_960KHz <= '1';

                        WHEN X"18" => --
                              adac_state <= (others => '0');
                              cs_n <= '1';
                              sd_reg1 <= sd_reg1;
                              adc_out1 <= adc_out1;
                              dac_in1 <= dac_in1;
                              sd_reg2 <= sd_reg2;
                              adc_out2 <= adc_out2;
                              dac_in2 <= dac_in2;
                              clk_960KHz <= '0';

                        WHEN OTHERS =>--
                              adac_state <= (others => '0');
                              cs_n <= '1';
                              sd_reg1 <= sd_reg1;
                              adc_out1 <= adc_out1;
                              dac_in1 <= dac_in1;
                              sd_reg2 <= sd_reg2;
                              adc_out2 <= adc_out2;
                              dac_in2 <= dac_in2;
                              clk_960KHz <= '0';

                  END CASE;
```

```
              else -- reset in order
                  adac_state <= (others => '0');
            cs_n <= '1';
            sd_reg1 <= (others => '0');
            adc_out1 <= (others => '0');
            dac_in1 <= (others => '0');
            sd_reg2 <= (others => '0');
            adc_out2 <= (others => '0');
            dac_in2 <= (others => '0');
                  clk_960KHz <= '0';

              end if;
            end if;
      end process;

end Behavioral;
```

Appendix B.3 (CLIP_INDICATOR.VHD)

```
---------------------------------------------------------------------
--------------
-- Company:  KARABUK UNIVERSITY
-- Engineer: BILGEHAN ERKAL
--
-- Create Date: 19.05.2022 17:00:00
-- Design Name: Signed version
-- Module Name: clip_indicator - Behavioral
-- Project Name:
-- Target Devices:
-- Tool Versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
---------------------------------------------------------------------
--------------

library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_SIGNED.ALL;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
use IEEE.NUMERIC_STD.ALL;

-- Uncomment the following library declaration if instantiating
-- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity clip_indicator is
    Port (
       clip_in          : in std_logic_vector(11 downto 0);
       clip_out         : out STD_LOGIC;
       clk              : in  STD_LOGIC --960KHZ
       );

end clip_indicator;

architecture Behavioral of clip_indicator is

signal c_in_s : std_logic_vector(11 downto 0) := (others => '0');
signal timex : std_logic_vector(18 downto 0):= (others => '0');

begin

c_in_s <= clip_in(11 downto 0);
```

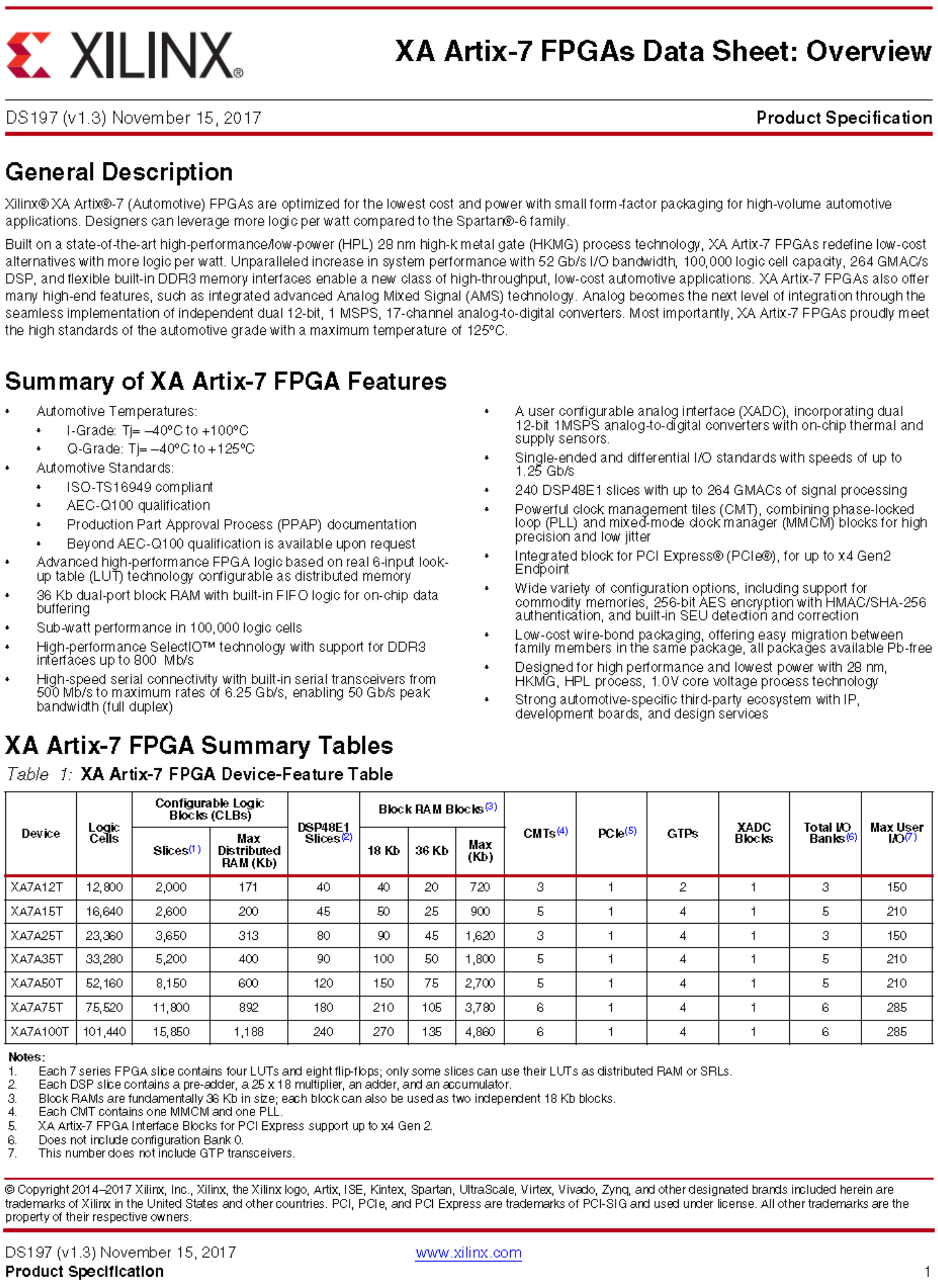
```
clip_proc:process(clk)
      begin
            if rising_edge(clk) then
                  if ((c_in_s < -1434) or (c_in_s > 1434)) then
                        clip_out <= '1';
                        timex <= "000" & X"0001";
                  else
                        if (timex = 0) then
                              clip_out <= '0';
                              timex <= timex;--(others => '0');
                        else
                              clip_out <= '1';
                              timex <= timex + 1;
                        end if;
                  end if;
            end if;
      end process;

end Behavioral;
```

APPENDIX C.

APPENDIX C. DATASHEETS OF CHIPS USED IN THE PROJECT

Appendix C.1 XILINX Artix-7 FPGA (CX7A35T-1CPG236C) Datasheet



XILINX®

XA Artix-7 FPGAs Data Sheet: Overview

DS197 (v1.3) November 15, 2017 **Product Specification**

General Description

Xilinx® XA Artix®-7 (Automotive) FPGAs are optimized for the lowest cost and power with small form-factor packaging for high-volume automotive applications. Designers can leverage more logic per watt compared to the Spartan®-6 family.

Built on a state-of-the-art high-performance/low-power (HPL) 28 nm high-k metal gate (HKMG) process technology, XA Artix-7 FPGAs redefine low-cost alternatives with more logic per watt. Unparalleled increase in system performance with 52 Gb/s I/O bandwidth, 100,000 logic cell capacity, 264 GMAC/s DSP, and flexible built-in DDR3 memory interfaces enable a new class of high-throughput, low-cost automotive applications. XA Artix-7 FPGAs also offer many high-end features, such as integrated advanced Analog Mixed Signal (AMS) technology. Analog becomes the next level of integration through the seamless implementation of independent dual 12-bit, 1 MSPS, 17-channel analog-to-digital converters. Most importantly, XA Artix-7 FPGAs proudly meet the high standards of the automotive grade with a maximum temperature of 125°C.

Summary of XA Artix-7 FPGA Features

- Automotive Temperatures:
  - I-Grade: Tj= –40°C to +100°C
  - Q-Grade: Tj= –40°C to +125°C
- Automotive Standards:
  - ISO-TS16949 compliant
  - AEC-Q100 qualification
  - Production Part Approval Process (PPAP) documentation
  - Beyond AEC-Q100 qualification is available upon request
- Advanced high-performance FPGA logic based on real 6-input look-up table (LUT) technology configurable as distributed memory
- 36 Kb dual-port block RAM with built-in FIFO logic for on-chip data buffering
- Sub-watt performance in 100,000 logic cells
- High-performance SelectIO™ technology with support for DDR3 interfaces up to 800 Mb/s
- High-speed serial connectivity with built-in serial transceivers from 500 Mb/s to maximum rates of 6.25 Gb/s, enabling 50 Gb/s peak bandwidth (full duplex)
- A user configurable analog interface (XADC), incorporating dual 12-bit 1MSPS analog-to-digital converters with on-chip thermal and supply sensors.
- Single-ended and differential I/O standards with speeds of up to 1.25 Gb/s
- 240 DSP48E1 slices with up to 264 GMACs of signal processing
- Powerful clock management tiles (CMT), combining phase-locked loop (PLL) and mixed-mode clock manager (MMCM) blocks for high precision and low jitter
- Integrated block for PCI Express® (PCIe®), for up to x4 Gen2 Endpoint
- Wide variety of configuration options, including support for commodity memories, 256-bit AES encryption with HMAC/SHA-256 authentication, and built-in SEU detection and correction
- Low-cost wire-bond packaging, offering easy migration between family members in the same package, all packages available Pb-free
- Designed for high performance and lowest power with 28 nm, HKMG, HPL process, 1.0V core voltage process technology
- Strong automotive-specific third-party ecosystem with IP, development boards, and design services

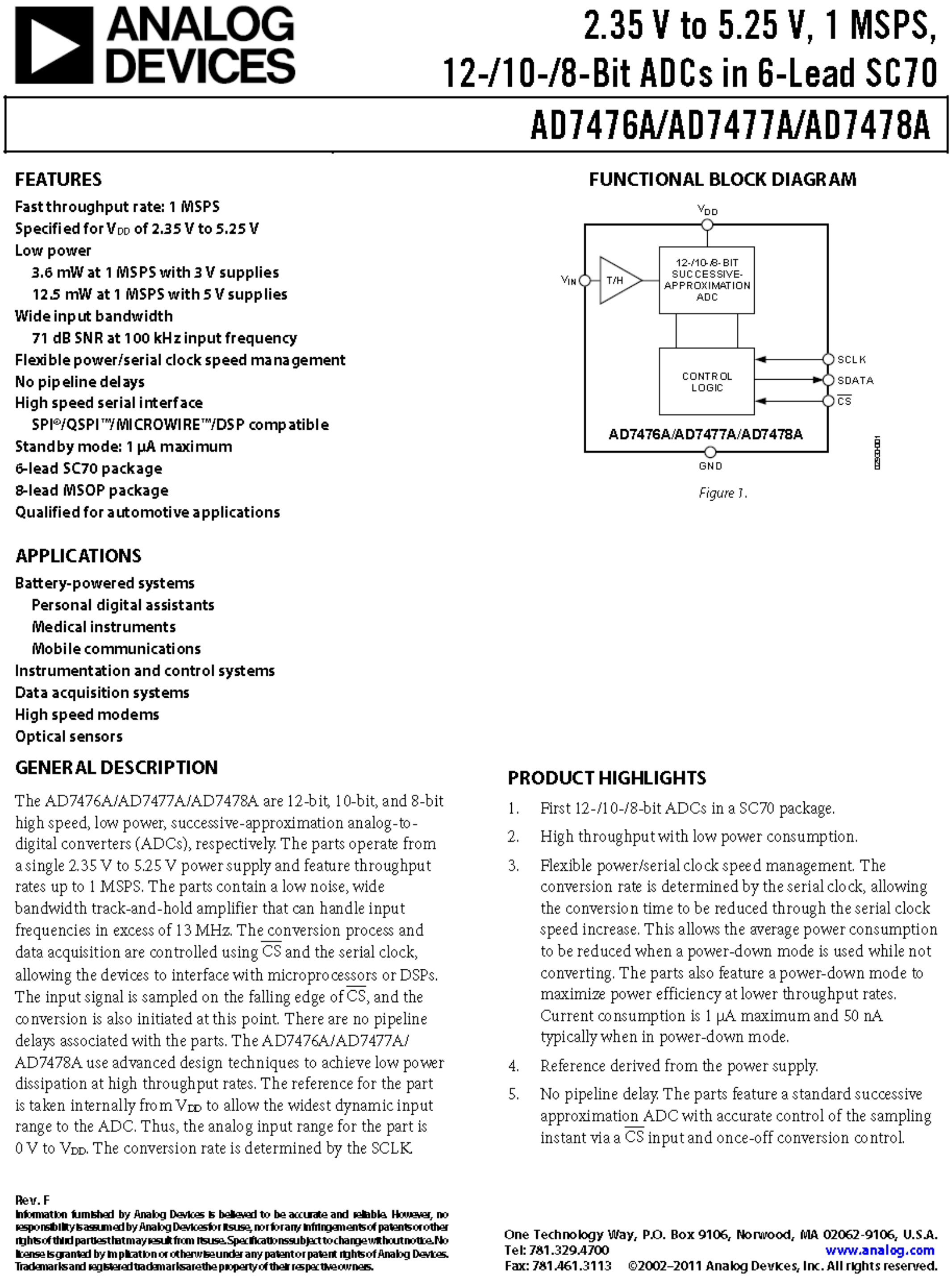
XA Artix-7 FPGA Summary Tables

Table 1: **XA Artix-7 FPGA Device-Feature Table**

Device	Logic Cells	Configurable Logic Blocks (CLBs)		DSP48E1 Slices[2]	Block RAM Blocks[3]			CMTs[4]	PCIe[5]	GTPs	XADC Blocks	Total I/O Banks[6]	Max User I/O[7]
		Slices[1]	Max Distributed RAM (Kb)		18 Kb	36 Kb	Max (Kb)						
XA7A12T	12,800	2,000	171	40	40	20	720	3	1	2	1	3	150
XA7A15T	16,640	2,600	200	45	50	25	900	5	1	4	1	5	210
XA7A25T	23,360	3,650	313	80	90	45	1,620	3	1	4	1	3	150
XA7A35T	33,280	5,200	400	90	100	50	1,800	5	1	4	1	5	210
XA7A50T	52,160	8,150	600	120	150	75	2,700	5	1	4	1	5	210
XA7A75T	75,520	11,800	892	180	210	105	3,780	6	1	4	1	6	285
XA7A100T	101,440	15,850	1,188	240	270	135	4,860	6	1	4	1	6	285

Notes:
1. Each 7 series FPGA slice contains four LUTs and eight flip-flops; only some slices can use their LUTs as distributed RAM or SRLs.
2. Each DSP slice contains a pre-adder, a 25 x 18 multiplier, an adder, and an accumulator.
3. Block RAMs are fundamentally 36 Kb in size; each block can also be used as two independent 18 Kb blocks.
4. Each CMT contains one MMCM and one PLL.
5. XA Artix-7 FPGA Interface Blocks for PCI Express support up to x4 Gen 2.
6. Does not include configuration Bank 0.
7. This number does not include GTP transceivers.

Appendix C.2 ADC Chip Analog Devices AD747



ANALOG DEVICES

2.35 V to 5.25 V, 1 MSPS, 12-/10-/8-Bit ADCs in 6-Lead SC70

AD7476A/AD7477A/AD7478A

FEATURES

Fast throughput rate: 1 MSPS
Specified for V_{DD} of 2.35 V to 5.25 V
Low power
- 3.6 mW at 1 MSPS with 3 V supplies
- 12.5 mW at 1 MSPS with 5 V supplies

Wide input bandwidth
- 71 dB SNR at 100 kHz input frequency

Flexible power/serial clock speed management
No pipeline delays
High speed serial interface
- SPI®/QSPI™/MICROWIRE™/DSP compatible

Standby mode: 1 µA maximum
6-lead SC70 package
8-lead MSOP package
Qualified for automotive applications

APPLICATIONS

Battery-powered systems
- Personal digital assistants
- Medical instruments
- Mobile communications

Instrumentation and control systems
Data acquisition systems
High speed modems
Optical sensors

FUNCTIONAL BLOCK DIAGRAM

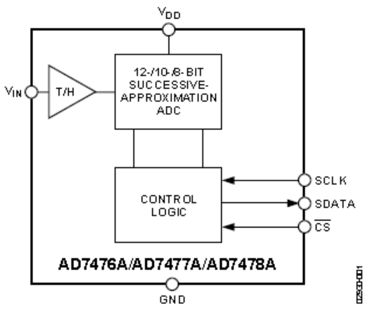


Figure 1.

GENERAL DESCRIPTION

The AD7476A/AD7477A/AD7478A are 12-bit, 10-bit, and 8-bit high speed, low power, successive-approximation analog-to-digital converters (ADCs), respectively. The parts operate from a single 2.35 V to 5.25 V power supply and feature throughput rates up to 1 MSPS. The parts contain a low noise, wide bandwidth track-and-hold amplifier that can handle input frequencies in excess of 13 MHz. The conversion process and data acquisition are controlled using $\overline{CS}$ and the serial clock, allowing the devices to interface with microprocessors or DSPs. The input signal is sampled on the falling edge of $\overline{CS}$, and the conversion is also initiated at this point. There are no pipeline delays associated with the parts. The AD7476A/AD7477A/AD7478A use advanced design techniques to achieve low power dissipation at high throughput rates. The reference for the part is taken internally from V_{DD} to allow the widest dynamic input range to the ADC. Thus, the analog input range for the part is 0 V to V_{DD}. The conversion rate is determined by the SCLK

PRODUCT HIGHLIGHTS

1. First 12-/10-/8-bit ADCs in a SC70 package.
2. High throughput with low power consumption.
3. Flexible power/serial clock speed management. The conversion rate is determined by the serial clock, allowing the conversion time to be reduced through the serial clock speed increase. This allows the average power consumption to be reduced when a power-down mode is used while not converting. The parts also feature a power-down mode to maximize power efficiency at lower throughput rates. Current consumption is 1 µA maximum and 50 nA typically when in power-down mode.
4. Reference derived from the power supply.
5. No pipeline delay. The parts feature a standard successive approximation ADC with accurate control of the sampling instant via a $\overline{CS}$ input and once-off conversion control.

One Technology Way, P.O. Box 9106, Norwood, MA 02062-9106, U.S.A.
Tel: 781.329.4700 www.analog.com
Fax: 781.461.3113 ©2002–2011 Analog Devices, Inc. All rights reserved.

6 Datasheet

Appendix C.3 Texas Instruments DAC121S101 DAC Chip Datasheet

TEXAS INSTRUMENTS

DAC121S101, DAC121S101-Q1

SNAS265J – JUNE 2005 – REVISED SEPTEMBER 2015

DAC121S101/-Q1 12-Bit Micro Power, RRO Digital-to-Analog Converter

1 Features

- DAC121S101-Q1 is AEC-Q100 Grade 1 Qualified and is Manufactured on an Automotive Grade Flow.
- Ensured Monotonicity
- Low Power Operation
- Rail-to-Rail Voltage Output
- Power-on Reset to Zero Volts Output
- Wide Temperature Range of –40°C to +125°C
- Wide Power Supply Range of 2.7 V to 5.5 V
- Small Packages
- Power Down Feature
- Key Specifications
  - 12-Bit Resolution
  - DNL -0.15, +0.25 LSB (Typical)
  - 8-µs Output Settling Time (Typical)
  - 4-mV Zero Code Error (Typical)
  - Full-Scale Error at −0.06 %FS (Typical)
  - 0.64-mW (3.6-V) / 1.43-mW (5.5-V) Normal Mode Power Consumption (Typical)
  - 0.14-µW (3.6-V) / 0.39-µW (5.5-V) Power-Down Mode (Typical)

2 Applications

- Battery-Powered Instruments
- Digital Gain and Offset Adjustment
- Programmable Voltage and Current Sources
- Programmable Attenuators
- Automotive

3 Description

The DAC121S101 device is a full-featured, general-purpose, 12-bit voltage-output digital-to-analog converter (DAC) that can operate from a single 2.7-V to 5.5-V supply and consumes just 177 µA of current at 3.6 V. The on-chip output amplifier allows rail-to-rail output swing and the three wire serial interface operates at clock rates up to 30 MHz over the specified supply voltage range and is compatible with standard SPI™, QSPI, MICROWIRE and DSP interfaces. Competitive devices are limited to 20-MHz clock rates at supply voltages in the 2.7 V to 3.6 V range.

The supply voltage for the DAC121S101 serves as its voltage reference, providing the widest possible output dynamic range. A power-on reset circuit ensures that the DAC output powers up to zero volts and remains there until there is a valid write to the device. A power-down feature reduces power consumption to less than a microWatt.

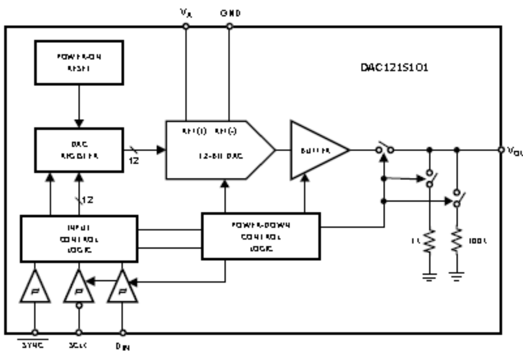
The low power consumption and small packages of the DAC121S101 make it an excellent choice for use in battery operated equipment.

Device Information(1)

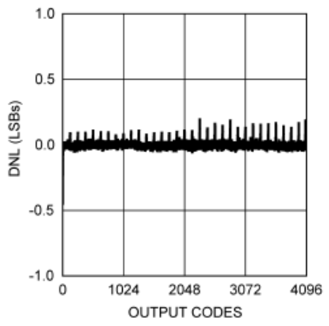
PART NUMBER	PACKAGE	BODY SIZE (NOM)
DAC121S101	SOT (6)	2.90 mm × 1.60 mm
	VSSOP (8)	3.00 mm × 3.00 mm
DAC121S101-Q1	SOT (6)	2.90 mm × 1.60 mm

(1) For all available packages, see the orderable addendum at the end of the data sheet.

Simplified Block Diagram



DNL vs. Output Code



RESUME

Ahmed Ibrahim M ALGHHRIUNI completed high school education in Petroleum Training and Qualifying Institute, after that, he started undergraduate program in High Institute of Industrial Technology Department of Electrical and Electronics Engineering in 2012. Then in 2021, he started M. Sc. Education at Karabük University Department of Electrical and Electronics Engineering.