ABSTRACT

With the advent of technology during the recent years, the communication has been majorly into digital domain. The modulation and demodulation schemes are implemented in the digital domain mostly. Also the systems are designed to be versatile in such a way that they are capable of adapting to various changes in terms of the modulation schemes, data rates and various other parameters.

With this idea in background, the goal of project is to develop a system which is to be implemented in the digital domain and is adaptive as well. This further transformed into the idea of developing an adaptive FM modulation and demodulation system. In order to implement the system in digital domain the hardware platform suitable is FPGA.

The generation of carriers for modulation is implemented inside the FPGA digitally using NCO which is the digital counterpart for voltage controlled oscillator. The demodulation of the FM wave is done using a digital PLL which is a non-coherent method of demodulation.

Further the system is made adaptive by providing selectivity with respect to carrier frequency, frequency sensitivity which decides the depth of modulation using a multiplexer architecture. For sinusoidal input signals, selectivity is provided to change the message frequency. To incorporate this feature the loop filter part of the DPLL is implemented on a separate microcontroller (ARDUINO DUE) and data is sent to and from the FPGA using the digital I/O pins on FPGA and the port pins on the microcontroller.

TABLE OF CONTENTS

ACRONYMS & ABBREVATIONS

AC	Alternating Current
ADC	Analog to Digital Converter
ADPLL	All Digital Phase Locked Loop
CRO	Cathode Ray Oscilloscope
DAC	Digital to Analog Converter
DC	Direct Current
DCO	Digitally Controlled Oscillator
DDS	Direct Digital Synthesizer
DMA	Direct Memory Access
DPLL	Digital Phase Locked Loop
DSP	Digital Signal Processing
DTMF	Dual Tone Multi Frequency
EEPROM	Electrically Erasable Programmable Read Only Memory
FCW	Frequency Control Word
FIR	Finite Impulse Response
FM	Frequency Modulation
FPGA	Field Programmable Gate Array
IDE	Integrated Development Environment
IIR	Impulse Invariant Response
LF	Loop Filter
LPF	Low Pass Filter
LUT	Look Up Tables
MCU	Micro Controller Unit
NCO	Numerically Controlled Oscillator
OTG	On the Go
PA	Phase Accumulator
PAC	Phase to Amplitude Convertor
PD	Phase Detector
PLL	Phase Locked Loop
PROM	Programmable Read Only Memory
PWM	Pulse Width Modulation
RAM	Random Access Memory
RF	Radio Frequency
ROM	Read Only Memory
RX	Receiver

SOC	System on Chip
TX	Transmitter
UART	Universal Asynchronous Receiver/Transmitter
USB	Universal System Bus
VCO	Voltage Controlled Oscillator
VHDL	VHSIC Hardware Description Language
VHSIC	Very High Speed Integrated Circuit

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

1.1 INTRODUCTION:

In the past radio receivers were designed with analog circuitry. This inherently has the same problems that all analog circuits have. That is, they are

susceptible to temperature variations, electrical noise, component aging, and they are complicated and inflexible. Initially as digital circuits and processors were developed, they were not useful for radio or any high frequency circuitry since they operated at low frequencies and their transistor density was not enough for the signal processing needed in receivers. However, with the exponential increase in transistor density, faster clock rates, and faster A/D converters radio frequency receivers and possibly higher frequency receivers and transmitters are now suited for the digital domain. Because of the advantages of digital communication systems, a concept of Software Defined Radio (SDR) has become popular in the literature. The ideal concept of SDR is to sample the RF signal with as little as possible analog manipulation. That is, ideally we would have an A/D converter at the output of an antenna and do the entire signal processing in the digital domain. However, sufficiently fast A/D converters are not cheap enough yet, therefore we still require a front end to generate an intermediate frequency to sample. Once the signal is in the digital domain the designer has all the benefits of digital signal processing as described before and the ease of configuration and reconfiguration.

Software-defined radio (SDR) is a radio communication system where components that have been typically implemented in hardware are instead implemented by means of software on a personal computer or embedded system.

In the current project we are trying to implement an FM modulation and demodulation system which is capable of adapting to selective message frequency, carrier frequency and frequency sensitivity.

The modulation and demodulation of FM are implemented using a non-coherent mechanism i.e. using a Digital PLL. As mentioned above implementing the phase locked loop using the analog circuitry might result in an inefficient system. Hence the system is implemented using FPGA platform.

To make the system adaptive to selective carrier and message frequencies and frequency sensitivity a multiplexer kind of logic is used providing the flexibility to select one among the multiple options.

In a similar manner the system cannot be made adaptive to variable message frequency on FPGA, as the loop filter cannot be varied on FPGA platform. Hence

the loop filter portion of the phase locked loop is implemented on a 32 bit ARM based microcontroller (Arduino Due).

1.2 PROBLEM STATEMENT

The project aims at developing a digital FM modulation and demodulation system for sinusoidal signals and audio samples. The system is to be made adaptive by providing selectivity with respect to carrier frequency, message frequency and frequency sensitivity.

The demodulation is carried out using a non-coherent demodulation system i.e. without the use of a local oscillator by using a digital phase locked loop. The DPLL is to be designed consisting of phase detector, loop filter and NCO.

The phase detector is implemented as a multiplier in time domain which is equivalent to a shift in the frequency domain producing two frequencies which are the sum and difference of frequencies at its input.

The loop filter is a digital IIR Butterworth 2^{nd} order filter implemented using the direct form 2 structure. The filter is to be implemented on the Arduino Due which is a 32 bit ARM based microcontroller (ARM Cortex M3).

NCO is the digital counter part of Voltage controlled oscillator which generates signals using sampled constants. The NCO is implemented on the FPGA which is to be designed to track the carrier frequency. The frequency output of the NCO varies accordingly with the external input.

The DPLL is to be integrated consisting of the phase detector and NCO on the FPGA and the loop filter on the Arduino. The data to and from the FPGA is to be sent using digital I/O pins on the FPGA. On the Arduino, the data is read and sent using 32 bit port registers.

1.3 PROJECT SCOPE AND CONSTRAINTS

The design implemented in the current project aims at developing a system capable of extracting different modulating signals modulated using different carriers and different frequency sensitivities. One of the research journals aims at developing a versatile system capable of demodulating both audio and video FM

broadcast signals. Such a system must be capable of adapting to variable carrier and data rates as well as variable modulation depth.

Some of constraints of the system include Memory and Sampling rate.

The Arduino due has a clock of 84 MHz. But the number of cycles required for sampling and providing the ADC value is very high resulting in a maximum sampling frequency of 666 kHz (period 1.5 us). As a result of this low sampling frequency the carrier for FM modulation is restricted to less than 50 kHz in accordance with the Nyquist theorem.

Also the Arduino due board has a very low flash memory of 512kB which puts a constraint on the number of audio samples for modulation. A maximum of 40000 samples can be stored (32 bit samples). At a sampling rate of 8 kHz, the audio signals will have a maximum length of 5 seconds.

1.4 COMPONENT LIST

- Arduino DUE
- Virtex 5 FPGA
- 8 bit DAC
- Oscilloscope
- Signal Generator

1.5 PROJECT FLOW

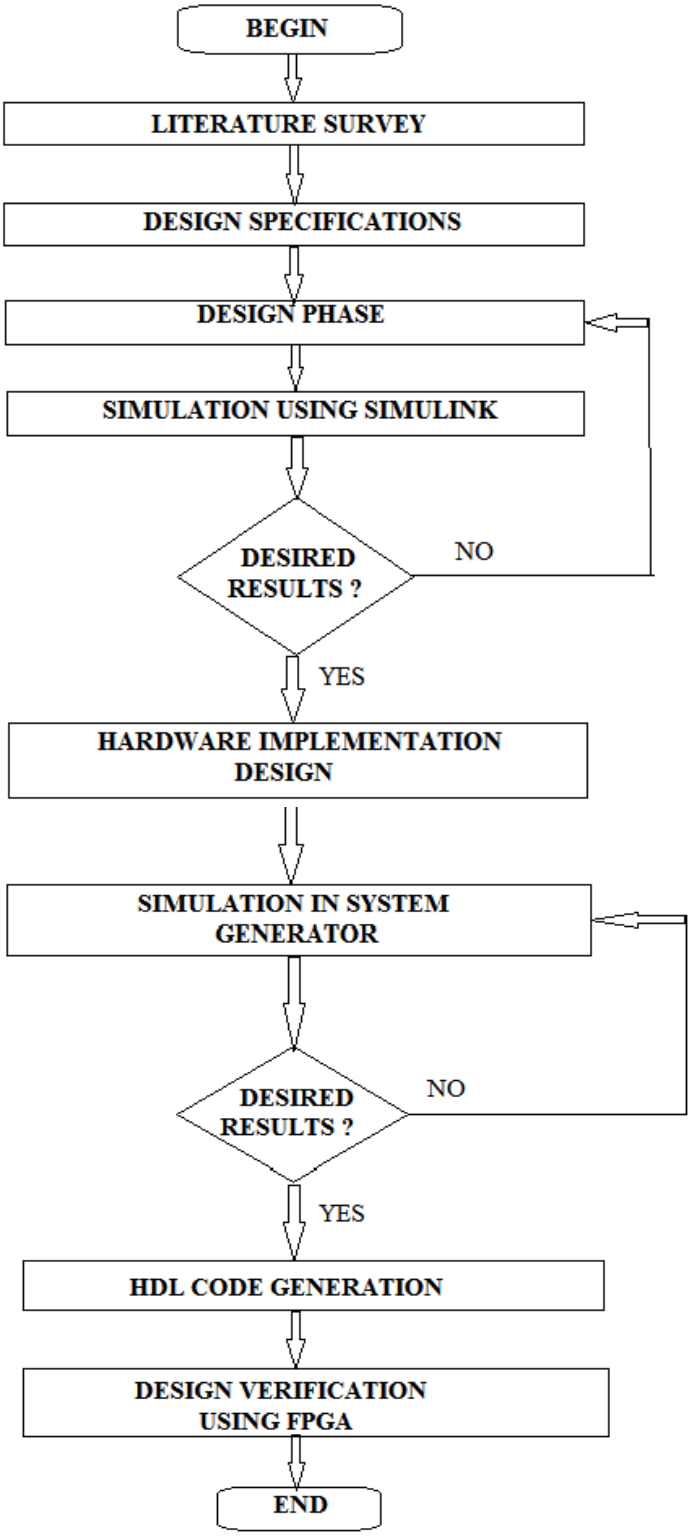


Fig 1.1 Project Flow

CHAPTER 2

LITERATURE SURVEY

2.1 DIGITAL DEMODULATION:

"FM Radio Receiver with Digital Demodulation" by N Burnett. A Senior Project presented to the Faculty of the Electrical Engineering Department California Polytechnic State University, San Luis Obispo. This paper reports on the design, construction, and testing of an FM receiver. The design is split into two portions, the analog FM front end and the digital demodulator. The job of the front end is to down convert the RF signal to a frequency that is low enough to sample with an analog to digital converter. The construction of the front end is done using what is known as "Ugly Construction". That is, all the components are soldered together floating over a ground plane. The second portion of the design is the demodulator. The phase lock loop method of demodulating FM signals is used. The phase lock loop demodulator is designed in the digital domain on an FPGA. The design approach used for the demodulator is a digital hardware implementation using VHDL. [2]

2.2 PHASE LOCKED LOOP:

"Design of a Digital FM Demodulator based on a 2nd Order All-Digital Phase-Locked Loop". Juan Pablo Martinez Brito, Sergio Bampi PGMICRO. Federal University of Rio Grande do Sul, UFRGS 91501-970 Porto Alegre RS Brazil. A software-defined radio (SDR) is a wireless communication device in which all of the signal processing is implemented in software. By simply downloading a new program, a SDR is able to interoperate with different wireless protocols, incorporate new services, and upgrade to new standards. Therefore, FPGAs have been used extensively for implementing essential functions in SDR architectures. In this paper, we explore the design of a Digital FM Receiver using the approach of an All-Digital Phase Locked-Loop (ADPLL). The digital FM Receiver circuit is designed using pure VHDL, then simulated and synthesized using Model Sim SE 6 and Leonardo Spectrum Level 3, respectively. The final circuit operates at a frequency up to 150MHz and occupies the area around 15K logic gates.

"Design of Digital Phase Locked Loop for Wireless Communication Receiver Application". Pradnya H.Golghate, Prof.Pankaj Hedaoo. International Journal of Advanced Research in Computer and Communication Engineering. In this paper, Frequency modulated receiver is designed using Digital phase locked loop circuitry which consists of Booth s multiplier, Loop filter and numerically controlled oscillator. This design is modelled in Verilog synthesis and performed place and route for design using Xilinx 13.1.In this paper, we propose a numerically controlled oscillator that can be tuned to desirable frequency according to the requirement. This design also achieves small area and small power consumption as compared to typical classical method of design. [3]

2.3 FM DEMODULATOR:

"DIGITAL FM MODULATOR AND DEMODULATOR FOR SDR". Richa Goel (Department of Electronics, Roorkee College of Engineering, Roorkee, India). INTERNATIONAL JOURNAL OF ENGINEERING SCIENCES & RESEARCH TECHNOLOGY. This paper deals with an FPGA implementation of a high performance FM modulator and demodulator for software defined Radio (SDR) system. The individual component of proposed FM modulator and demodulator has been optimized in such a way that the overall design consists of a high-speed, area optimized and low-power features.

"Design and Implementation of FM modem on FPGA for SDR using Simulink". Lavanya T and Natraj URS HD. International Journal of Engineering Research Volume No.5 Issue: Special 5, pp: 992 -1128. This paper describes the implementation of FM modulation and demodulation for Software Defined Radio (SDR). Due to high performance and re-configurability, signal processing is done on FPGA. Analog modulation scheme like frequency modulation is performed in digital domain on virtex4 FPGA. [4]

2.4 NUMERICALLY CONTROLLED OSCILLATOR:

"A Numerically controlled oscillator for all Digital Phase Locked Loop". Anupama Patil and Dr P .H .Tandel International Journal of Engineering Trends and Technology (IJETT) – Volume 38 Number 4-August 2016. Numerically Controlled Oscillator (NCO) is a digital oscillator signal generator. It generates a synchronous, clocked, discrete waveform, usually sinusoidal. NCOs are often used in combination with a digital to analog converter (DAC) at the output to create a direct digital synthesizer (DDS). NCOs are used in many communications systems which are completely digital or mixed signal, like arbitrary waveform synthesis, precise control for phased array radar or SONAR systems. A FPGA based Implementation method greatly improves the performance, reduces the development cycle and reduces cost. The basic NCO architecture is improved and enhanced with the minimum hardware in order to facilitate the complete system level support for different kinds of modulation with minimal FPGA resources. This paper presents the implementation of Sinusoidal Wave Generation using NCO module which improves the performance, reduces the power & area requirement. [6]

CHAPTER 3

BLOCK DIAGRAM AND DESCRIPTION

The present chapter elaborates the system block diagram and the sub blocks with the help of which it is realized. The illustration of how the sub blocks are realized with the help of Xilinx block set is as explained below.

3.1 BLOCK DIAGRAM

Fig 3.1 depicts the general block diagram of the digital PLL with FM input and Demodulation output.

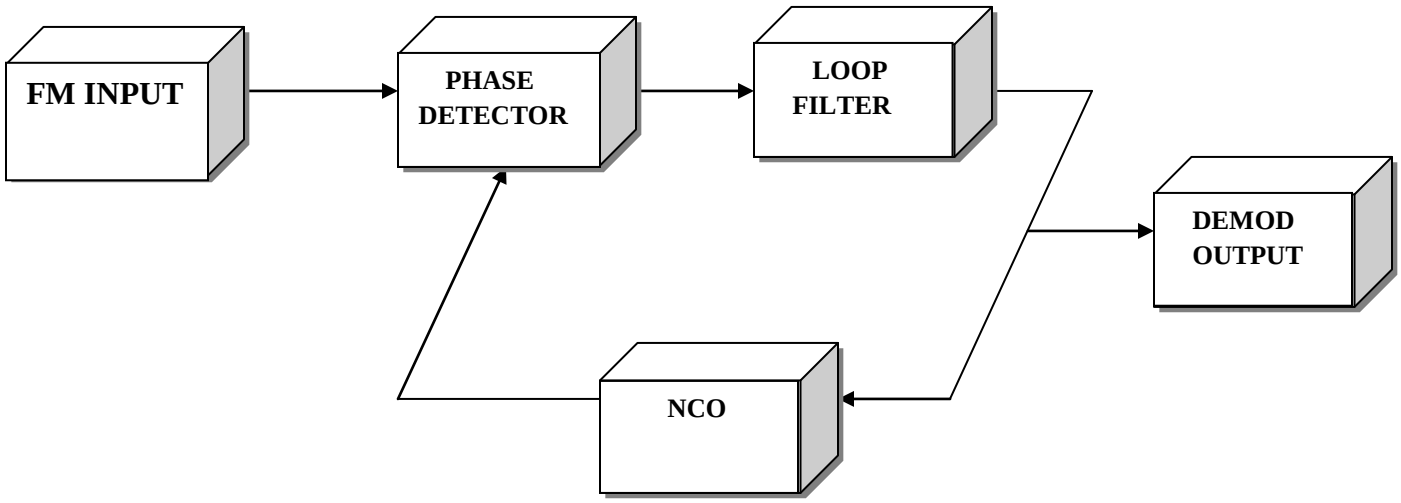


Fig 3.1 System Block Diagram

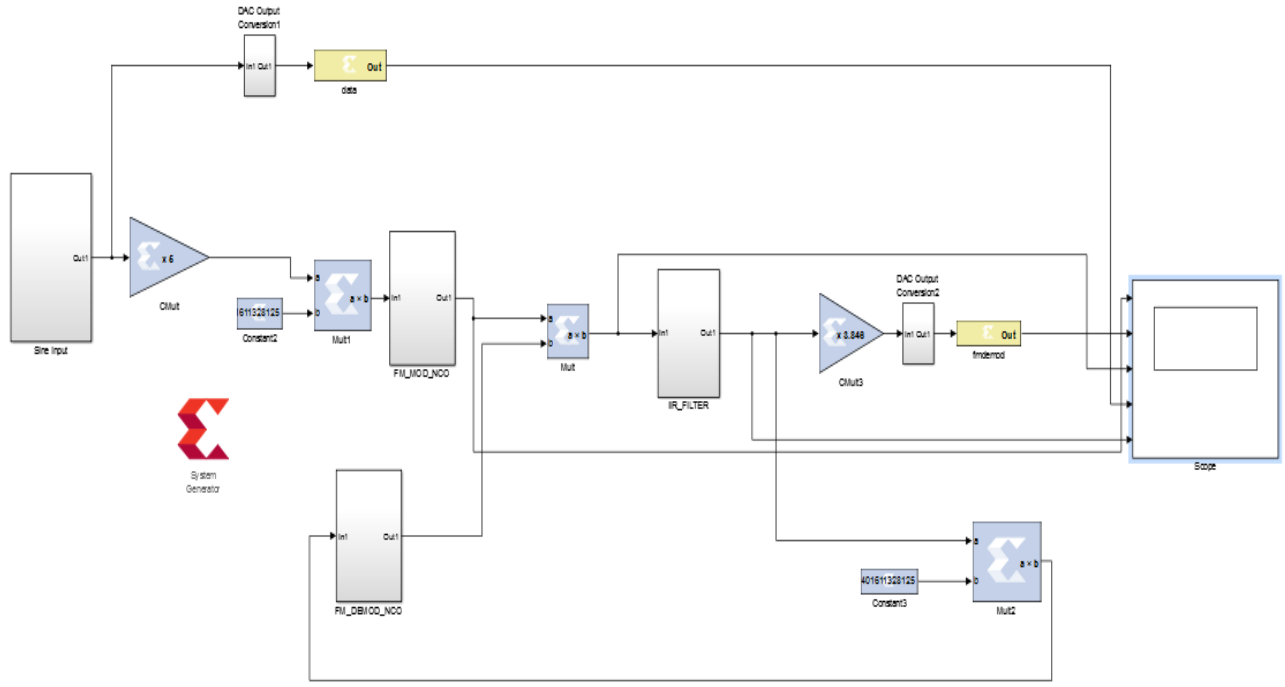


Fig 3.2 System Block Diagram in Xilinx Block set without Arduino

Fig 3.2 depicts a detailed block diagram of the system consisting of a single sinusoidal input signal with a particular value of sensitivity for FM modulator. The FM modulated signal is given as input to the phase detector. The phase detector in the digital domain is implemented as a multiplier with two inputs. The output from the phase detector is given to the digital filter (IIR Butterworth 2^{nd} order Low pass) which extracts the modulating signal from the phase detector output which consists of sum and difference of frequencies given at its input. The Output from the filter is given as feedback to the NCO which tracks the deviation from the carrier frequency. The loop in this manner extracts the message signal by tracking the frequency variations in the FM signal.

Fig 3.3 is similar to Fig 3.2 but variation is given with respect to message (sinusoidal) signal frequency, carrier frequency and frequency sensitivity making the system adaptive.

Unlike the block diagram depicted in fig 3.2, in the real system the loop filter is not implemented on FPGA in order to keep up with the constraint of variation in message frequency as mentioned in the problem statement.

Hence the loop filter here is designed and implemented on ARDUINO DUE which is a 32 bit ARM based microcontroller. The data from the phase detector on FPGA is read using PORT C on Arduino and data is feedback to NCO on FPGA using PORT B.

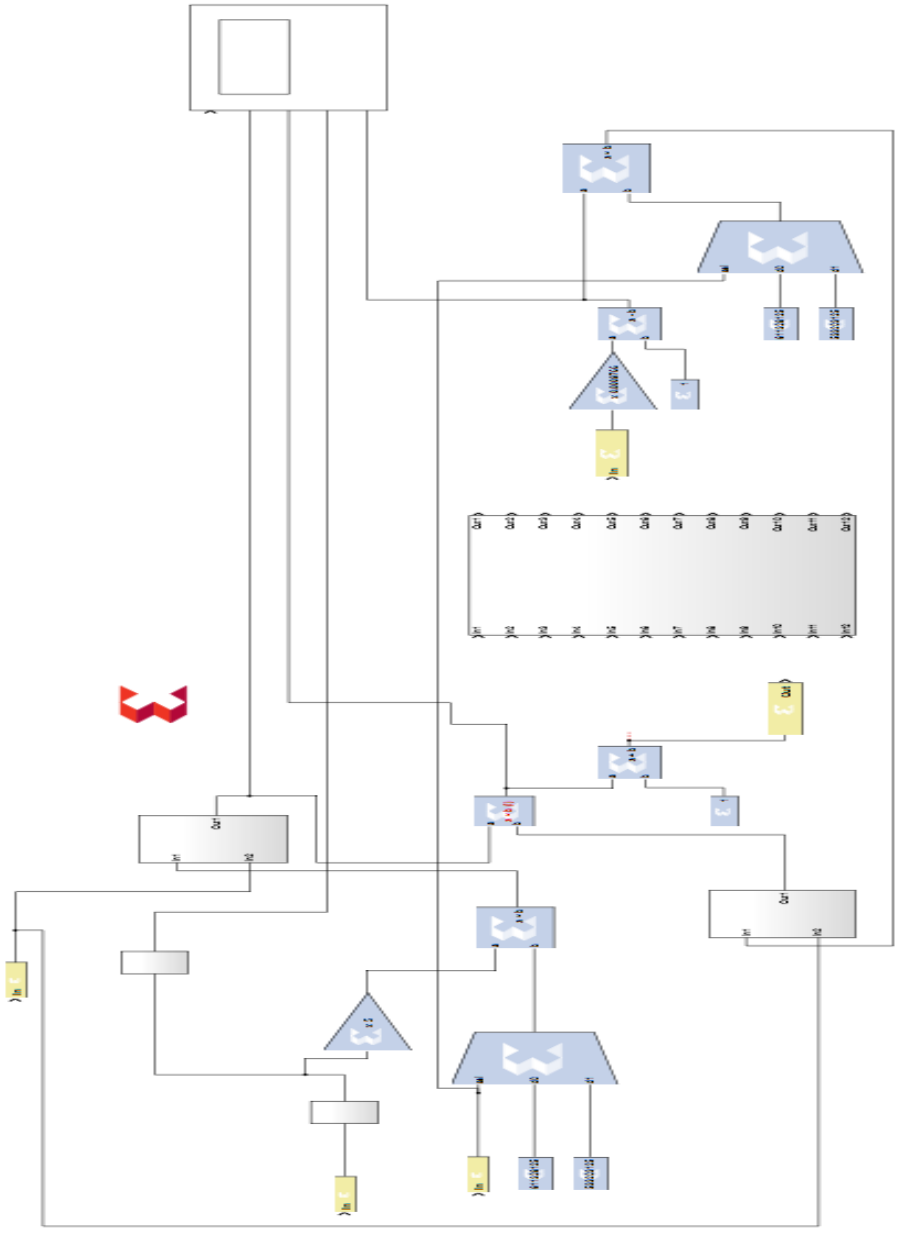


Fig 3.3 System Block Diagram in Xilinx Block set with Arduino

Fig 3.4 depicts the system block diagram for audio input

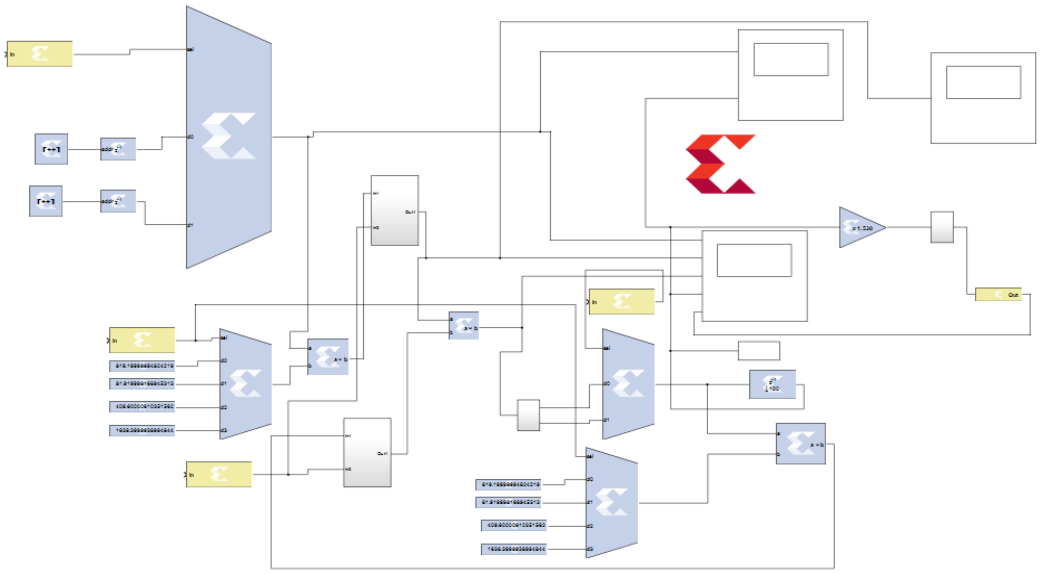


Fig 3.4 Block Diagram for audio input

The above block diagram depicts the audio samples modulation and demodulation system. Here the audio samples which are being modulated are stored inside the FPGA ROM. Two different samples are provided as input for modulation, one of which could be chosen using a multiplexer. Selectivity is provided for carrier as well as frequency sensitivity. Four different carriers of frequencies 80kHz, 60kHz, 100kHz and 200kHz are provided and four different frequency sensitivities of values 1kHz, 5kHz, 10kHz and 20kHz. The audio samples are sampled at 8 kHz, as the audio frequencies are in the range 300Hz to 3.4kHz and keeping in accordance with Nyquist's sampling constraint. The carrier is chosen to be 80kHz and the sampling rate for FM signals is 800kHz. The phase detector and the loop filter as explained in the previous block diagrams are implemented as a 32-bit multiplier and a 2^{nd} order digital IIR Butterworth filter. The demodulated signal will have a sampling frequency of 800kHz. Hence to playback the demodulated audio, it must be down sampled (decimated) by a factor of 100 resulting in a sampling frequency of 8kHz. These samples are assigned to the digital Output pins on FPGA using 12-bit resolution (as depicted in the diagram assigned to the Gateway Out block). This data from the digital pins is read using port pins on Arduino and the samples are sent to Matlab on the host PC using serial communication to playback the demodulated audio samples and draw inferences on the modulation and demodulation techniques.

FM INPUT:

Fig 3.5 depicts the FM modulator block diagram.

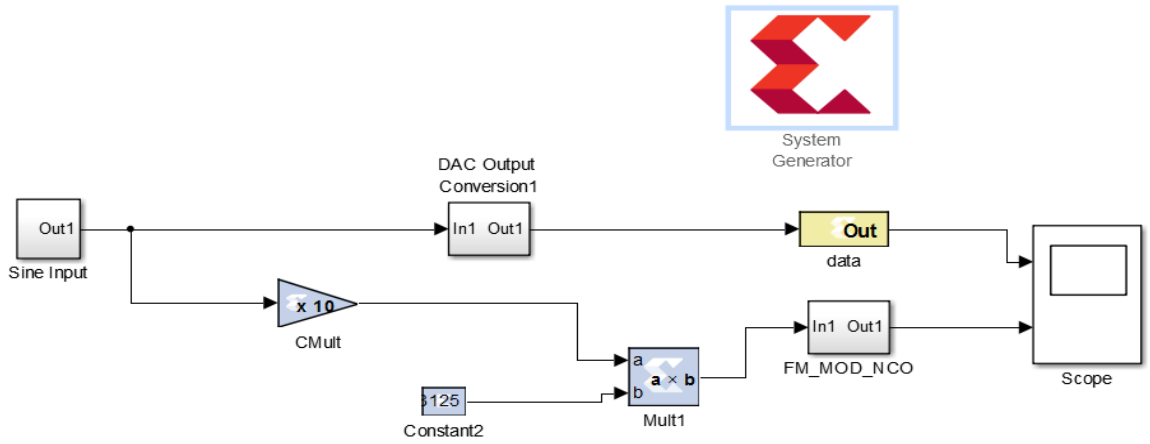


Fig 3.5 FM Input

Fig 3.6 depicts the modulating signal and the resulting FM signal.

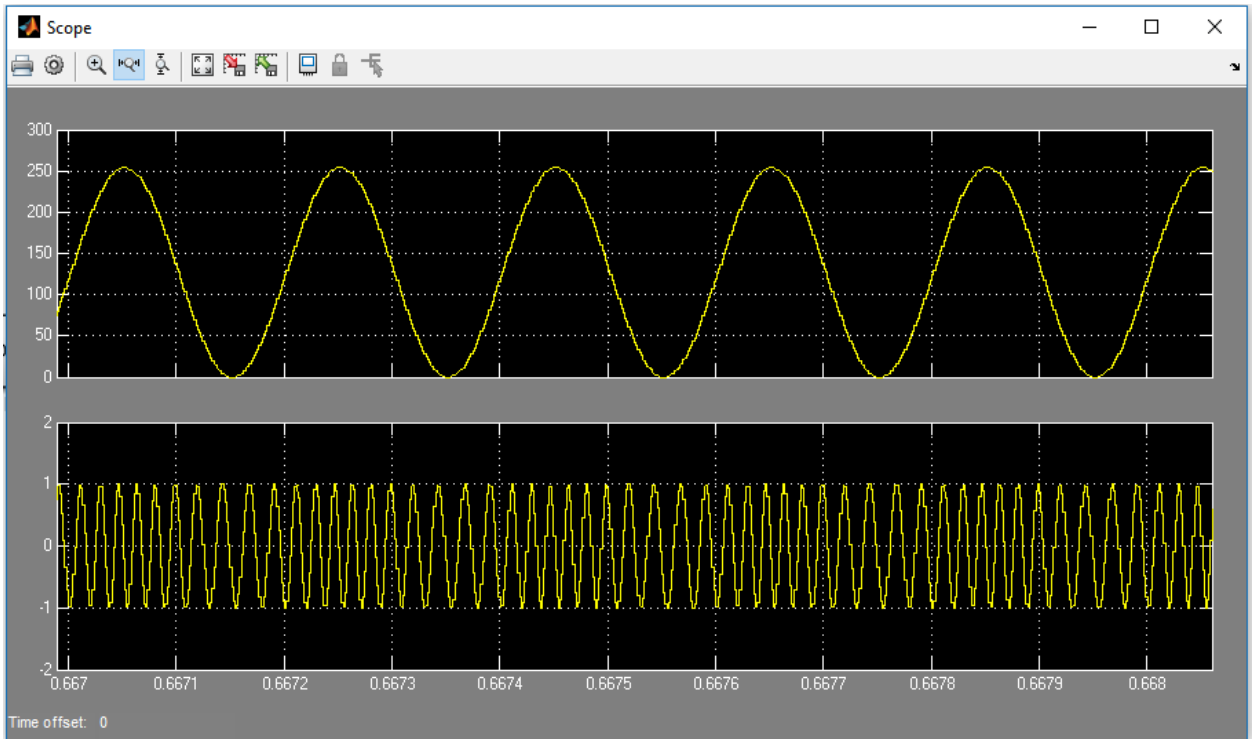


Fig 3.6 FM waveform

The mechanism used for generating FM is by giving the modulating signal as an input to the NCO. With external signal given as input to the NCO, the frequency of the output signal varies according to the variations in the modulating signal. In the Xilinx block set in order ensure that the output of NCO varies in accordance with the external input, the modulating signal has to be multiplied with the constant of value equal to the required frequency sensitivity which is further given as input to the NCO block.

PHASE DETECTOR:

Fig 3.7 depicts the Phase Detector Block Diagram.

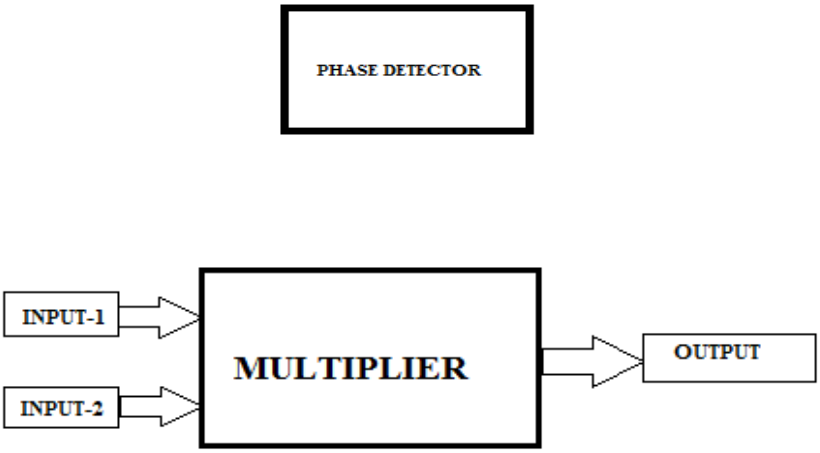


Fig 3.7 Phase Detector Block Diagram

Fig 3.8 depicts the Phase Detector in Xilinx Block set.

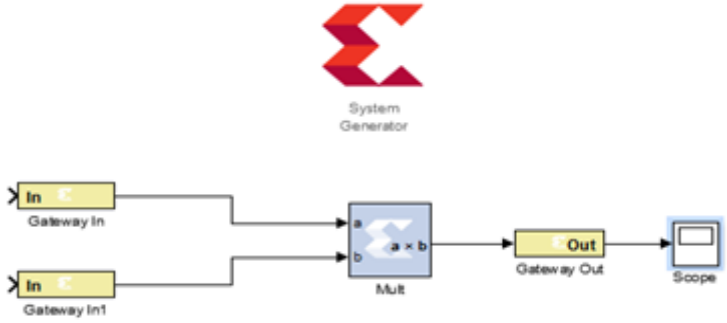


Fig 3.8 Phase Detector in Xilinx Block set

Phase detector in the digital domain is implemented as a multiplier with two inputs in Xilinx block set. One input is the FM modulating signal and the other input is the output of the signal from the NCO block to which the input is from the loop filter at the demodulation end. The output which consists of sum and difference of the frequencies is given as input to the loop filter.

LOOP FILTER:

Fig 3.9 depicts the Loop Filter Block Diagram.

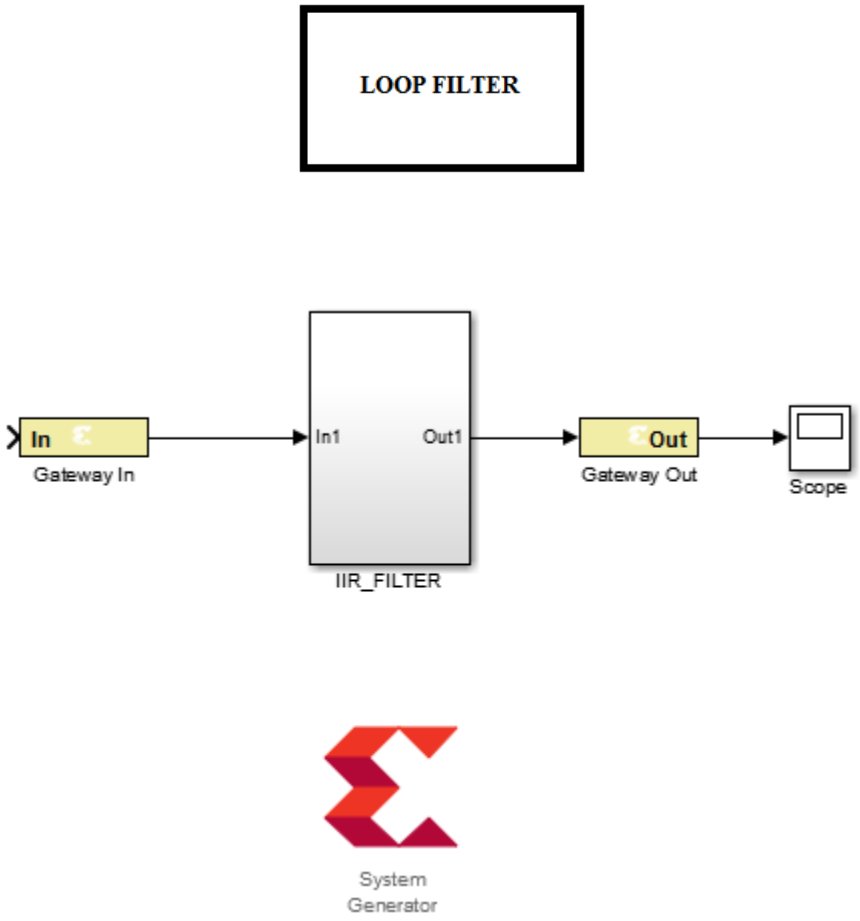


Fig 3.9 Loop Filter Block Diagram

Fig 3.10 depicts the loop filter structure (Direct Form-II)

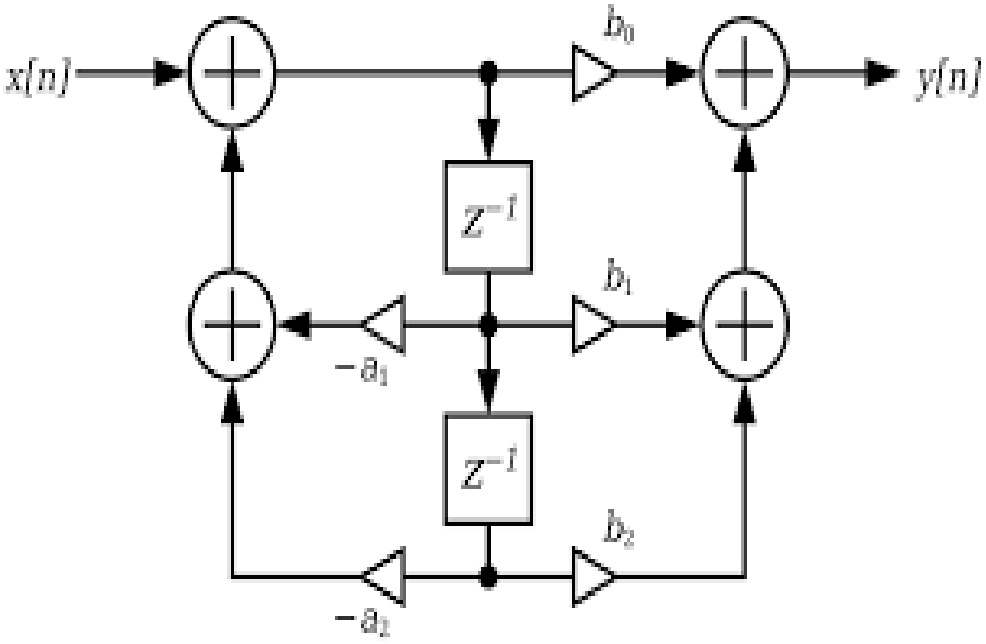


Fig 3.10 Loop Filter Structure

Fig 3.11 depicts the Loop filter using the Xilinx Block set.

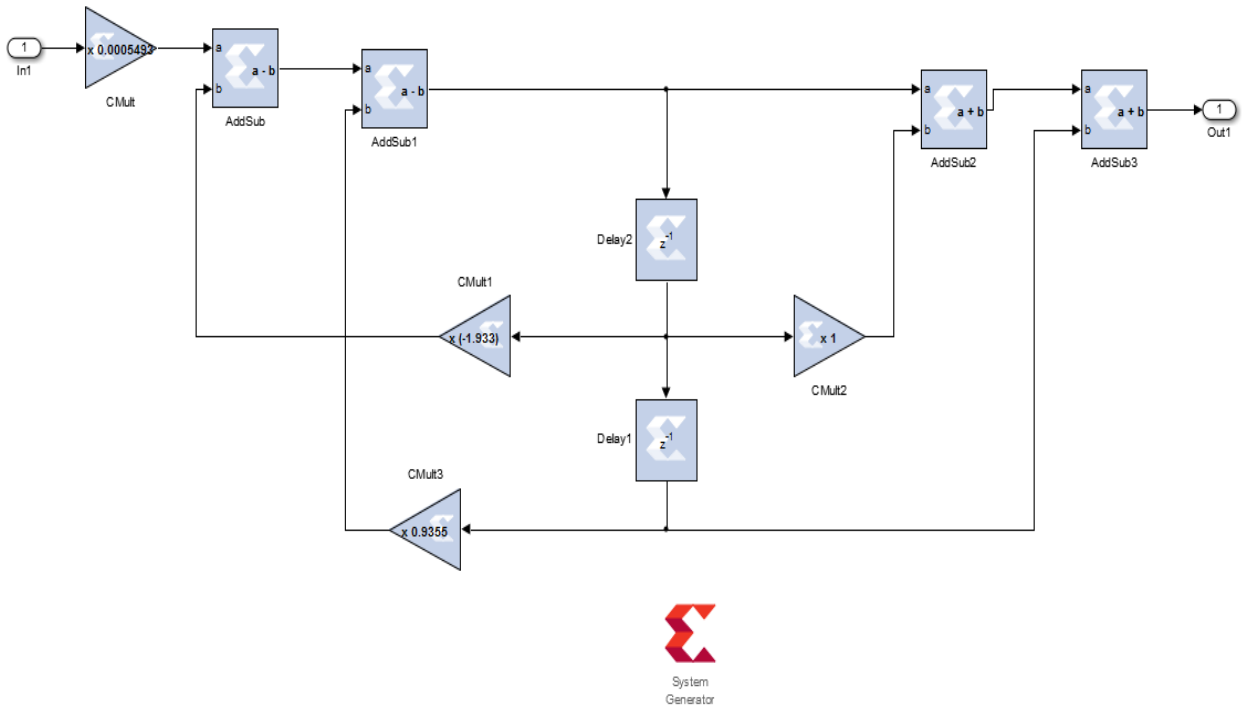
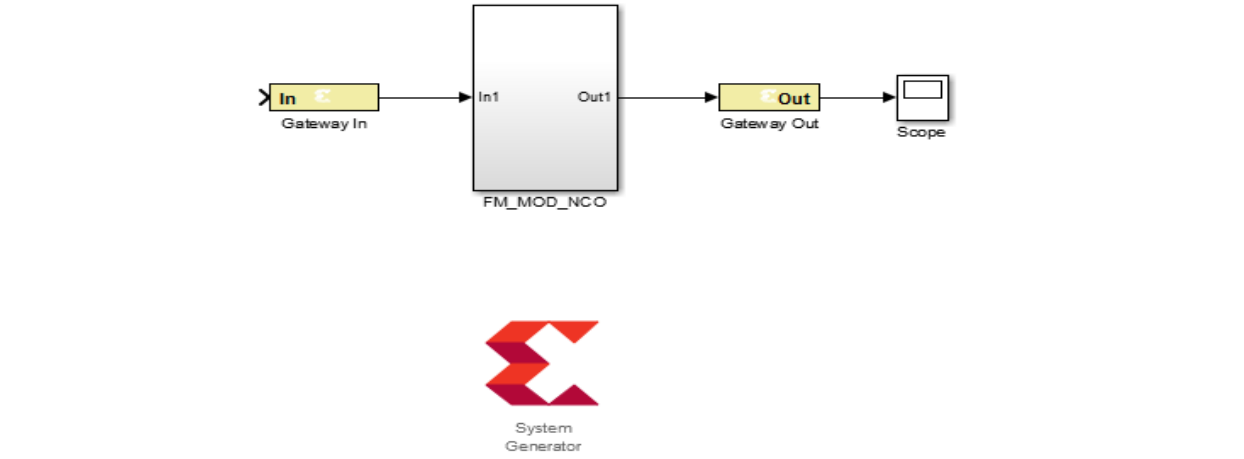


Fig 3.11 Loop Filter Using Xilinx Block set

The loop filter designed is a Butterworth IIR low pass filter of order two and is realized using direct form structure –II. The filter is built with the help of the adders, constants and delay block and is designed to have a cut off frequency equal to that of the modulating signal. The input to the loop filter is the signal from the phase detector which consists of sum and difference of frequencies given at its input. The filter based on its cut-off extracts the difference (lower) frequency. The output from the filter is given as feedback to the NCO at the demodulation side.

NUMERICALLY CONTROLLED OSCILLATOR (NCO):

Fig 3.12 depicts the NCO Block Diagram in Xilinx Block set.



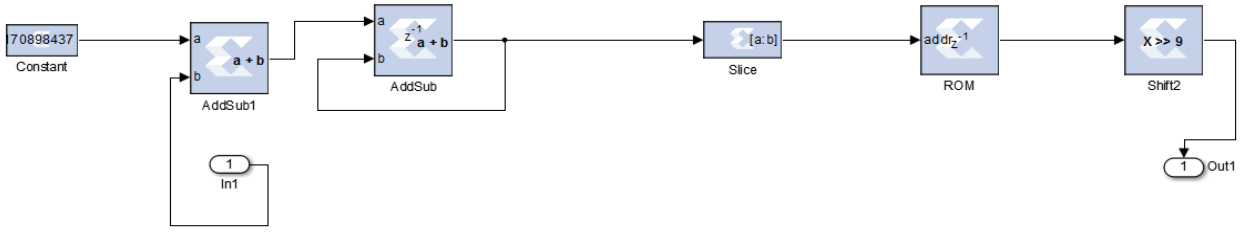


Fig 3.12 NCO in Xilinx Block set

NCO block is used at the modulation and demodulation side of the circuit. At the modulation side the input to the NCO block is the modulating signal and the output is the FM modulating signal. At the demodulation side the input to the NCO is the output from the loop filter and the output is given as feedback to the phase detector. With external signal given as input to the NCO, the frequency of the output signal varies according to the variations in the modulating signal. In the Xilinx block set in order ensure that the output of NCO varies in accordance with the external input, the modulating signal (external signal) has to be multiplied with the constant of value equal to the required frequency sensitivity which is further given as input to the NCO block.

3.2 DESCRPTION OF SUB BLOCKS

3.2.1 NUMERICALLY CONTROLLED OSCILLATOR

3.2.1.1 INTRODUCTION:

A numerically controlled oscillator (NCO) is a digital signal generator which creates a synchronous (i.e. clocked), discrete-time, discrete-valued representation of a waveform, usually sinusoidal. NCOs are often used in conjunction with a digital-to-analog converter (DAC) at the output to create a direct digital synthesizer (DDS).

Numerically controlled oscillators offer several advantages over other types of oscillators in terms of agility, accuracy, stability and reliability. NCOs are used in many communications systems including digital up/down converters used in 3G wireless and software radio systems, digital PLLs, radar systems, drivers for optical or acoustic transmissions and multilevel FSK/PSK modulators or demodulators.

The NCO is a discrete equivalent of the Voltage controlled Oscillator used in analog systems. The NCO block consists of Frequency control word, Accumulator unit, Bit extractor and Phase to amplitude converter. The frequency controlled word is determined by the desired carrier frequency and the sampling frequency. The frequency controlled word is accumulated over time using the feedback loop and the output overflows when it reaches the maximum value. The

output of the accumulator is given to the phase to amplitude converter which is a database (cosine rom) which contains mapping between phase and amplitude values over one cycle. [6]

Fig 3.13 depicts the NCO Block Diagram.

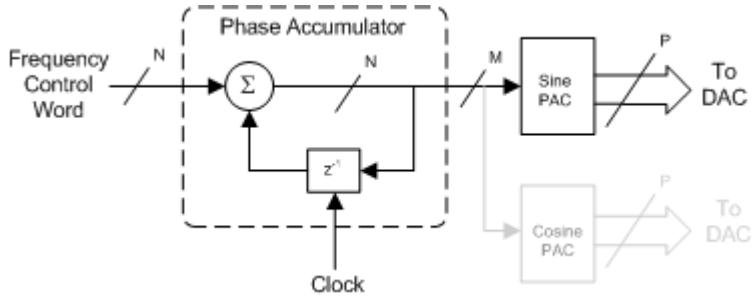


Fig 3.13 NCO Block Diagram

3.2.1.2 OPERATION:

An NCO generally consists of two parts:

- A phase accumulator (PA), which adds to the value held at its output a frequency control value at each clock sample.
- A phase-to-amplitude converter (PAC) which uses the phase accumulator output word (phase word) usually as an index into a waveform look-up table (LUT) to provide a corresponding amplitude sample.

Phase Accumulator: A binary phase accumulator consists of an N-bit binary adder and a register. Each clock cycle produces a new N-bit output consisting of the previous output obtained from the register summed with the frequency control word (FCW) which is constant for a given output frequency. The resulting output waveform is a staircase. In some configurations, the phase output is taken from the output of the register which introduces one clock cycle latency but allows the adder to operate at a higher clock rate.

Phase to Amplitude converter: The phase-amplitude converter creates the sample-domain waveform from the truncated phase output word received from the PA. The PAC can be a simple read only memory containing 2^M contiguous

samples of the desired output waveform which typically is a sinusoid. Often though, various tricks are employed to reduce the amount of memory required. This include various trigonometric expansions, trigonometric approximations and methods which take advantage of the quadrature symmetry exhibited by sinusoids. Alternatively, the PAC may consist of random access memory which can be filled as desired to create an arbitrary waveform generator.

3.2.2 PHASE LOCKED LOOP

3.2.2.1 INTRODUCTION:

A phase-locked loop or phase lock loop abbreviated as PLL is a control system that generates an output signal whose phase is related to the phase of an input signal. While there are several differing types, it is easy to initially visualize as an electronic circuit consisting of a variable frequency oscillator and a phase detector. The oscillator generates a periodic signal and the phase detector compares the phase of that signal with the phase of the input periodic signal, adjusting the oscillator to keep the phases matched. Bringing the output signal back towards the input signal for comparison is called a feedback loop since the output is "fed back" toward the input forming a loop.

Keeping the input and output phase in lock step also implies keeping the input and output frequencies the same. Consequently, in addition to synchronizing signals, a phase-locked loop can track an input frequency, or it can generate a frequency that is a multiple of the input frequency. These properties are used for computer clock synchronization, demodulation, and frequency synthesis.

Phase-locked loops are widely employed in radio, telecommunications, computers and other electronic applications. They can be used to demodulate a signal, recover a signal from a noisy communication channel, generate a stable frequency at multiples of an input frequency (frequency synthesis), or distribute precisely timed clock pulses in digital logic circuits such as microprocessors. Since a single integrated circuit can provide a complete phase-locked-loop building block, the technique is widely used in modern electronic devices, with output frequencies from a fraction of a hertz up to many giga-hertz.

3.2.2.2 TYPES OF PHASE LOCKED LOOPS:

There are several variations of PLLs. Some terms that are used are analog phase-locked loop (APLL) also referred to as a linear phase-locked loop (LPLL), digital phase-locked loop (DPLL), all digital phase-locked loop (ADPLL), and software phase-locked loop (SPLL).

Analog or linear PLL (APLL)

Phase detector is an analog multiplier. Loop filter is active or passive. Uses a Voltage-Controlled Oscillator (VCO).

Digital PLL (DPLL)

An analog PLL with a digital phase detector (such as XOR, edge-trigger JK, phase frequency detector). May have digital divider in the loop. [7]

All digital PLL (ADPLL)

Phase detector, filter and oscillator are digital. Uses a numerically controlled oscillator (NCO).

Software PLL (SPLL)

Functional blocks are implemented by software rather than specialized hardware.

Neuronal PLL (NPLL)

Phase detector, filter and oscillator are neurons or small neuronal pools. Uses a rate controlled oscillator (RCO). Used for tracking and decoding low frequency modulations (<1 kHz), such as those occurring during mammalian-like active sensing.

3.2.2.3 APPLICATIONS:

Phase-locked loops are widely used for synchronization purposes; in space communications for coherent demodulation and threshold extension, bit synchronization, and symbol synchronization. Phase-locked loops can also be used to demodulate frequency-modulated signals. In radio transmitters, a PLL is used to

synthesize new frequencies which are a multiple of a reference frequency, with the same stability as the reference frequency.

Other applications include:

- Demodulation of both FM and AM signals
- Recovery of small signals that otherwise would be lost in noise (lock-in amplifier to track the reference frequency)
- Recovery of clock timing information from a data stream such as from a disk drive
- Clock multipliers in microprocessors that allow internal processor elements to run faster than external connections, while maintaining precise timing relationships.
- DTMF decoders, modems, and other tone decoders, for remote control and telecommunications.
- DSP of video signals; Phase-locked loops are also used to synchronize phase and frequency to the input analog video signal so it can be sampled and digitally processed.
- Atomic force microscopy in tapping mode, to detect changes of the cantilever resonance frequency due to tip–surface interactions.
- DC motor drive.

3.2.2.4 ALL DIGITAL PHASE LOCKED LOOP (ADPLL)

ADPLL contains digital blocks. It uses negative feedback control loop. It takes digital signal only. The signal could be single or combination of parallel digital signals. It consists three blocks:

1. Phase Detector (PD)

2. Loop Filter (LF) and

3. Digitally Controlled Oscillator (DCO).

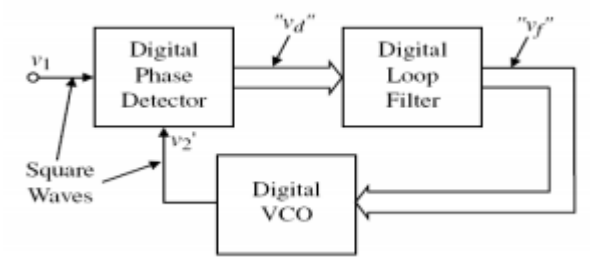


Fig. 3.14. General block diagram of ADPLL

Fig. 3.15 XOR gate phase detector

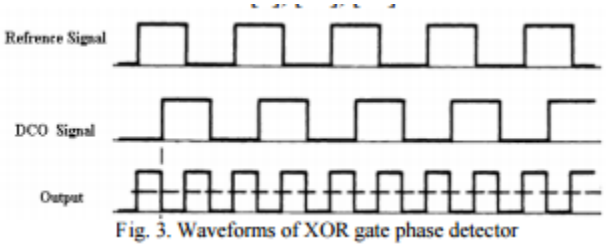


Fig. 3. Waveforms of XOR gate phase detector

Fig. 3.16 Waveforms of XOR gate phase detector

Fig 3.14 gives basic structure of an ADPLL. The aim of the ADPLL is to interlace the phase input v1 and output v2 and also the frequency. To reduce the difference among two signals PD is used. For removing noise LF is used. Finally, the digitally-controlled oscillator (DCO) gets the signals from LF and makes closer to the input signal. To realize an ADPLL, existing elements must be digital circuits. There are some advantages: No off-chip components and Insensitive to technology. [3]

Phase Detector: It is also called phase comparator. It compares between input and DCO output signal. Output depends upon the phase error. Output signal contains low frequency and higher frequency component. Some of the existing phase detectors are explained below.

1) EXOR gate phase detector: It uses an EXOR logic gate. It compares the reference and DCO signal. Fig 3.15 XOR gate phase detector Disadvantages of this are it has phase limitation [-90, +90] degrees and it does not sense edges signal edges. Fig 3.16 shows the “locked” state. Fig 3.16 Waveforms of XOR gate phase detector

2) Edge triggered JK flip-flop phase detector contains a JK FF. A phase limitation of this is -180 degrees to +180 degrees.

3) Flip-flop counter phase detector: The phase detector contains a counter and a FF. Flip-flop Counter phase detector compares reference and the DCO output signal. In this case FF input S takes input signal and R takes DCO output signal. Output of FF is high when there is error among S and R inputs. Q enables the counter. FF input S resets counter. Output of counter depends upon the phase error. Clock frequency of counter is very high it is M times multiple of input signal is large positive integer.

Loop Filter: It is nothing but an integrator. UP/down counter loop filter is simplest loop filter. It is always operate in conjunction with EXOR or JK FF phase detector. For getting clock and direction signal a pulse forming circuit is used. Counter is incremented on each UP pulses and it is decremented on each down signals .So counter adds both pulses .So it works like an integrator.

Digitally Controlled Oscillators: Digitally Controlled oscillators are nothing but a modified oscillator. Depending upon output of the loop filter they change their frequency. Divide by N counter DCO is a simple ÷N counter works as DCO. High frequency signal operates at very high frequency. Divide by N counter produces N bit parallel output. Drawback of it is we can't design jitter.

3.2.3 LOOP FILTER

3.2.3.1 INTRODUCTION:

The design of the PLL, loop filter is crucial to the operation of the whole phase locked loop. The choice of the circuit values here is usually a very carefully balanced compromise between a number of conflicting requirements.

The PLL filter is needed to remove any unwanted high frequency components which might pass out of the phase detector and appear in the NCO tune line. They would then appear on the output of the Voltage Controlled Oscillator, NCO, as spurious signals. To show how this happens take the case when a mixer is used as a phase detector. When the loop is in lock the mixer will produce two signals: the sum and difference frequencies. As the two signals entering the phase detector have the same frequency the difference frequency is

zero and a DC voltage is produced proportional to the phase difference as expected. The sum frequency is also produced and this will fall at a point equal to twice the frequency of the reference. If this signal is not attenuated it will reach the control voltage input to the NCO and give rise to spurious signals.

When other types of phase detector are used similar spurious signals can be produced and the filter is needed to remove them.

The filter also affects the ability of the loop to change frequencies quickly. If the filter has a very low cut-off frequency then the changes in tune voltage will only take place slowly, and the NCO will not be able to change its frequency as fast. This is because a filter with a low cut-off frequency will only let low frequencies through and these correspond to slow changes in voltage level.

Conversely a filter with a higher cut-off frequency will enable the changes to happen faster. However when using filters with high cut-off frequencies, care must be taken to ensure that unwanted frequencies are not passed along the tune line with the result those spurious signals are generated.

The loop filter also governs the stability of the loop. If the filter is not designed correctly then oscillations can build up around the loop, and large signals will appear on the tune line. This will result in the NCO being forced to sweep over wide bands of frequencies. The proper design of the filter will ensure that this cannot happen under any circumstances.

3.2.3.2 IIR FILTER:

Infinite Impulse Response (IIR) is a property applying to many linear time-invariant systems like the electronic and digital filters. IIR filters are distinguished by having impulse response which does not become exactly zero past a certain point, but continues indefinitely which is in contrast to Finite Impulse Response (FIR) in which the impulse response h(t) does become exactly zero at times $t > T$ for some finite T, thus being of finite duration. The presence of feedback in the topology of a discrete-time filter generally creates an IIR response. The z domain transfer function of an IIR filter contains a non-trivial denominator,

describing those feedback terms. The transfer functions pertaining to IIR filters have been extensively optimized for their amplitude and phase characteristics. [3]

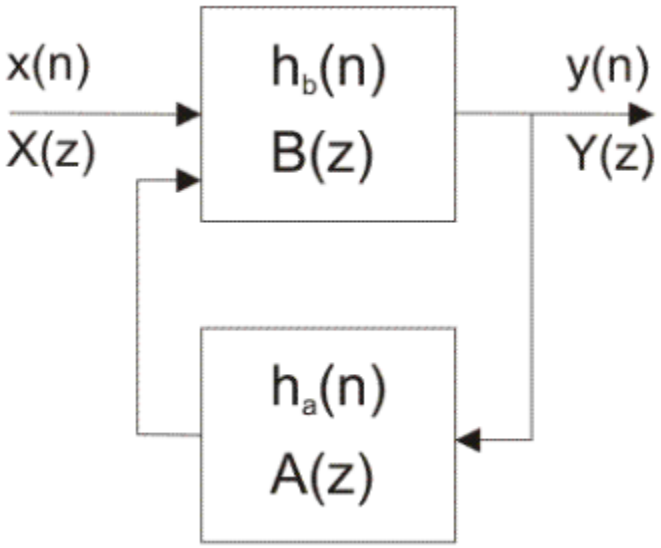


Fig 3.17 IIR Filter

Compared to FIR Filter, IIR Filter has the following advantages:

- Transfer function of IIR filter will have both zeros and poles.
- Less memory and the computational complexity is less.
- It uses current input sample value, past input and output samples to obtain current output sample value.

3.2.3.3 BUTTERWORTH FILTER:

The Butterworth filter is a type of signal processing filter designed to have as flat frequency response as possible in the pass band. It is also referred to as a **maximally flat magnitude filter**. The frequency response of the Butterworth filter is maximally flat (i.e. has no ripples) in the passband and rolls off towards zero in the stopband. When viewed on a logarithmic Bode plot, the response slopes off linearly towards negative infinity. Compared with a Chebyshev Type I/Type II filter or an elliptic filter, the Butterworth filter has a slower roll-off, and thus will require a higher order to implement a particular stopband specification, but

Butterworth filters have a more linear phase response in the pass-band than Chebyshev Type I/Type II and elliptic filters can achieve.

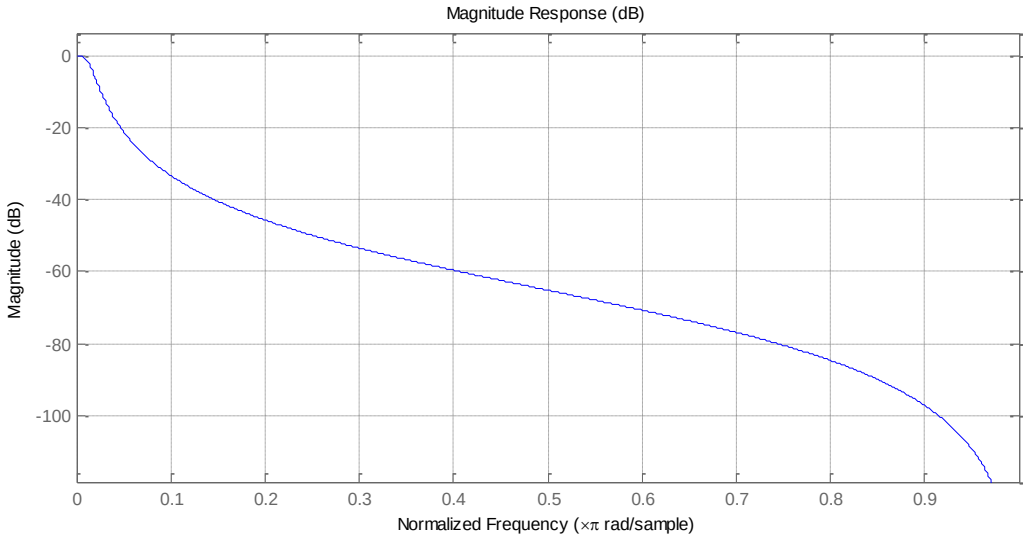


Fig 3.18 Magnitude Response of the Filter

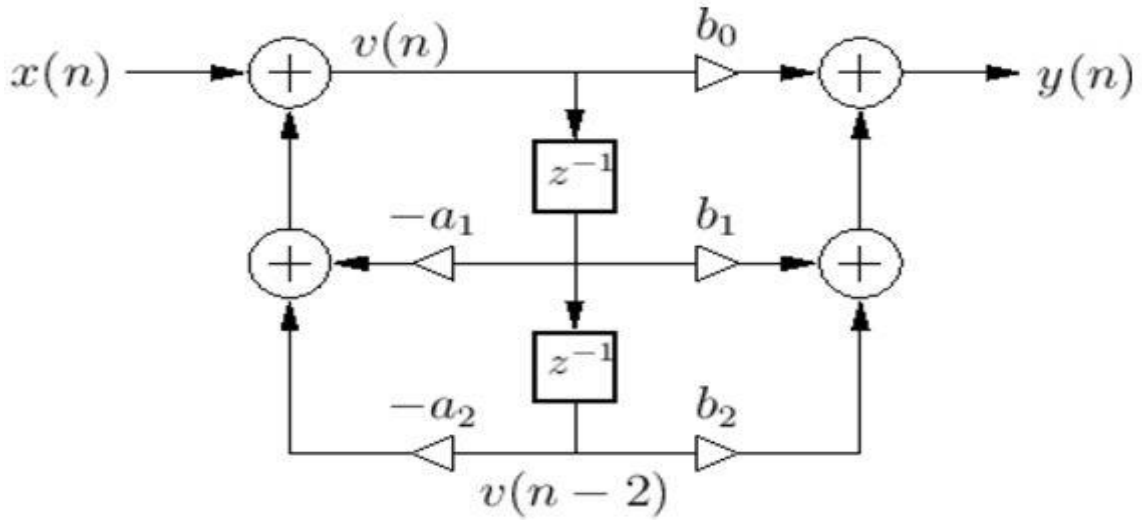
Comparison with other Linear Filters:

- Monotonic amplitude response in both passband and stopband.
- Quick roll-off around the cut-off frequency, which improves with increasing order.
- Considerable overshoot and ringing in step response, which worsens with increasing order.
- Slightly non-linear phase response.
- Group delay largely frequency-dependent.

3.2.3.4 DIRECT FORM –II STRUCTURE:

The direct form II only needs N delay units, where N is the order of the filter potentially half as much as direct form I. The structure is obtained by reversing the order of the numerator and denominator sections of Direct Form I, since they are in fact two linear systems, and the commutative property applies. It can be regarded

as a two-pole filter section followed by a two-zero filter section. It is canonical with respect to delay. This happens because delay elements associated with the two-pole and two-zero sections are shared. The poles and zeros are sensitive to round off errors in the coefficients ‘a’ and ‘b’.



$$v(n) = x(n) - a_1 v(n-1) - a_2 v(n-2)$$
$$y(n) = b_0 v(n) + b_1 v(n-1) + b_2 v(n-2)$$

Fig 3.19 Direct Form-II Structure

CHAPTER 4

HARDWARE COMPONENTS

4.1 ARDUINO DUE

4.1.1 INTRODUCTION:

The Arduino Due is a microcontroller board based on the Atmel SAM3X8E ARM Cortex-M3 CPU. It is the first Arduino board based on a 32-bit ARM core microcontroller. It has 54 digital input/output pins (of which 12 can be used as PWM outputs), 12 analog inputs, 4 UARTs (hardware serial ports), a 84 MHz clock, an USB OTG capable connection, 2 DAC (digital to analog), 2 TWI (SDA and SCL pins), a power jack, an SPI header, a JTAG header, a reset button and an erase button. It runs at 3.3V. [8]

Fig 4.1 ARDUINO DUE BOARD

The Arduino Due has two ports namely programming port and native port.The Programming port is connected to an ATmega16U2, which provides a virtual COM port to software on a connected computer. The 16U2 is also

connected to the SAM3X hardware UART. Serial on pins RX0 and TX0 provides Serial-to-USB communication for programming the board through the Atmega16U2 microcontroller. The Arduino software includes a serial monitor which allows simple textual data to be sent to and from the board. The RX and TX LEDs on the board will flash when data is being transmitted via the Atmega16U2 chip and USB connection to the computer (but not for serial communication on pins 0 and 1).

The Native USB port is connected to the SAM3X. It allows for serial (CDC) communication over USB. This provides a serial connection to the Serial Monitor or other applications on your computer. It also enables the Due to emulate a USB mouse or keyboard to an attached computer. [8]

4.1.2 KEY SPECIFICATIONS:

Voltage:

The microcontroller mounted on the Arduino Due runs at 3.3V, this means that you can power your sensors and drive your actuators only with 3.3V. Connecting higher voltages, like the 5V commonly used with the other Arduino boards will damage the Due. The board can take power from the USB connectors or the DC plug. If using the DC connector, supply a voltage between 7V and 12V. The Arduino Due has an efficient switching voltage regulator, compliant with the USB host specification. If the Native USB port is used as host by attaching a USB device to the mirco-A usb connector, the board will provide the power to the device. When the board is used as an usb host, external power from the DC connector is required.

Serial ports on the Due:

The Arduino Due has two USB ports available. The Native USB port (which supports CDC serial communication using the SerialUSB object) is connected directly to the SAM3X MCU. The other USB port is the Programming port. It is connected to an ATMEL 16U2 which acts as a USB-to-Serial converter. This Programming port is the default for uploading sketches and communicating with the Arduino. The USB-to-serial converter of the Programming port is connected to the first UART of the SAM3X. It’s possible to communicate over this port using

the "Serial" object in the Arduino programming language. The USB connector of the Native port is directly connected to the USB host pins of the SAM3X. Using the Native port enables you to use the Due as a client USB peripheral (acting as a mouse or a keyboard connected to the computer) or as a USB host device so that devices can be connected to the Due (like a mouse, keyboard, or an Android phone). This port can also be used as a virtual serial port using the "SerialUSB" object in the Arduino programming language.

Automatic (Software) Reset:

The SAM3X microcontroller differs from AVR microcontrollers because the flash memory needs to be erased before being re-programmed. A manual procedure would involve holding the erase button for a second, pressing the upload button in the IDE, then the reset button. Because a manual erase-flash procedure is repetitive, this is managed automatically by both USB ports, in two different ways:

Native port: Opening and closing the ''Native' port at the baud rate of 1200bps triggers a "soft erase" procedure: the flash memory is erased and the board is restarted with the bootloader. If, for some reason, the MCU were to crash during this process, it is likely that the soft erase procedure wouldn't work as it's done in software by the MCU itself. Opening and closing the Native port at a baud rate other than 1200bps will not reset the SAM3X. To use the serial monitor, and see what your sketch does from the beginning, you'll need to add few lines of code inside the setup(). This will ensure the SAM3X will wait for the SerialUSB port to open before executing the sketch. Pressing the Reset button on the Due causes the SAM3X reset as well as the USB communication. This interruption means that if the serial monitor is open, it's necessary to close and reopen it to restart the communication.

Programming port: The Programming port uses a USB-to-serial chip connected to the first UART of the MCU (RX0 and TX0). The USB to-serial chip has two pins connected to the Reset and Erase pins of the SAM3X. When you open this serial port, the USB-to-Serial activates the Erase and Reset sequence before it begins communicating with the UART of the SAM3X. This procedure is much more reliable and should work even if the main MCU has crashed. To

communicate serially with the Programming port, use the "Serial" object in the IDE. All existing sketches that use serial communication based on the Uno board should work similarly. The Programming port behaves like the Uno's serial port in that the USB-to-Serial chip resets the board each time you open the serial monitor (or any other serial communication). Pressing the Reset button while communicating over the Programming port doesn't close a USB connection with the computer because only the SAM3X is reset.

USB Host:

The Due has the ability to act as a USB host for peripherals connected to the SerialUSB port. For additional information and examples, see the USB host reference page. When using the Due as a host, it will be providing power to the attached device. It is strongly recommended to use the DC power connector when acting as a host.

ADC and PWM resolutions:

The Due has the ability to change its default analog read and write resolutions (10-bits and 8-bits, respectively). It can support up to 12-bit ADC and PWM resolutions. See the analog write resolution and analog read resolution pages for information.

4.1.3 ADVANTAGES:

- Two digital-to-analog converters (DACs), which allow the board to output true analog values (instead of PWM). This means you can play audio out it
- USB on-the-go (OTG) capability allows the Due to act as both a USB device and a host. So you can hook up other USB devices – like flash drives, WiFi modules, or phones – to the Due.
- Direct Memory Access (DMA) allows the microcontroller to offload memory-access tasks, so it can perform other operations at the same time.

4.2 VIRTEX-5 FPGA

4.2.1 INTRODUCTION:

Virtex is the flagship family of FPGA products developed by Xilinx. Other current product lines include Kintex (mid-range) and Artix (low-cost), each including configurations and models optimized for different applications. In addition, Xilinx offers the Spartan low-cost series, which continues to be updated and is nearing production utilizing the same underlying architecture and process node as the larger 7-series devices.

Virtex FPGAs are typically programmed in hardware description languages such as VHDL or Verilog, using the Xilinx ISE or Vivado Design Suite computer software.

The Virtex-5 LX and the LXT are intended for logic-intensive applications, and the Virtex-5 SXT is for DSP applications. With the Virtex-5, Xilinx changed the logic fabric from four-input LUTs to six-input LUTs. With the increasing complexity of combinational logic functions required by SoC designs, the percentage of combinational paths requiring multiple four-input LUTs had become a performance and routing bottleneck. The new six-input LUT represented a tradeoff between better handling of increasingly complex combinational functions, at the expense of a reduction in the absolute number of LUTs per device. The Virtex-5 series is a 65 nm design fabricated in 1.0 V, triple-oxide process technology.

Virtex-5 FPGAs are available in -3, -2, -1 speed grades, with -3 having the highest performance. Virtex-5 FPGA DC and AC characteristics are specified for both commercial and industrial grades. Except the operating temperature range or unless otherwise noted, all the DC and AC electrical parameters are the same for a particular speed grade (that is, the timing characteristics of a -1 speed grade industrial device are the same as for a -1 speed grade commercial device). However, only selected speed grades and/or devices might be available in the industrial range. [9]

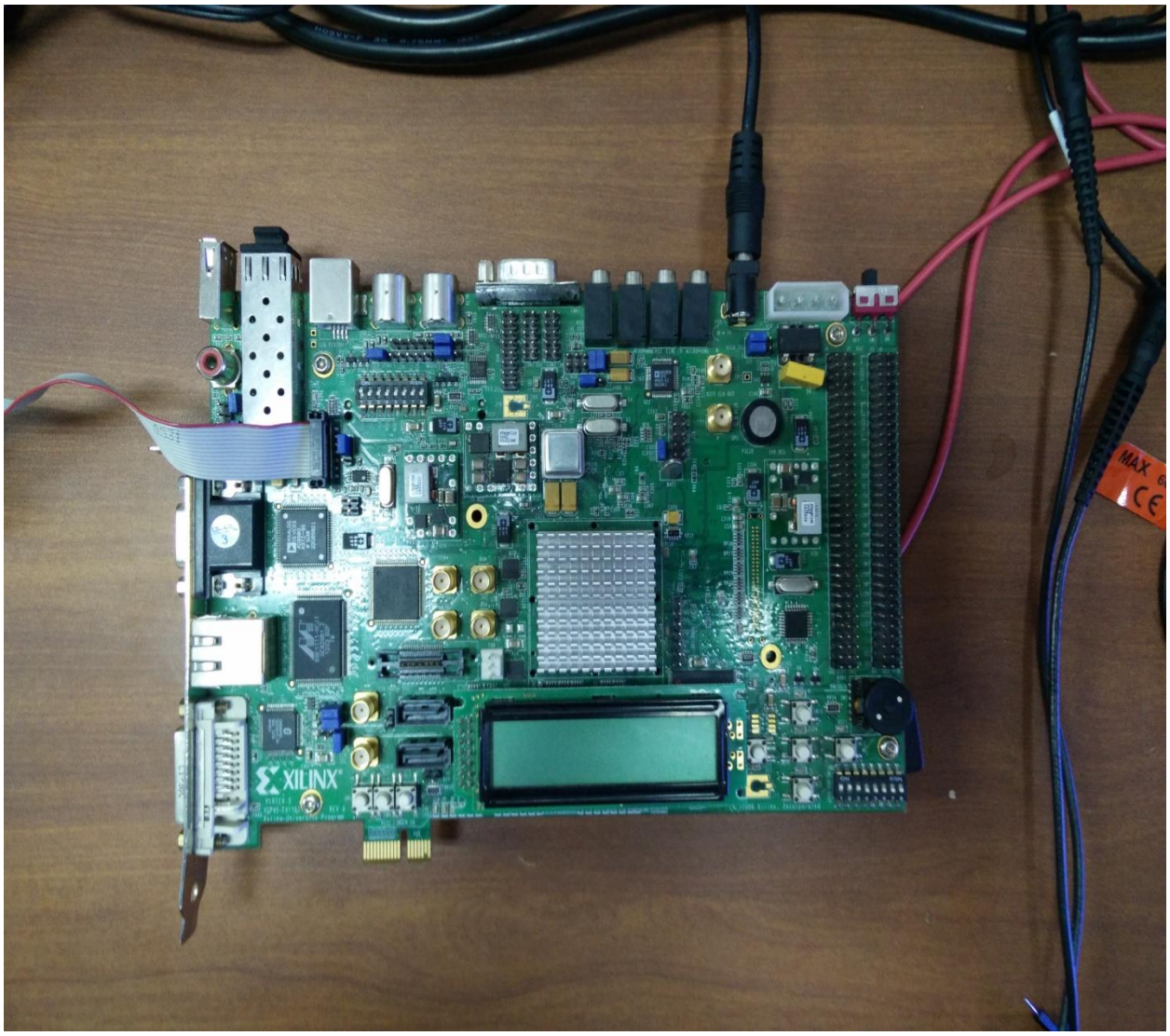


Fig 4.2 FPGA VIRTEX 5 BOARD

4.2.2 KEY SPECIFICATIONS:

• Xilinx Virtex-5 FPGA
  - XC5VLX50T-1FFG1136 (ML505)
  - XC5VSX50T-1FFG1136 (ML506)
  - XC5VFX70T-1FFG1136 (ML507)

• Two Xilinx XCF32P Platform Flash PROMs (32 Mb each) for storing large device configurations.

• Xilinx System ACE™ Compact Flash configuration controller with Type I Compact Flash connector

• Xilinx XC95144XL CPLD for glue logic

• 64-bit wide, 256-MB DDR2 small outline DIMM (SODIMM), compatible with EDK supported IP and software drivers
• Clocking
  - Programmable system clock generator chip
  - One open 3.3V clock oscillator socket
  - External clocking via SMAs (two differential pairs)
• General purpose DIP switches (8), LEDs (8), pushbuttons, and rotary encoder
• Expansion header with 32 single-ended I/O, 16 LVDS-capable differential pairs, and 14 spare I/Os shared with buttons and LEDs, power, JTAG chain expansion capability, and IIC bus expansion
• Stereo AC97 audio codec with line-in, line-out, 50-mW headphone, microphone-in jacks, SPDIF digital audio jacks, and piezo audio transducer.
• RS-232 serial port, DB9 and header for second serial port
• 16-character x 2-line LCD display
• One 8-Kb IIC EEPROM and other IIC capable devices
• PS/2 mouse and keyboard connectors
• Video input/output
  - Video input (VGA)
  - Video output DVI connector (VGA supported with included adapter)
• ZBT synchronous SRAM, 9 Mb on 32-bit data bus with four parity bits
• Intel P30 StrataFlash linear flash chip (32 MB)
• Serial Peripheral Interface (SPI) flash (2 MB)
• 10/100/1000 tri-speed Ethernet PHY transceiver and RJ-45 with support for MII, GMII, RGMII, and SGMII Ethernet PHY interfaces
• USB interface chip with host and peripheral ports
• Rechargeable lithium battery to hold FPGA encryption keys
• JTAG configuration port for use with Parallel Cable III, Parallel Cable IV, or Platform
USB download cable
• On-board power supplies for all necessary voltages
• Temperature and voltage monitoring chip with fan controller
• 5V @ 6A AC adapter
• Power indicator LED
• MII, GMII, RGMII, and SGMII Ethernet PHY Interfaces

• GTP/GTX: SFP (1000Base-X)
• GTP/GTX: SMA (RX and TX Differential Pairs)
• GTP/GTX: SGMII
• GTP/GTX: PCI Express® (PCIe™) edge connector (x1 Endpoint)
• GTP/GTX: SATA (dual host connections) with loopback cable
• GTP/GTX: Clock synthesis ICs
• Mictor trace port
• BDM debug port
• Soft touch port
• System monitor

4.2.3 APPLICATIONS:

1. Xilinx speeds the development of embedded vision, applications in markets where systems must be highly differentiated, extremely responsive, and able to immediately adapt to the latest algorithms and image sensors.

2. Flexible, standards-based solution that combines software programmability, real-time processing, hardware optimization and any-to-any connectivity with the security and safety needed for Industrial IoT systems.

3. It is used in Real-time image analytics which includes object detection, recognition, classification, and tracking enable applications such as lane departure warning and pedestrian detection.

4.3 DIGITAL TO ANALOG CONVERTER

Signals coming from the FPGA have to be converted to appropriate form, which is done by digital to analog converter (DAC)

The DAC chosen (Intersil CA3338EZ) has following features:

- 10 MHZ maximum operation – output unipolar (~1V, depends on o/p resistance)
- No of bits – 8
- Digital Input – unsigned (range 0 to 255)

The DAC required for implementation is shown in Fig 4.3.

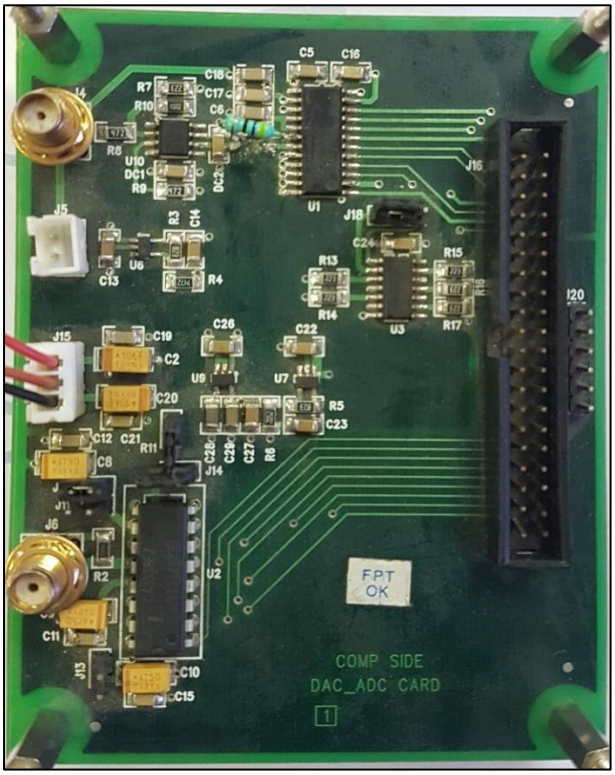


Fig 4.3 DAC/ADC CARD

CHAPTER 5

SIMULATION RESULTS

5.1 SINE WAVE INPUT

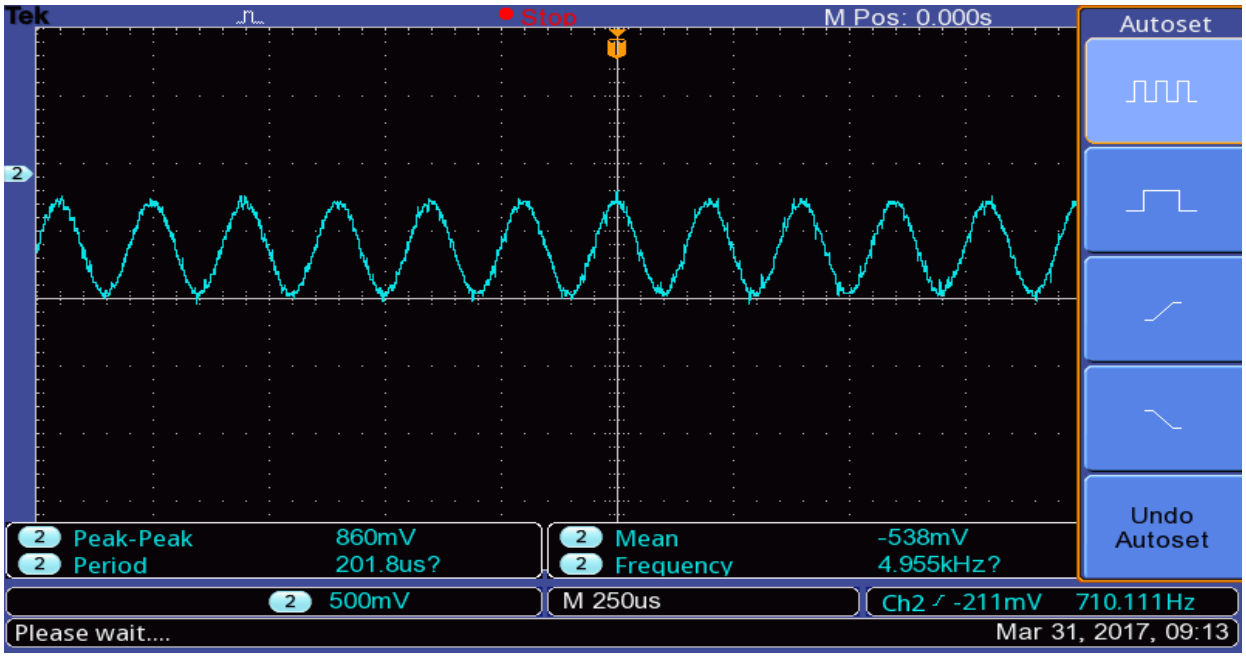


Fig 5.1 SINE INPUT

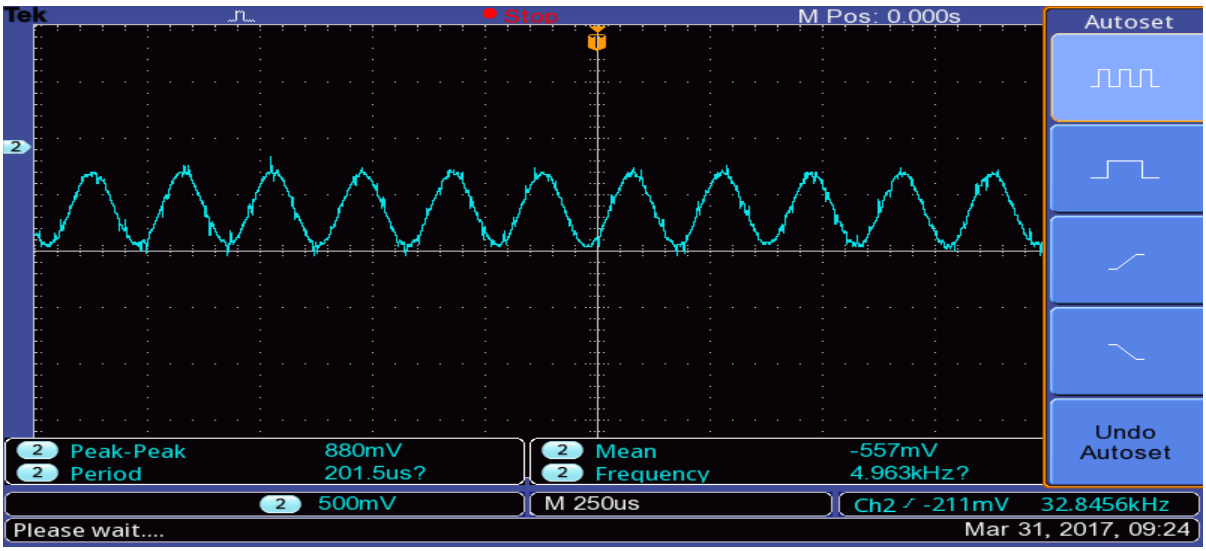


Fig 5.2 FILTER OUTPUT

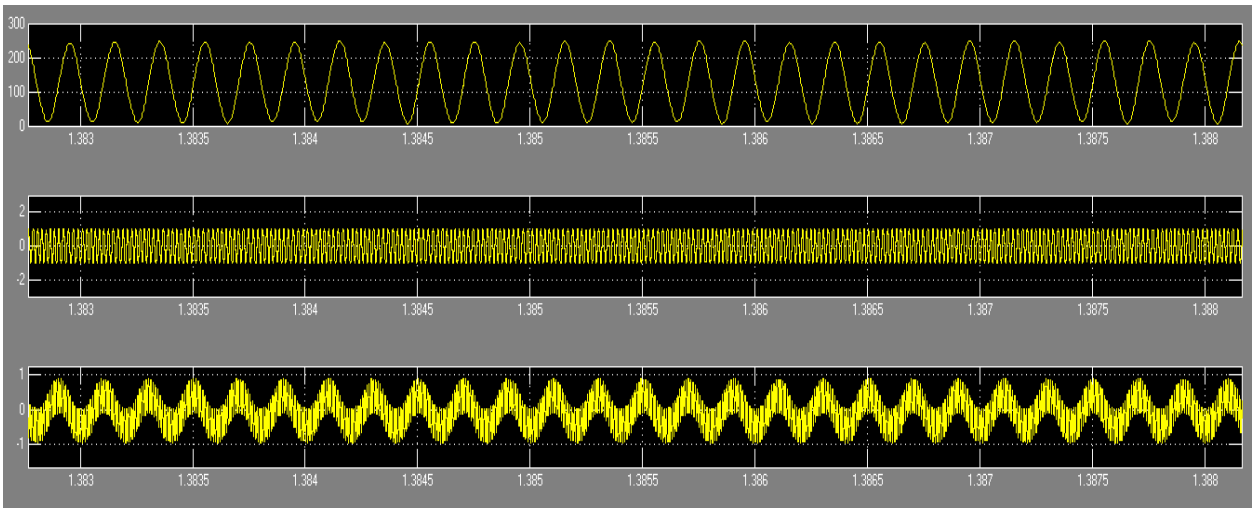


Fig 5.3 PHASE DETECTOR OUTPUT WITH SINE INPUT AND FM WAVE

VARIABLE CARRIER AND SENSITIVITY

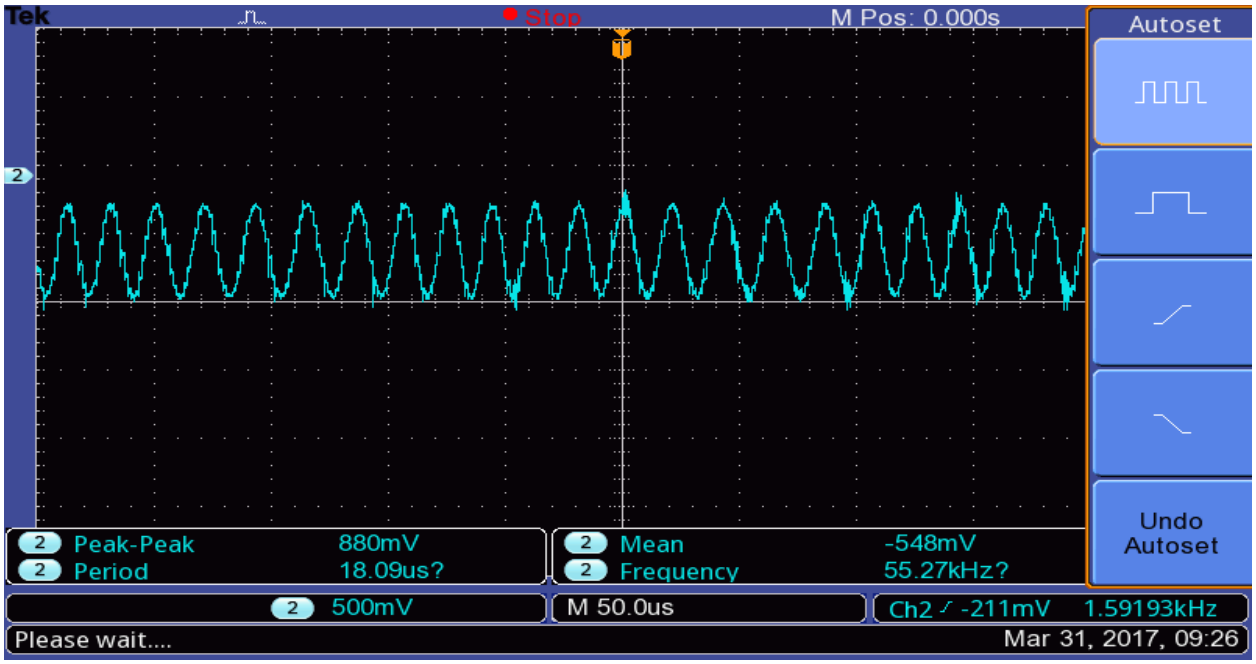


Fig 5.4 50 kHz CARRIER AND 1000 Hz/V SENSITIVITY

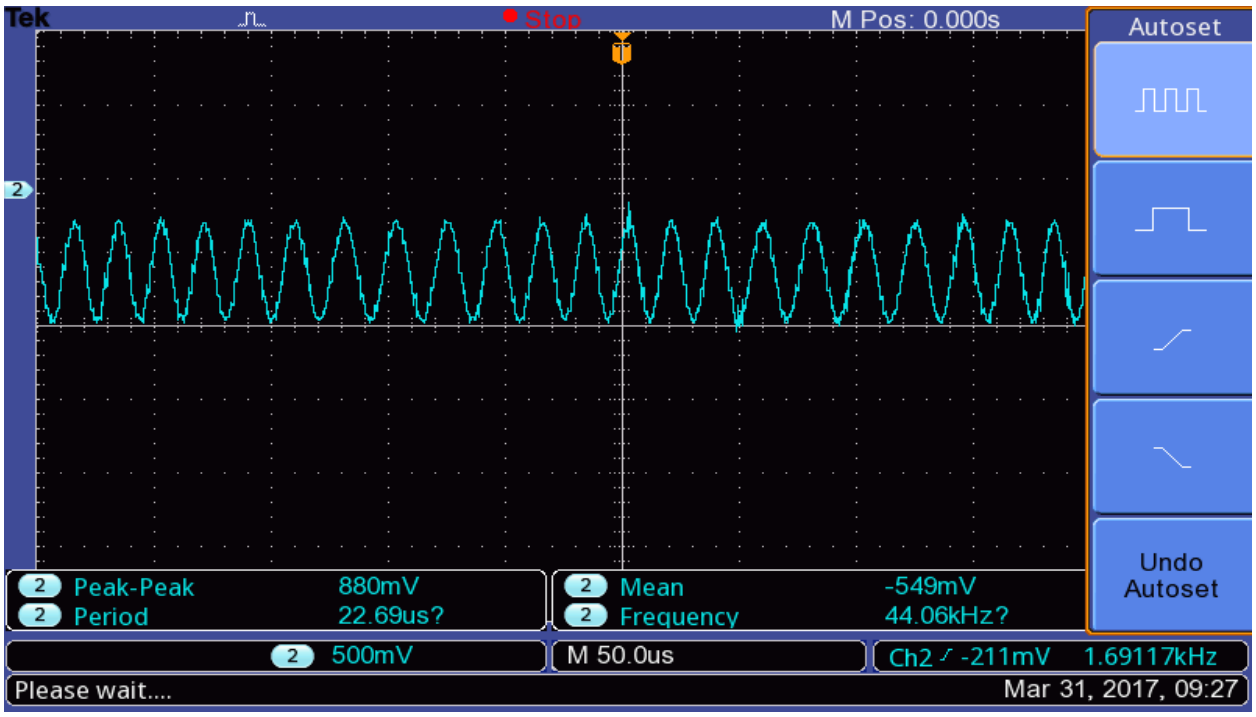


Fig 5.5 50 kHz CARRIER AND 1000 Hz/V SENSITIVITY

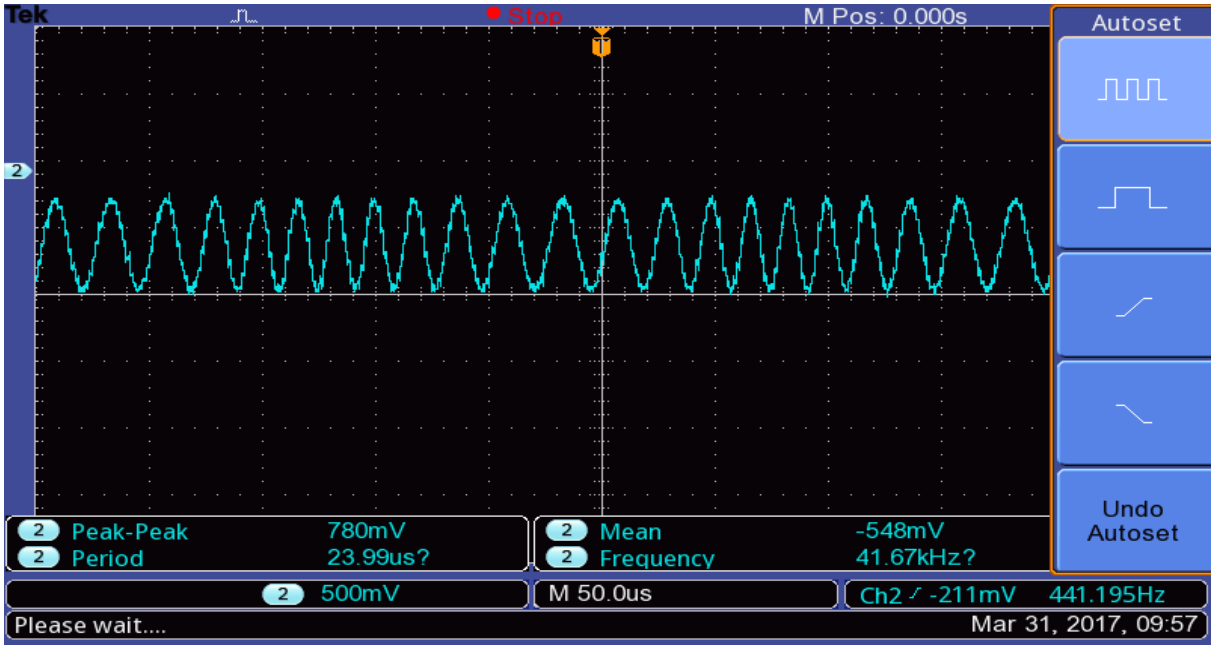


Fig 5.6 50 kHz CARRIER AND 2000 Hz/V SENSITIVITY

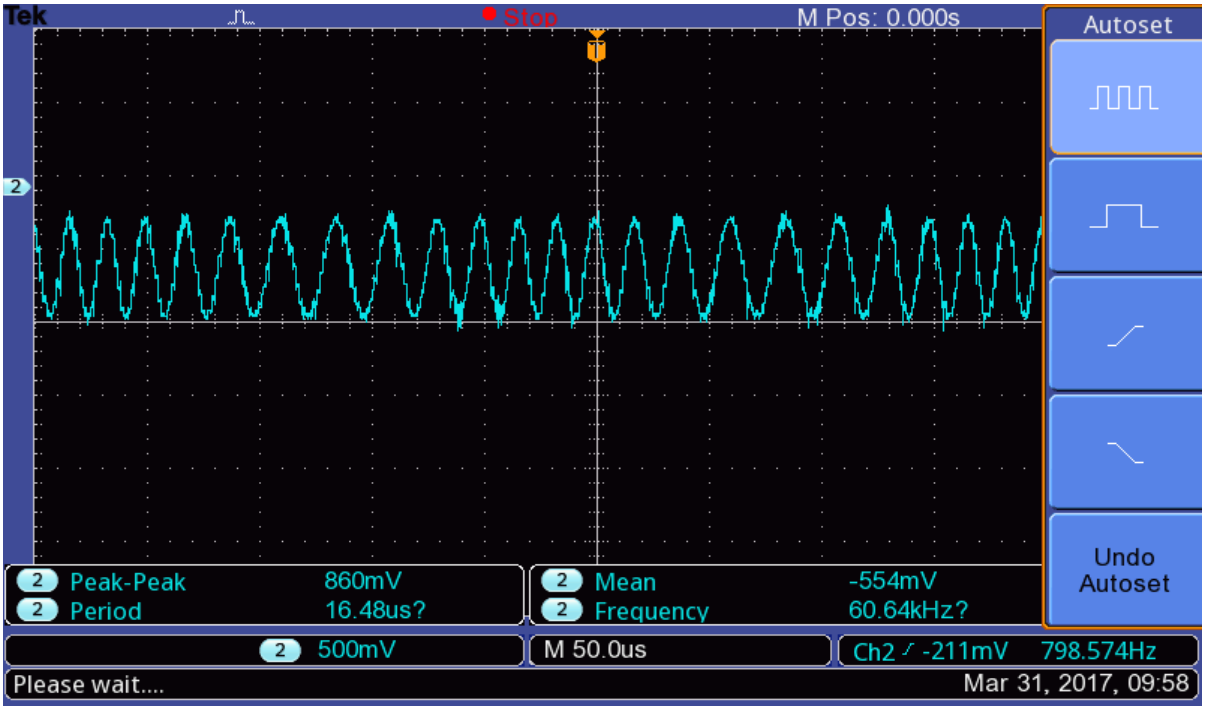


Fig 5.7 50 kHz CARRIER AND 2000 Hz/V SENSITIVITY

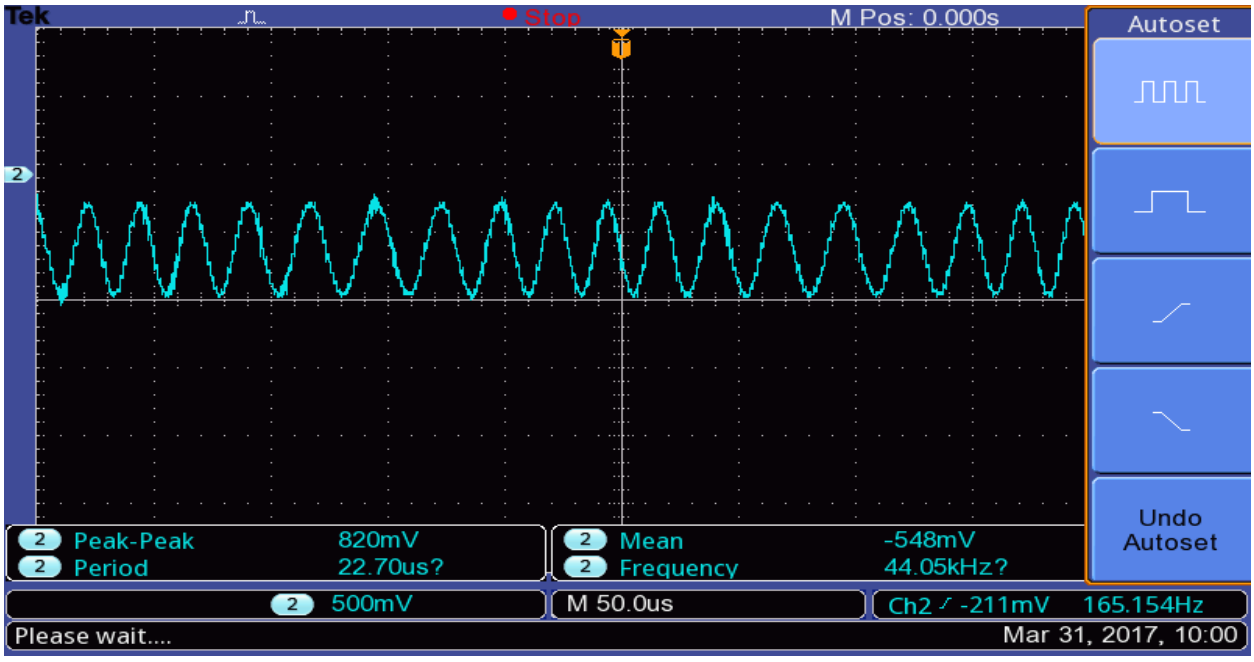


Fig 5.8 40 kHz CARRIER AND 1000 Hz/V SENSITIVITY

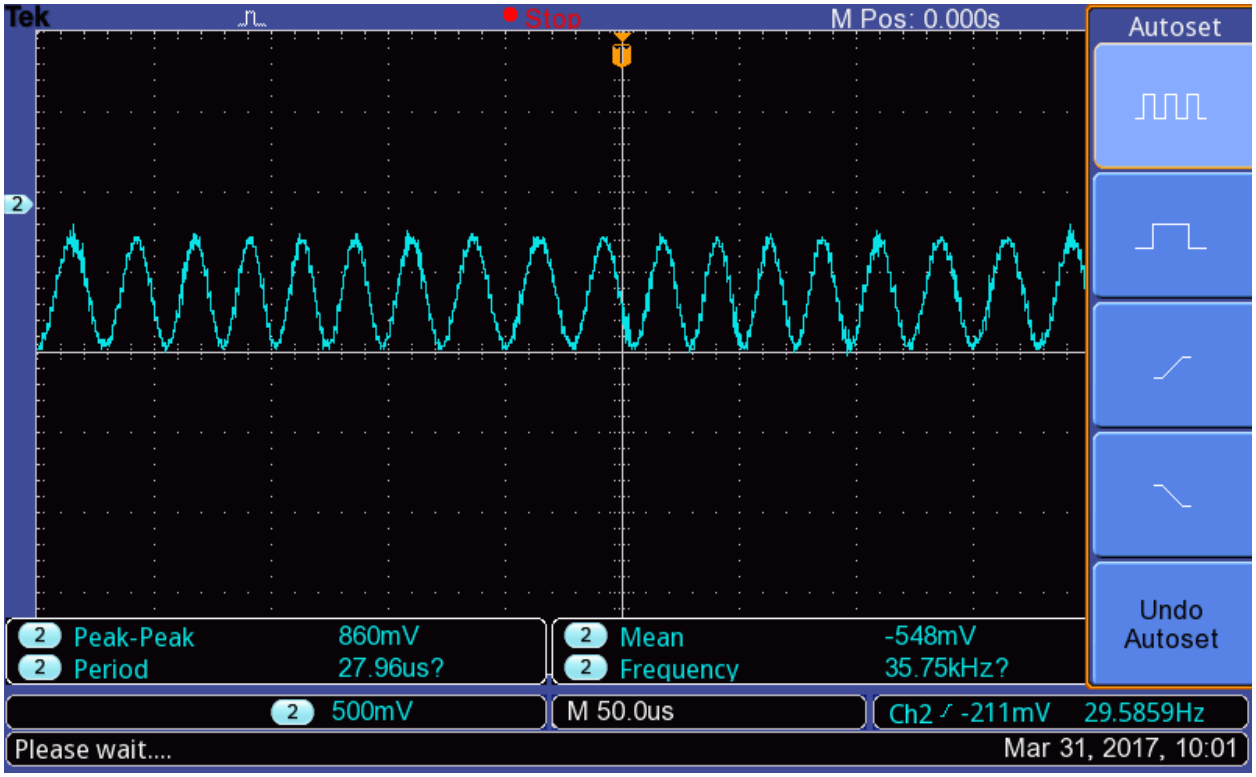


Fig 5.9 40 kHz CARRIER AND 1000 Hz/V SENSITIVITY

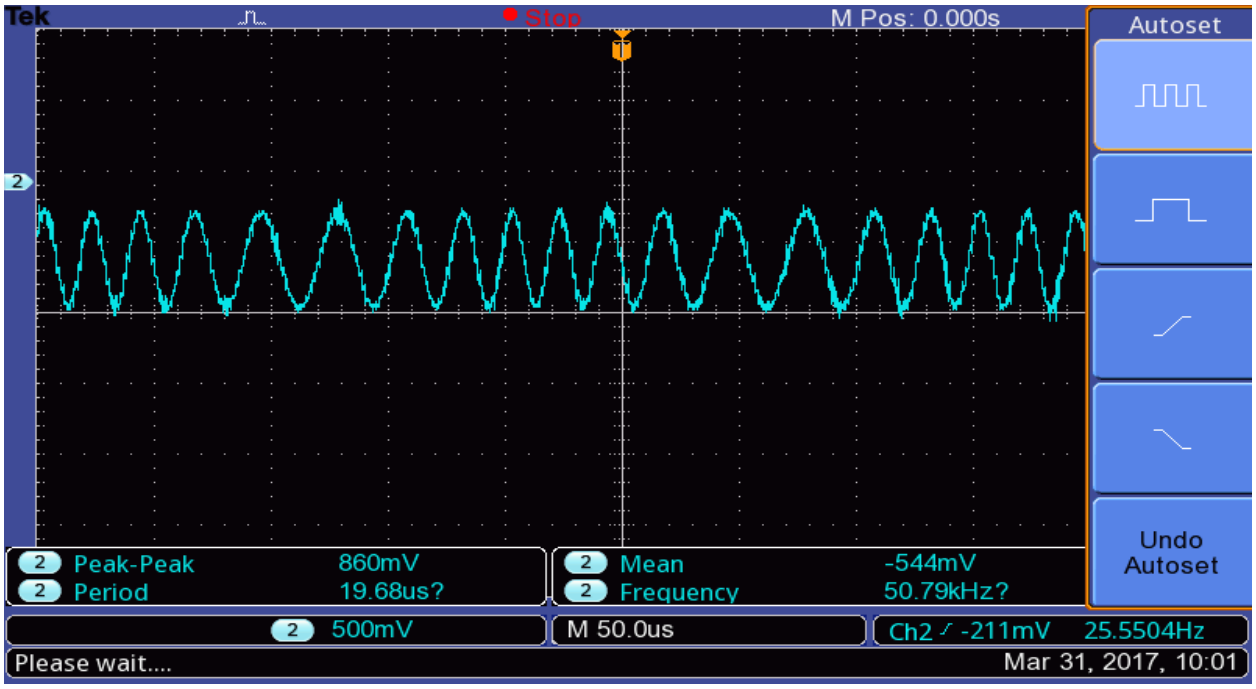


Fig 5.10 40 kHz CARRIER AND 2000 Hz/V SENSITIVITY

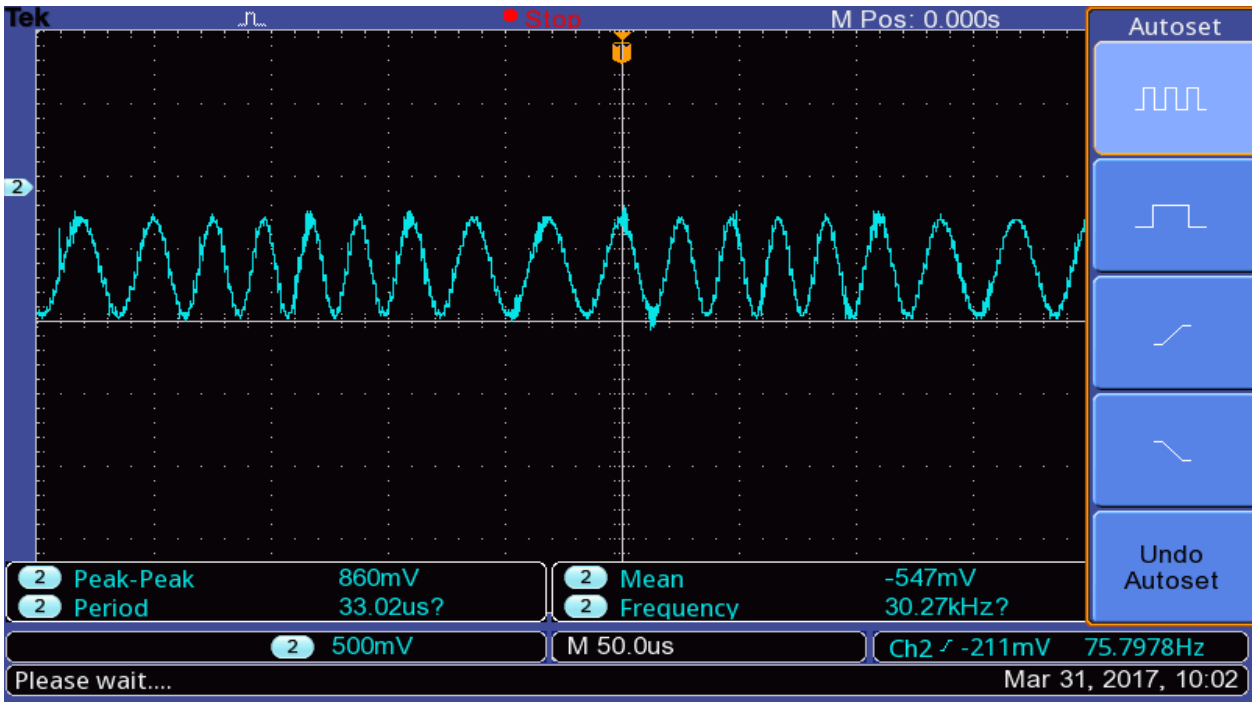


Fig 5.11 40 kHz CARRIER AND 2000 Hz/V SENSITIVITY

5.2 AUDIO INPUT-1

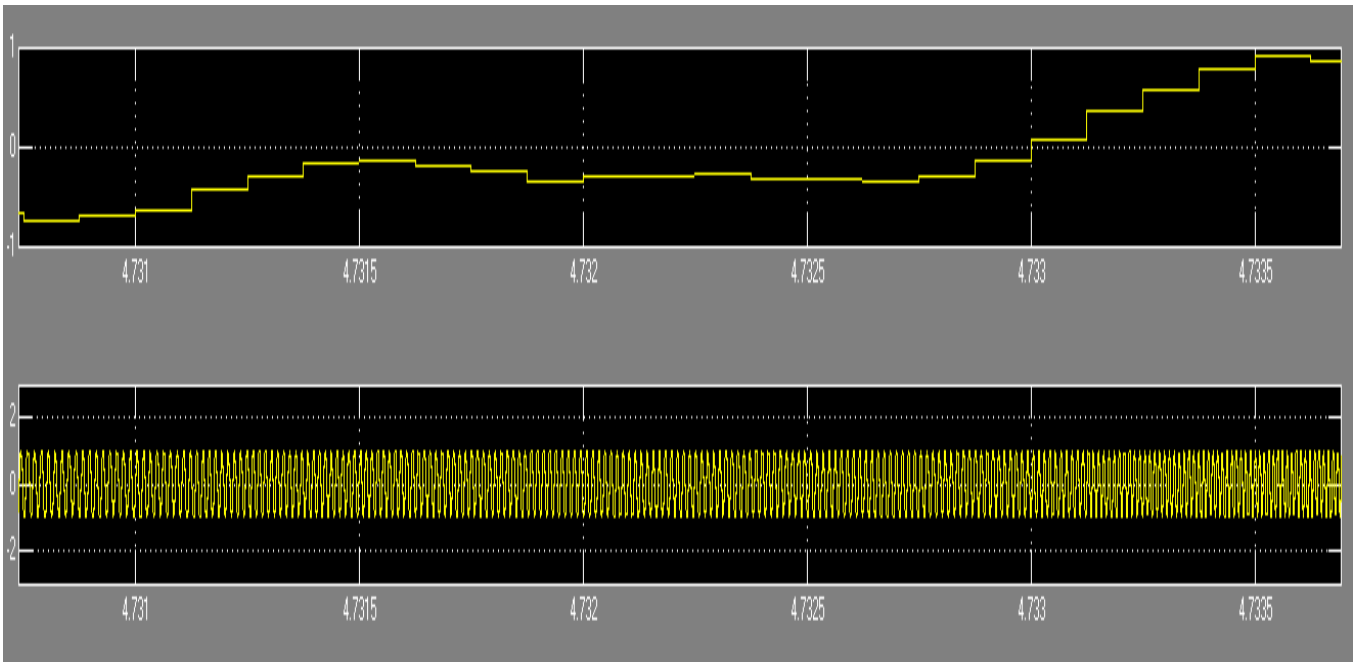


Fig 5.12 FM Signal (Audio Input)

VARIATION IN SENSITIVITY

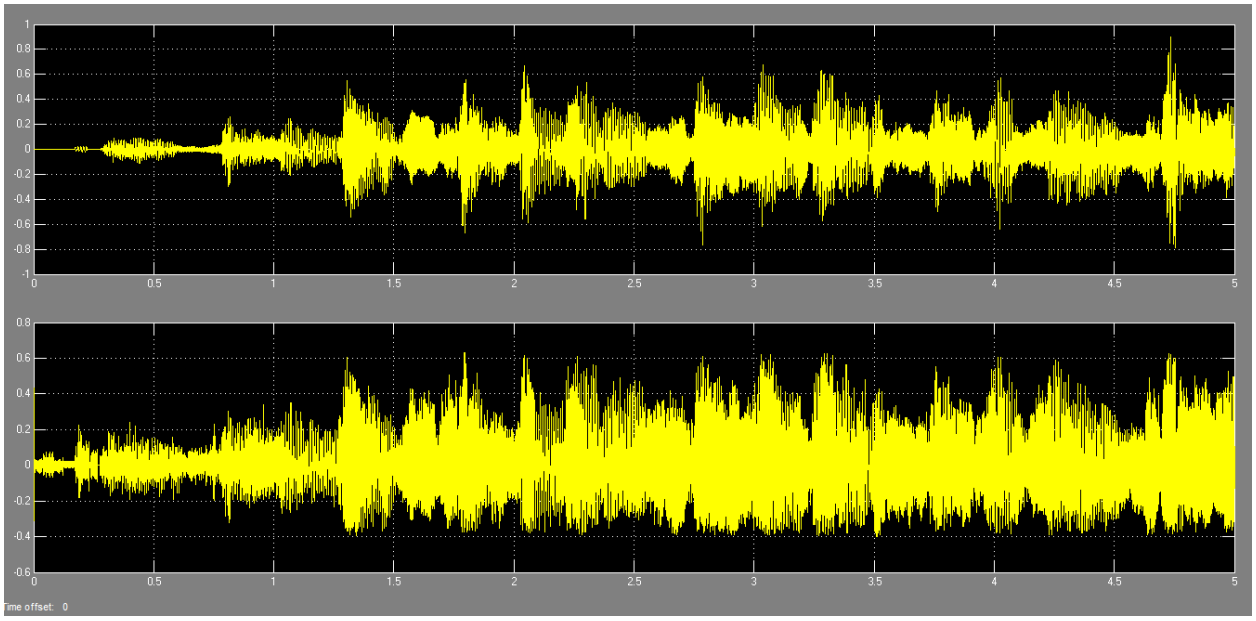


Fig 5.13 **10 kHz/V**

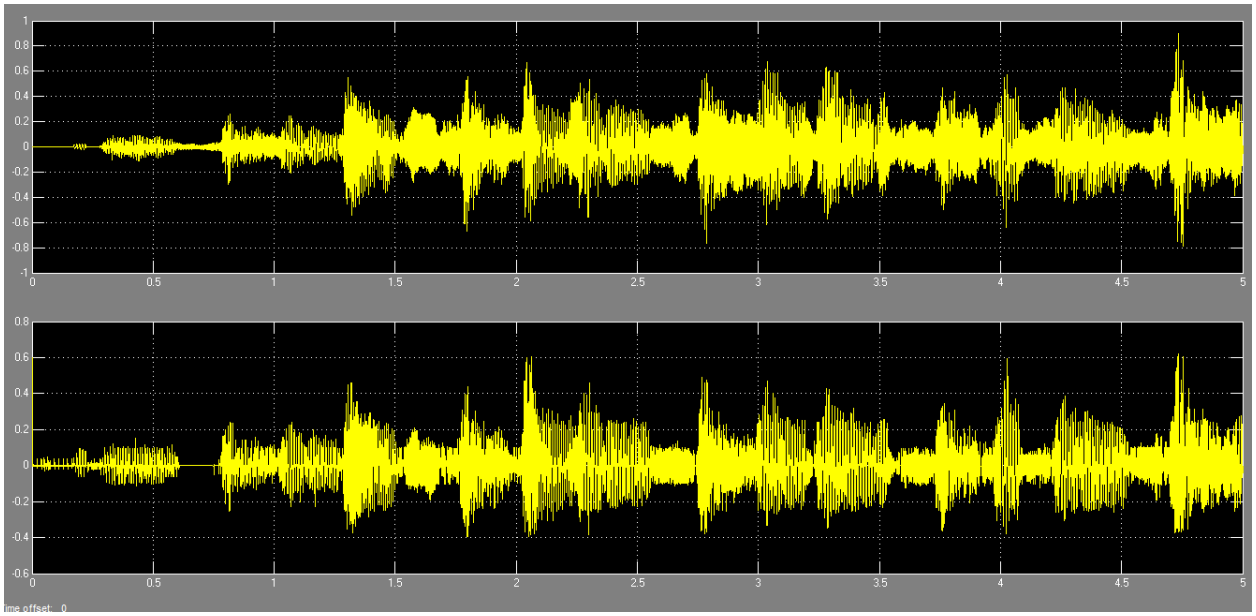


Fig 5.14 **1 kHz/V**

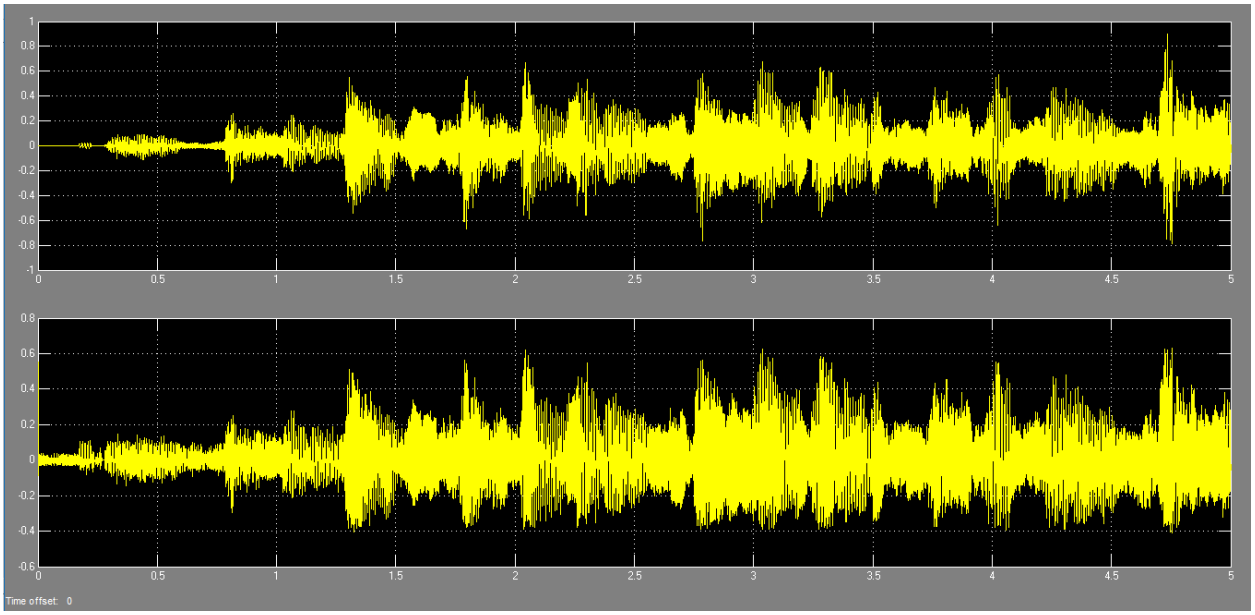


Fig 5.15 **5 kHz/V**

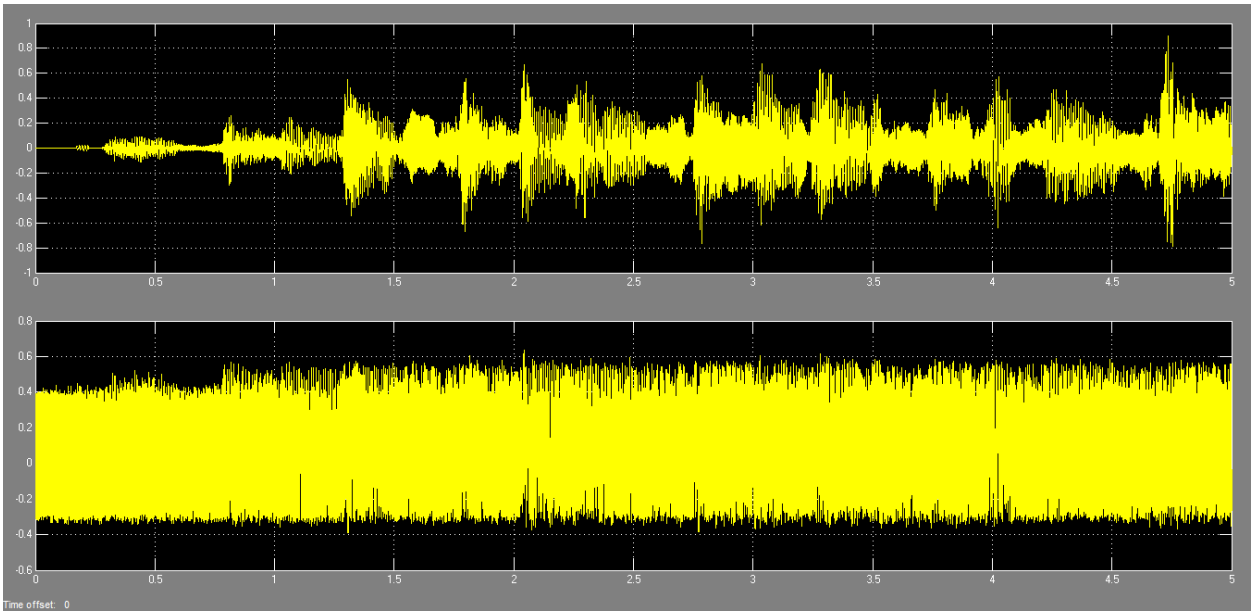


Fig 5.16 **20 kHz/V**

5.3 AUDIO INPUT-2:

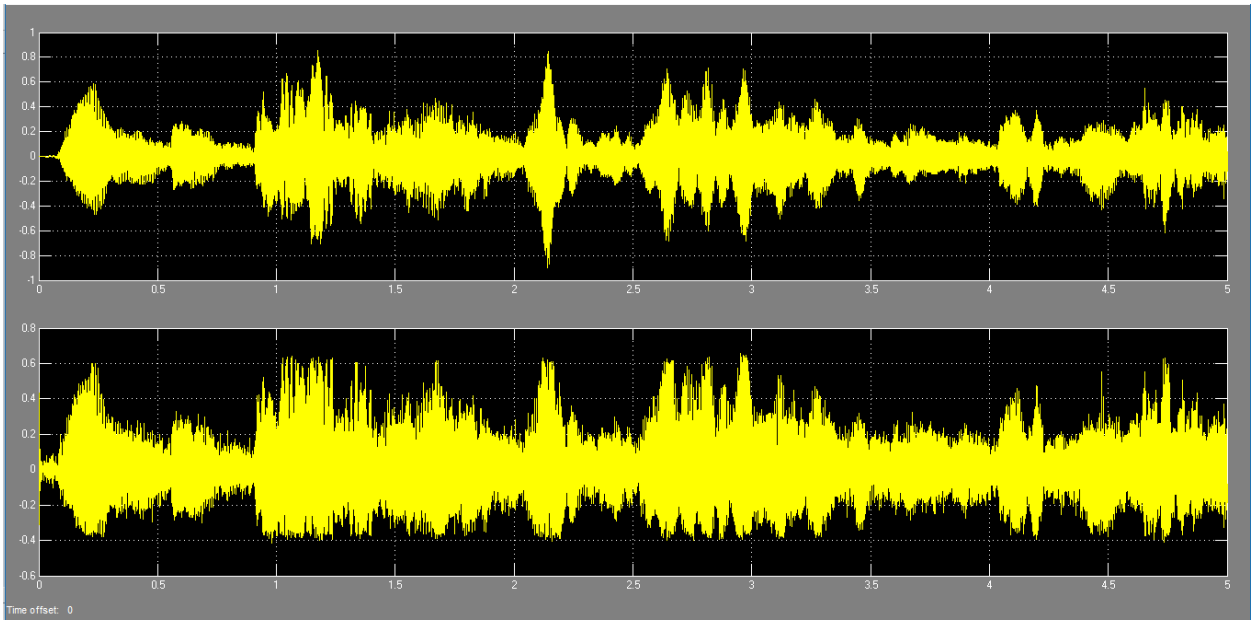


Fig 5.17 **10 kHz/V**

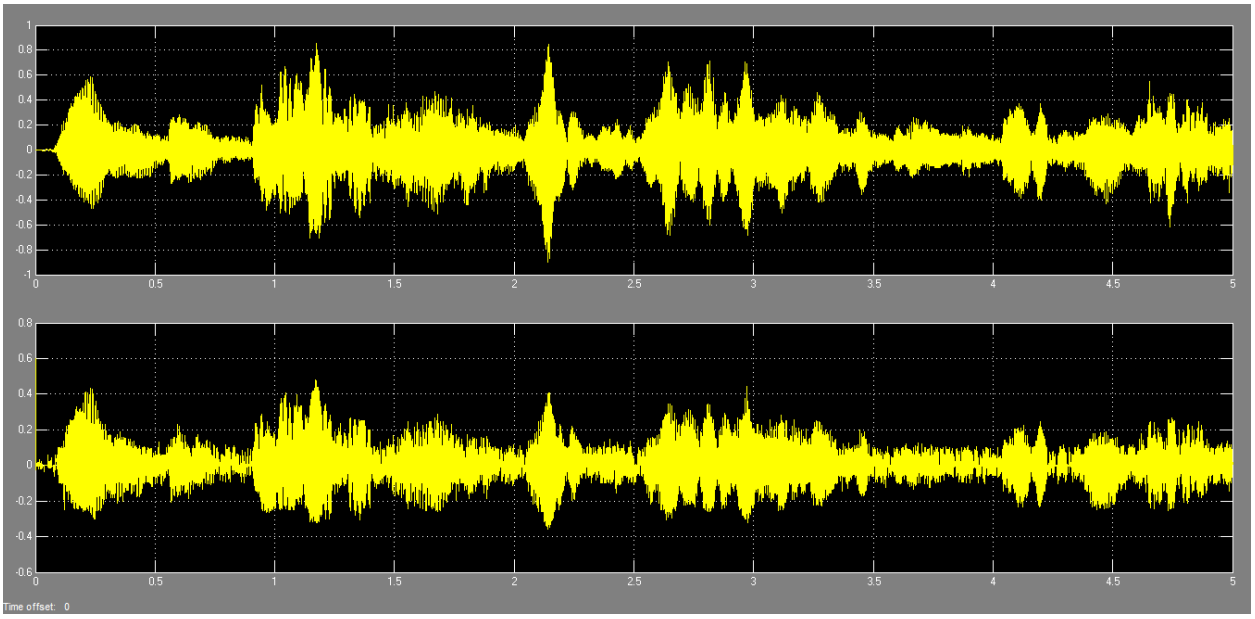


Fig 5.18 **1 kHz/V**

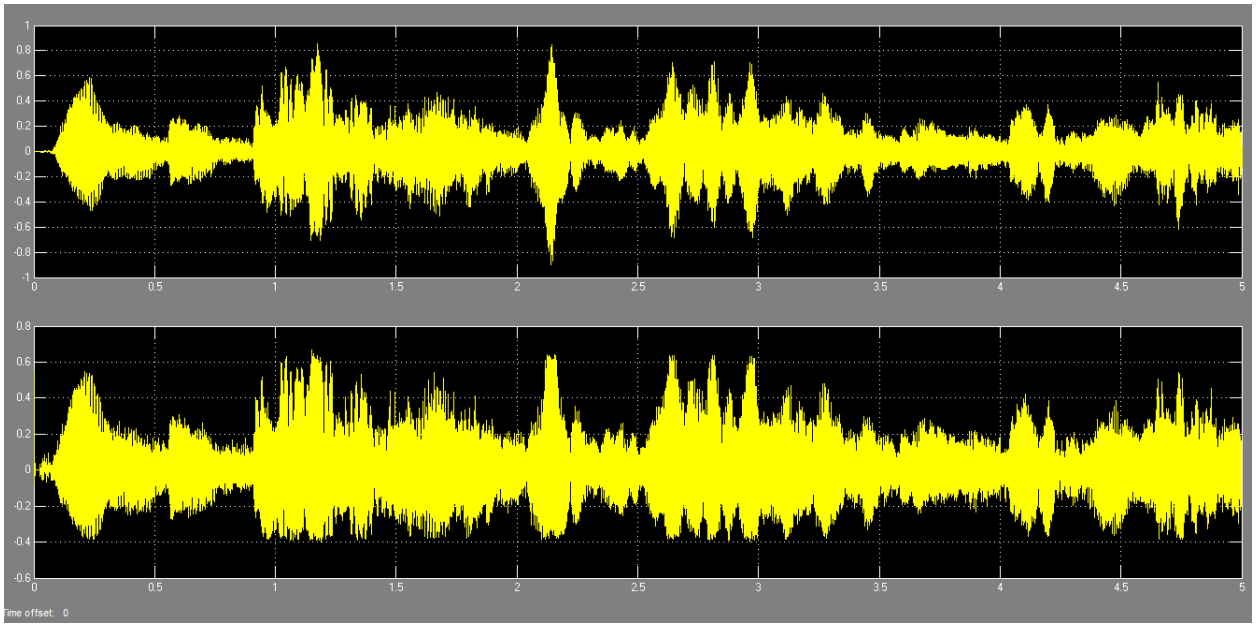


Fig 5.19 **5 kHz/V**

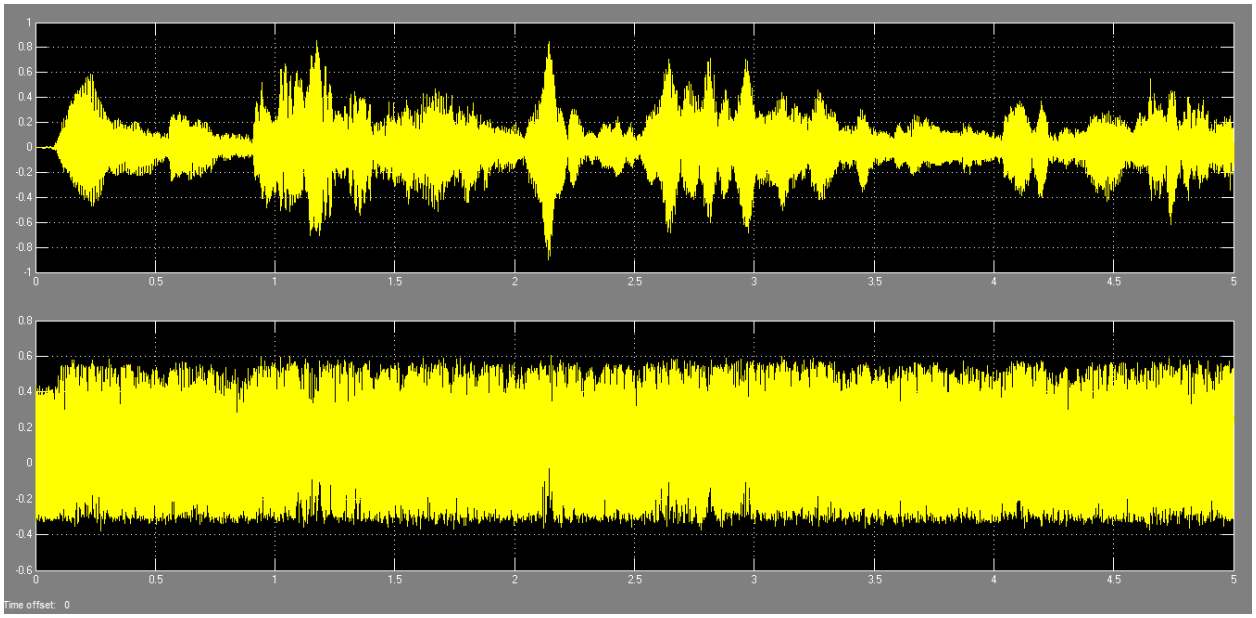


Fig 5.20 **20 kHz/V**

CHAPTER 6

RTL SCHEMATIC

RTL schematic is a Register Transfer Level graphical representation of a pre-optimized design in terms of generic symbols, such as adders, multipliers, counters, AND gates ,OR gates that are independent of targeted Xilinx device. After the HDL synthesis phase of the synthesis process, it can be displayed which helps us to:

- Analyze timing path.
- Analyze how components were inferred.
- Identify the issues and improve the design early in the design process.

RTL SCHEMATICS OF THE BLOCKS:

SINE INPUT:

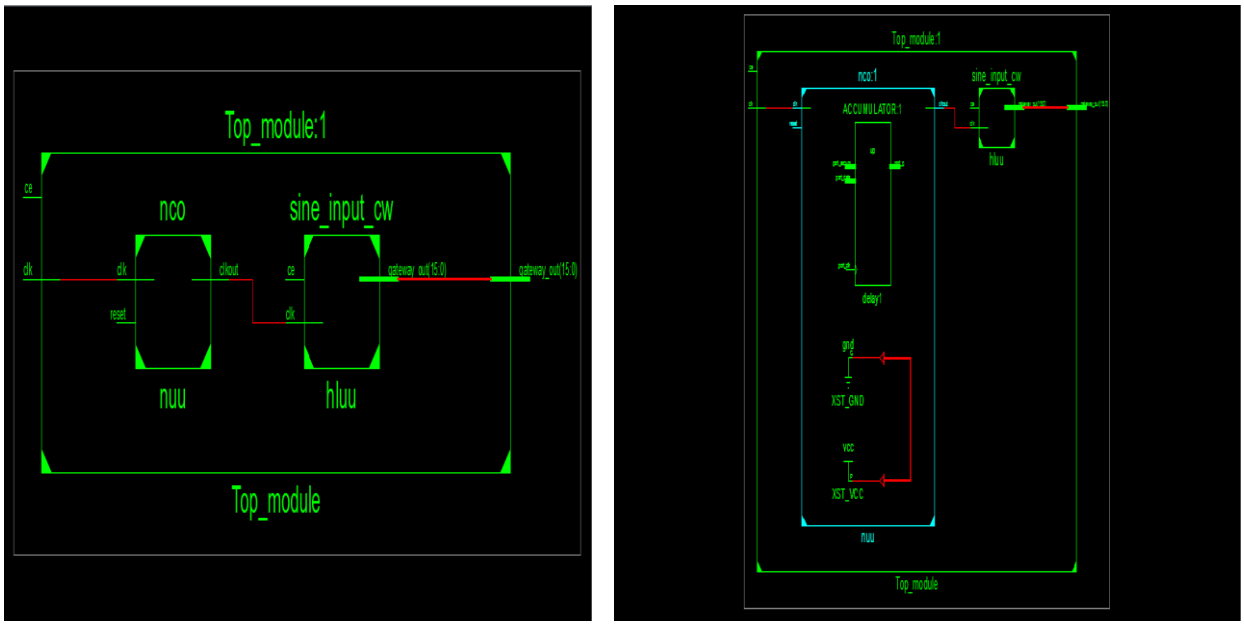


Fig 6.1 RTL Schematic of Sine Input

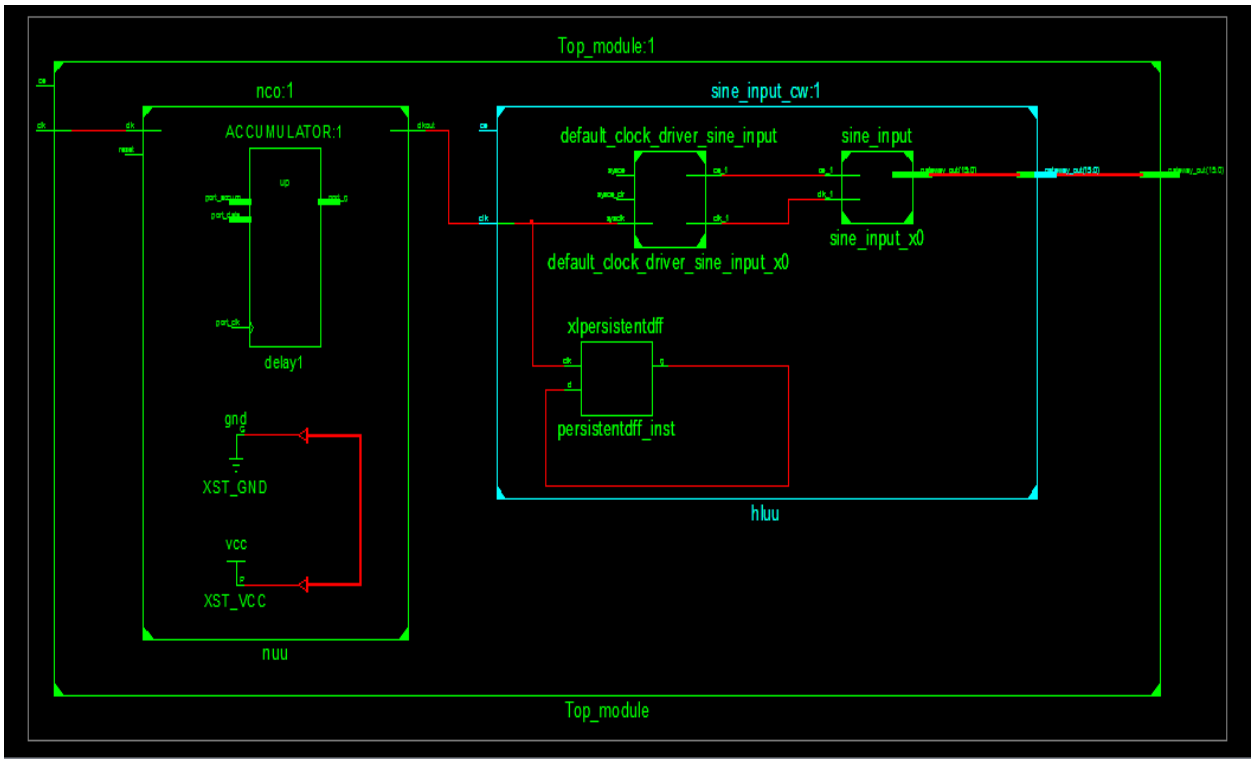


Fig 6.2 RTL Schematic of Sine Input Expanded

NCO:

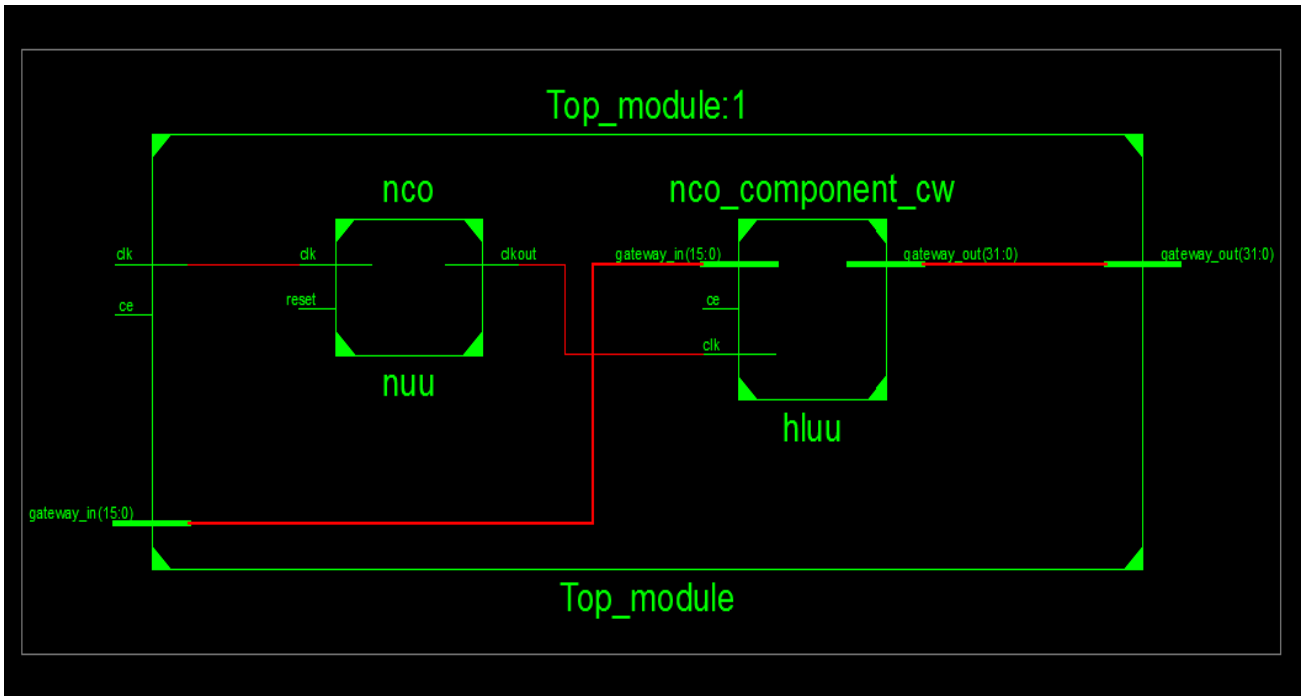


Fig 6.3 RTL Schematic of NCO

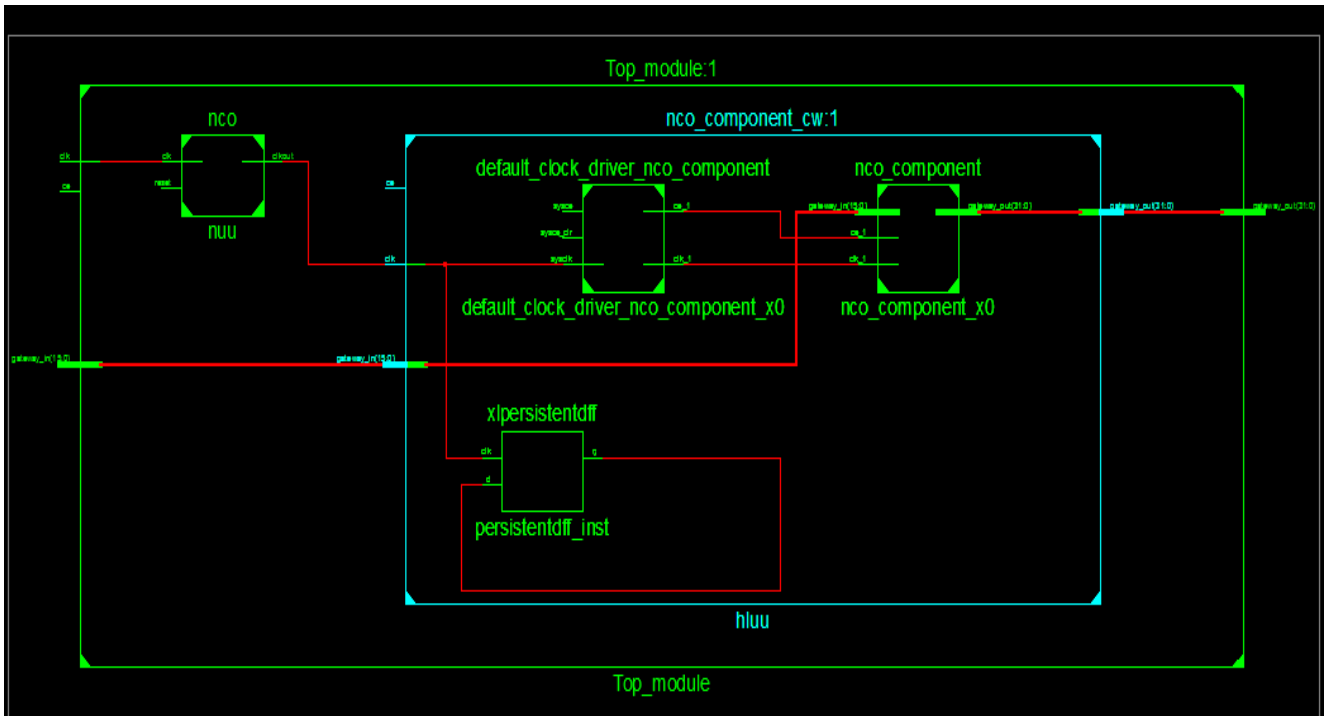


Fig 6.4 RTL Schematic of NCO Expanded

PHASE DETECTOR:

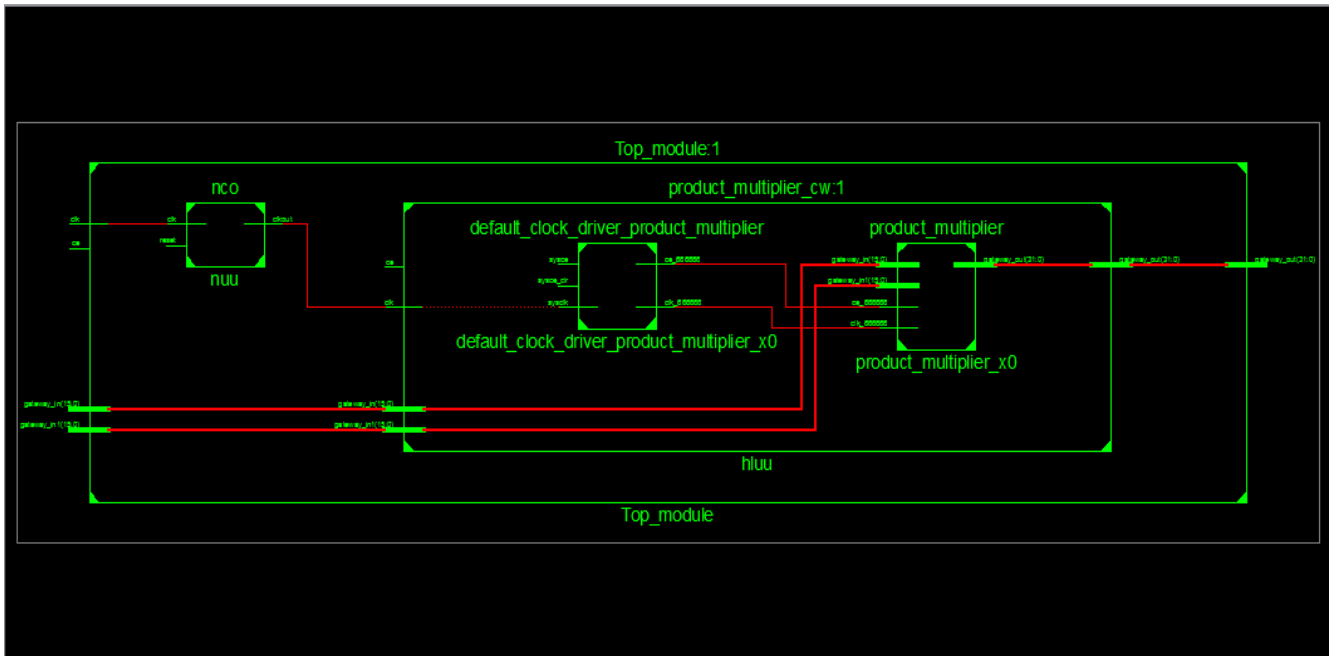


Fig 6.5 RTL Schematic of Phase Detector

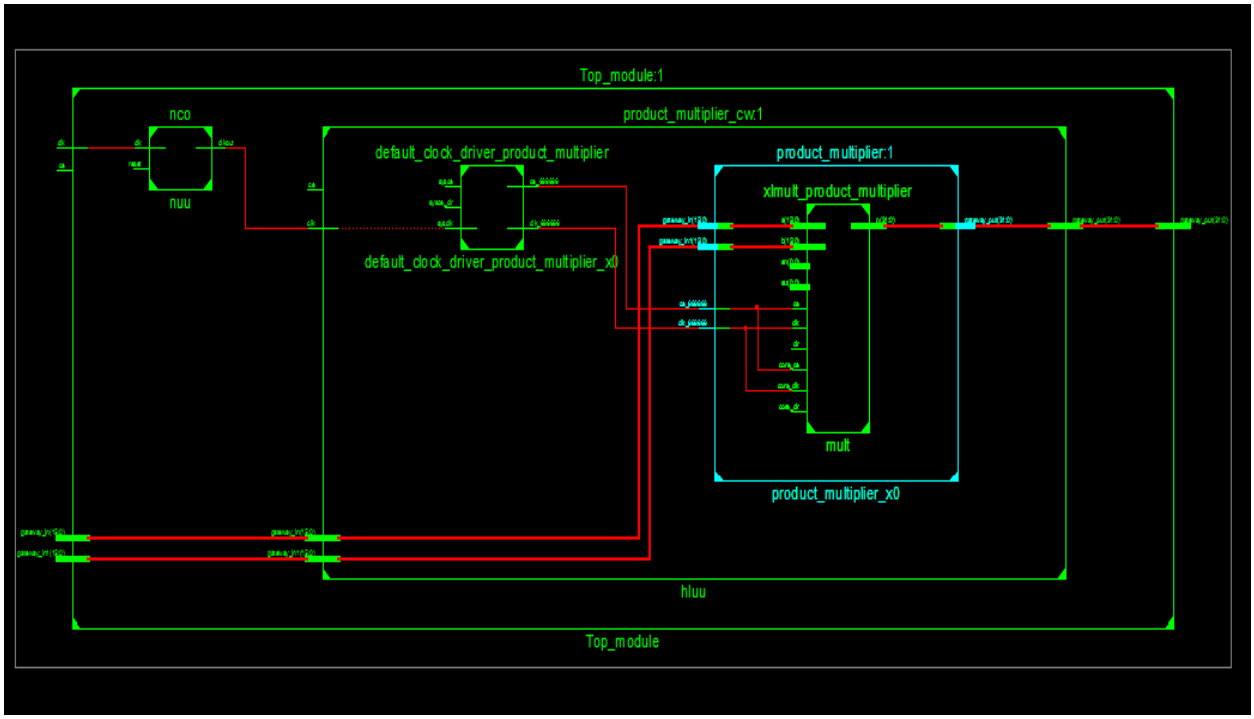


Fig 6.6 RTL Schematic of Phase Detector Expanded

FILTER COMPONENT:

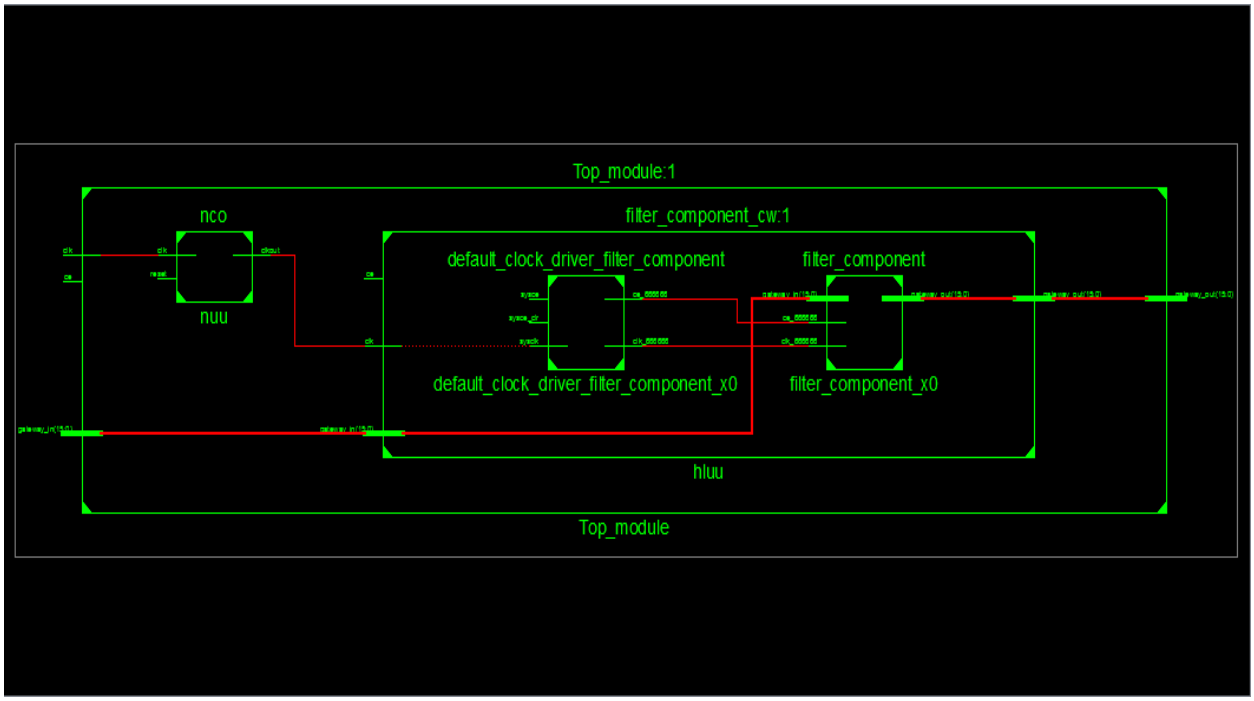


Fig 6.7 RTL Schematic of Filter

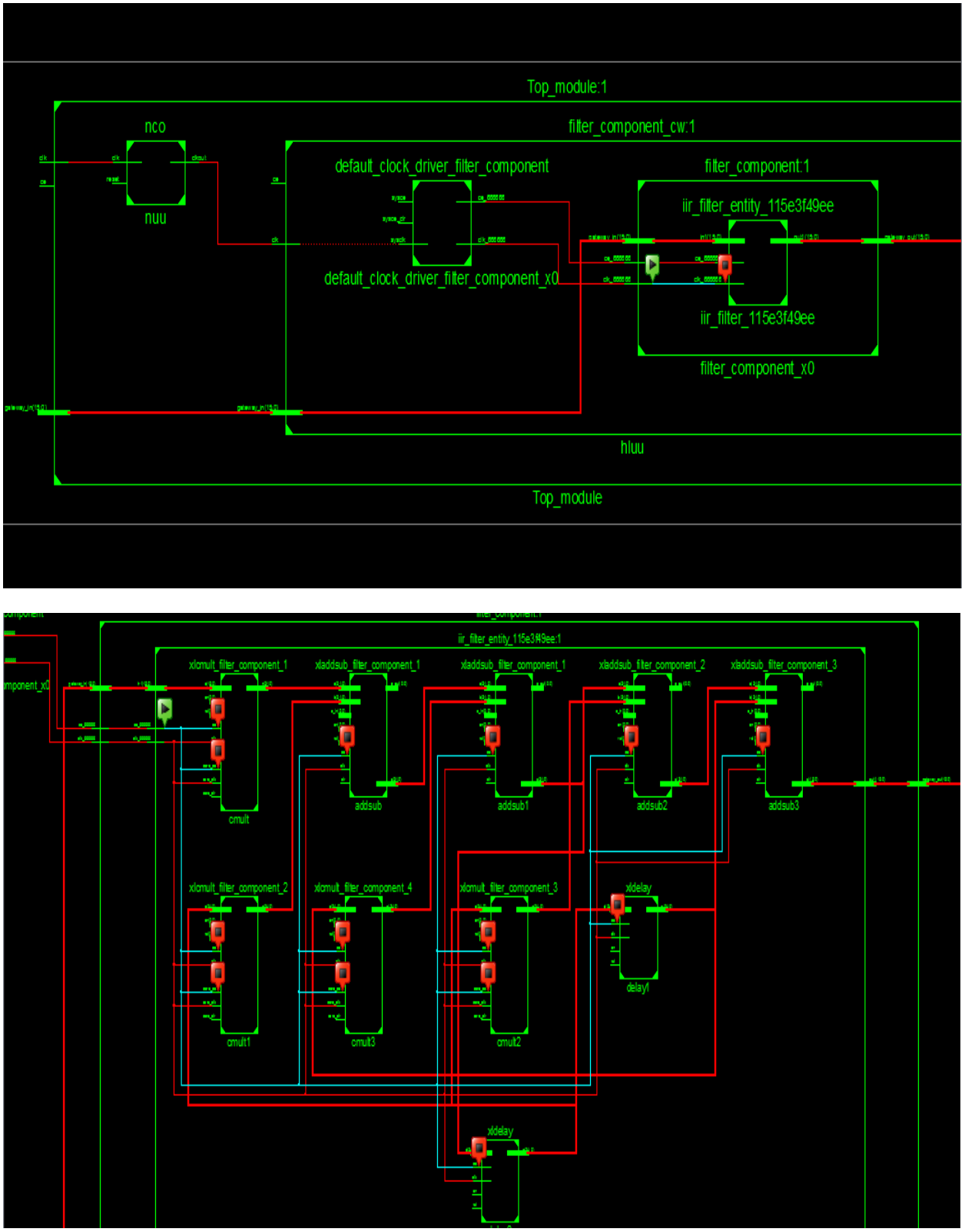


Fig 6.8 RTL Schematic of Filter Expanded

CHAPTER 7

CONCLUSION AND FUTURE SCOPE

With the idea of implementing an adaptive digital modulation and demodulation system, the project with the title ADAPTIVE FM DEMODULATOR has been successfully designed and implemented on the FPGA platform.

The system is implemented in the digital domain with modulation and demodulation being implemented using Numerical controlled oscillator and digital phase locked loop.

The system is made adaptive by providing selectivity in terms of message frequency, carrier frequency and frequency sensitivity using a multiplexer kind of architecture with its select lines given as input to the user to switch between different options.

The system is implemented at the baseband level as the maximum carrier frequency is of 80 kHz. An extension to this system would be the implementation at RF frequencies. This can be achieved by designing a front end extension to the existing system using programmable local oscillators which would up-convert the baseband modulated signal to the desired RF frequency for transmission and down-convert the RF frequency to the desired baseband frequency for demodulation and processing.

CHAPTER 8

. REFERENCES

[1] Mitola. *"The Software Radio Architecture"*. IEEE Communications Magazine, pp. 26-38, Jun-2015.

[2] N Burnett. *"FM Radio Receiver with Digital Demodulation".* A Senior Project presented to the Faculty of the Electrical Engineering Department California Polytechnic State University, San Luis Obispo, 2014.

[3] Juan Pablo Martinez Brito, Sergio Bampi. *"Design of a Digital FM Demodulator based on a 2nd Order All-Digital Phase-Locked Loop".* PGMICRO. Federal University of Rio Grande do Sul, UFRGS 91501-970 Porto Alegre RS Brazil, 2014.

[4] Richa Goel (Department of Electronics, Roorkee College of Engineering, Roorkee, India). *"Digital FM modulator and demodulator for SDR"*. International Journal of Engineering Sciences & Research Technology, July 2015

[5] Lavanya T and Natraj URS HD. *"Design and Implementation of FM modem on FPGA for SDR using Simulink".* International Journal of Engineering Research Volume No.5 Issue: Special 5, pp: 992 -1128, Nov- 2015.

[6] Anupama Patil and Dr P .H .Tandel *"A Numerically controlled oscillator for all Digital Phase Locked Loop".* International Journal of Engineering Trends and Technology (IJETT) – Volume 38 Number 4- August 2016.

[7] Pradnya H.Golghate, Prof.Pankaj Hedaoo. *"Design of Digital Phase Locked Loop for Wireless Communication Receiver Application".* International Journal of Advanced Research in Computer and Communication Engineering.

[8] https://www.arduino.cc/en/Main/arduinoBoardDue

[9] https://www.xilinx.com/support/documentation/data_sheets

APPENDIX

APPENDIX A

INDIVIDUAL CONTRIBUTION

SREESHA Y M

My contribution to the project is the design and implementation of the loop filter part of the digital phase locked loop for FM demodulation.

A survey on the multiple options available for the loop filter design had to be conducted. The digital loop filter can be implemented using a Butterworth, Chebyshev or Elliptic filter. Further the filter can be implemented as an infinite impulse response filter or as a finite impulse response filter. Further the order of the filter has to be decided based on the required roll off. After a thorough survey Butterworth IIR filter of 2^{nd} order with a direct form 2 structure was found to be ideal for the project.

Further the hardware platform for implementation of the above mentioned loop filter had to be chosen. Arduino Due was found suitable for this requirement. Arduino Due is a 32 bit microcontroller based on the ARM architecture. It uses an ARM Cortex M3 processor (SAM3X8E) with a 5-stage pipeline architecture. It has a high frequency clock of 84Mhz.It has 54 Digital and 12 Analog I/O pins and 2 DAC pins. It has support for serial communication protocols such as UART, I2C, SPI and CAN.

Taking the data input to the filter is implemented using the traditional AnalogRead function. The Normal ADC function AnalogRead provides a low sampling rate of 50 kHz. Hence to increase the sampling rate, the internal registers of the ARM processor have to be directly accessed. Similarly, the DAC function AnalogWrite also has a low reconstruction rate and hence the internal registers must be accessed to increase the rate. Some of the internal 32 bit registers such as mode register, control register and channel enable register had to be modified to obtain the necessary sampling and reconstruction rates.

This resulted in ADC channels being set to free running mode and enabled it for sampling the analog input applied. This results in a sampling period of 1.5us i.e. a sampling frequency of 666.666 kHz. Further after modifying the internal DAC registers resulted in the reconstruction rate same as the sampling rate.

For the audio samples modulation and demodulation, the loop filter design is based on its frequency response. The frequency response curve was obtained by first sampling at 8 kHz and taking the 64 point FFT of the obtained samples. Based on the frequency response the filter was further designed.

APPENDIX B

INDIVIDUAL CONTRIBUTION

SURAJ RAO B

My contribution to the project is 1) Enabling the serial communication between Arduino and MATLAB (PC) and identification and utilization of ports available on Arduino DUE to enable communication between FPGA and Arduino. 2) Adaptation of the system when audio samples are given as input.

In order to ensure that the loop filter part of the PLL is adaptive, the filter coefficients have to be changed with the variations in the message frequency accordingly. To implement this feature the filter coefficient calculations were performed in MATLAB and sent to Arduino via serial communication.

Filter Coefficients are calculated in MATLAB and are sent to Arduino through Serial Communication.

Filter coefficients are calculated in MATLAB using the function **"butter".** [B, A] = butter(N, Wn) designs an Nth order low pass digital Butterworth filter and returns the filter coefficients in length N+1 vectors B (numerator) and A (denominator). The coefficients are listed in descending powers of z. The cut-off frequency Wn must be $0.0 < Wn < 1.0$, with 1.0 corresponding to half the sample rate. Filter coefficients are sent from Matlab to Arduino at a baud rate of 9600 using the USB ports.

Further the data taken out from the FPGA and data input to the FPGA are sent and received respectively using digital I/O pins. Towards the loop filter on the microcontroller the digital data is read and written in 12- bit resolution using parallel I/O 32 bit Arm ports. To achieve this functionality, a study of the ARM processor's 32 bit ports and the port functionality was done. Further the port functions for reading and writing port data such as PDSR (Port Data Status Register) and ODSR (Output Data Set Register) were programmed with correct amount of delay values such that the data rate matches on both sides i.e. FPGA and Arduino to successfully transmit and receive data.

In case of audio samples modulation and demodulation, towards the demodulation end, the samples are assigned to the digital Output pins and data is at frequency of 8 kHz which is the original audio signal sampling rate. These demodulated samples are to be sent back to MATLAB on the PC to play the demodulated audio. This is achieved by using the port read function PDSR on Arduino at 8 kHz rate and further these samples are sent using serial communication at a baud rate of 9600 to the MATLAB PC. Thus using the port read functions and serial communication, the demodulated audio playback was achieved.

APPENDIX C

INDIVIDUAL CONTRIBUTION

NISHCHITH JAIN S

My contribution to the project is 1) Designing of the system in Simulink using Xilinx block set making it adaptable to sensitivity variation and a set of carrier frequencies. 2) Adaptation of the system when audio samples are given as input.

The FM modulated signal is given as input to the phase detector. The phase detector in the digital domain is implemented as a multiplier with two inputs. The output from the phase detector is given to the digital filter (IIR Butterworth 2^{nd} order Low pass) which extracts the modulating signal from the phase detector output which consists of sum and difference of frequencies given at its input. The Output from the filter is given as feedback to the NCO which tracks the deviation from the carrier frequency. The loop in this manner extracts the message signal by tracking the frequency variations in the FM signal.

Frequency Sensitivity is the parameter which determines the frequency deviation of the FM wave and also the quality of the demodulated output. The constant value given at the input of the multiplexer determines the frequency sensitivity. The constant is calculated as the ratio of frequency sensitivity to the sampling frequency multiplied by two power the number of bits used to represent the number (here 16).

Using a 2:1 Multiplexer we have two values for frequency sensitivity which are 1000Hz/V and 2000Hz/V respectively. The control input (Select Line) for changing the sensitivity values are controlled through a DIP Switch on FPGA.

The system is made adaptive with respect to variable carrier frequency as well. A set of four carriers are used here of frequencies 40 kHz, 45 kHz, 50 kHz, 55 kHz respectively. A 4:1 multiplexer is used to choose one among the four carriers. The constants given as input to the multiplexer are calculated as mentioned above as the ratio of carrier frequency to the sampling frequency

multiplied by two power the number of bits used to represent the number (here 16).

For the audio samples modulation, the carrier and frequency sensitivity were made adaptive by providing 4 different set of carriers and sensitivity values. Towards the demodulation end, for the audio playback the samples received by serial communication had to be reconstructed using functions such as AudioWrite() and Sound(). Using these functionalities, the demodulated audio was successfully played and results were inferred.

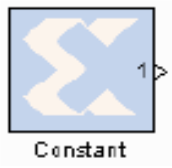
APPENDIX D

XILINX BLOCKS USED IN THE DESIGN

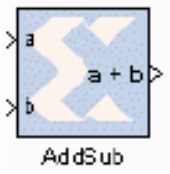
System Generator: System Generator is the industry's leading high-level tool for designing high-performance processor systems using Xilinx All Programmable devices. It provides control of system and simulation parameters, and is used to invoke the code generator. It is also referred to as the System Generator "token" because of its unique role in the design. Every Simulink model containing any element from the Xilinx Block set must contain at least one System Generator block (token). Once this block is added to a model, it is possible to specify how code generation and simulation should be handled. With System Generator,

- Develop highly parallel systems with the industry's most advanced FPGAs
- Provide system modeling and automatic code generation from Simulink and MATLAB
- Integrates RTL, embedded, IP, MATLAB and hardware components of a DSP system
- A key component of the Xilinx DSP Targeted Design Platform

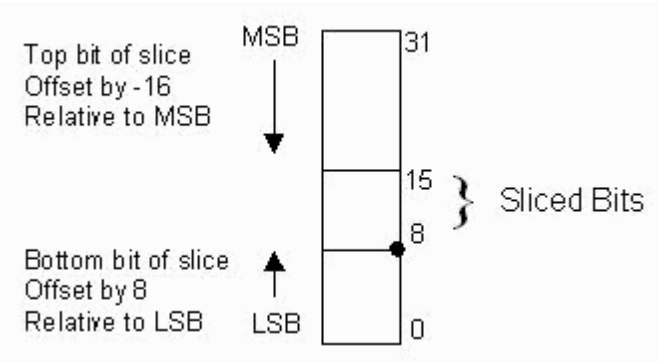
Constant Block: It generates a constant that can be a fixed-point value, a Boolean value, or a DSP48 instruction. It can be used to directly drive the inputs on Xilinx blocks.



Add Sub Block: It implements an adder/subtractor. The operation can be fixed (Addition or Subtraction) or changed dynamically under control of the sub mode signal.



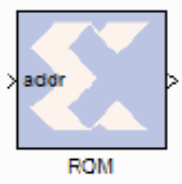
Slice Block: Extracts a given range of bits from each input sample and presents it at the output. The output type is ordinarily unsigned with binary point at zero but can be Boolean when the slice is one bit wide.



ROM Block: It is a single port read-only memory (ROM). Values are stored by word and all words have the same arithmetic type, width, and binary point position. Each word is associated with exactly one address. An address can be any unsigned fixed-point integer from 0 to d-1, where d denotes the ROM depth (number of words). The memory contents are specified through a block parameter. The block has one input port for the memory address and one output port for data out. The address port must be an unsigned fixed point integer. The blocks have two possible Xilinx LogiCORE implementations, using either distributed or block memory. When implementing single port ROM blocks on Virtex®-4, Virtex-5, Virtex-6, Spartan-6, and Spartan®-3A DSP devices, maximum timing performance is possible if the following conditions are satisfied:

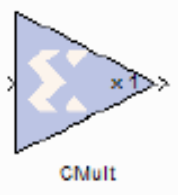
• The option **Provide reset port for output register** is un-checked
• The option **Depth** is less than 16,384

• The option **Latency** is set to 2 or higher



Shift Block: The Xilinx Shift block performs a left or right shift on the input signal. The result will have the same fixed-point container as that of the input.

CMult Block: It implements a *gain* operator, with output equal to the product of its input by a constant value. This value can be a MATLAB expression that evaluates to a constant.



Mult Block: It implements a multiplier. It computes the product of the data on its two input ports, producing the result on its output port.

Delay Block: It implements a fixed delay of L cycles.
The delay value is displayed on the block in the form z-L, which is the Z-transform of the block's transfer function. Any data provided to the input of the block will appear at the output after L cycles. The rate and type of the data of the output will be inherited from the input. This block is used mainly for matching pipeline delays in other portions of the circuit. The delay block differs from the register block in that the register allows a latency of only 1 cycle and contains an initial value

parameter. The delay block supports a specified latency but no initial value other than zeros. The figure below shows the **Delay** block behaviour when **L=4** and **Period=1s**.

Gateway Out: Xilinx Gateway Out blocks are the outputs from the Xilinx portion of your Simulink design. This block converts the System Generator fixed-point Data type into Simulink Double. According to its configuration, the Gateway Out block can either define an output port for the top level of the HDL design generated by System Generator, or be used simply as a test point that will be trimmed from the hardware representation.

Scope: It helps to observe the obtained graphs/output.

In: Provide an input port for a subsystem or model.

Out: Provide an output port for a subsystem or model.

APPENDIX E

ARDUINO CODE:

RECEIVING COEFFICIENTS FROM MATLAB AND REALIZATION OF FILTER:

```
#define pinport PIOC
#define pininput PIOB
unsigned long values[2000],valuesIn[2000];
unsigned long out[2000];
float a[2],b[3];
unsigned long u[2000],v[2000];
boolean flag,flag2;
unsigned int i,j,k,g,d,e,z,ab,cd,xyz;
double p,q,r;
float value[7];
void setup()
{
  Serial.begin(9600);
  SerialUSB.begin(9600);
  k=0;
  analogWriteResolution(12);
  analogWrite(DAC1,0);
  flag=false;
  flag2=false;

  pinMode(48,INPUT);
  pinMode(3,INPUT);
  pinMode(33, OUTPUT) ;
  pinMode(34, OUTPUT) ;
  pinMode(35, OUTPUT) ;
  pinMode(36, OUTPUT) ;
  pinMode(37, OUTPUT) ;
  pinMode(38, OUTPUT) ;
  pinMode(39, OUTPUT) ;
  pinMode(40, OUTPUT) ;
  pinMode(41, OUTPUT) ;
  pinMode(51, OUTPUT) ;
  pinMode(50, OUTPUT) ;
  pinMode(49, OUTPUT) ;
  pinport -> PIO_CODR = 0xFFFFFFFF;
  pinport -> PIO_OWER = 0xFFFFFFFF;
```

```
  pinMode(20,INPUT);
  pinMode(21,INPUT);
  pinMode(53,INPUT);
  pinMode(DAC0,INPUT);
  pinMode(DAC1,INPUT);
  pinMode(A8,INPUT);
  pinMode(A9,INPUT);
  pinMode(A10,INPUT);
  pinMode(A11,INPUT);
  pinMode(52,INPUT);
  pinMode(2,INPUT);
  pinMode(22,INPUT);

  pininput ->PIO_CODR = 0xFFFFFFFF;
  pininput ->PIO_OWER = 0xFFFFFFFF;
}

void loop()
{
  while(SerialUSB.available()>0)
  {
    value[k]=SerialUSB.parseFloat();
    k++;
    if(k==7)
    {
      coefficients(value);
       break;
     }
}
cd=digitalRead(3);
z=digitalRead(48);
    if(flag)
    {
      if(z==0)
      {
        for(i=0;i<2000;i++)
        {
        valuesIn[i]=pininput ->PIO_PDSR&104853504;;
         delayMicroseconds(1);
        }
        for(ab=0;ab<2000;ab++)
        {
          values[ab] =(((valuesIn[ab] >> 12) & 1023) + ((valuesIn[ab] >> 15) & 3072));
        }
```

```
  u[0]=values[0];
  u[1]=values[1]-a[0]*u[0];
  out[0]=0;
  out[0]=b[0]*u[0];
  out[1]=0;
  out[1]=out[1]+b[0]*u[1];
  out[1]=out[1]+b[1]*u[0];
  for(i=2;i<2000;i++)
  {
    u[i]=values[i];
    for(j=1;j<=2;j++)
    {
      u[i]=u[i]-a[j-1]*u[i-j];
    }
    v[i]=0;
    for(j=0;j<=2;j++)
    {
      v[i]=v[i]+b[j]*u[i-j];
    }
    out[i]=v[i];
  }

  Serial.println("Hello");
  }
  }
  if(flag2)
  {
if(cd==0)
{
 for(g=0;g<2000;g++)
 {
  pinport -> PIO_ODSR =((( out[g] & 511) <<1) | ((out[g] & 3584)<<3));
  delayMicroseconds(1);
  }
  Serial.println("World");
}
else
{
   for(i=0;i<2000;i++)
  {
  dacc_set_channel_selection(DACC_INTERFACE, 1);       //select DAC channel 1
  dacc_write_conversion_data(DACC_INTERFACE, out[i]);
  delay(0.1);//write on DAC
  }
```

```
}
}
}
void coefficients(float value2[])
{
if((int)value2[6]==1)
 {

   Serial.println('message signal frequency is between 5kHz and 6kHz');
   value2[0]=value2[0]/10000;                    // B(1)=B(1)*(10^4)
   value2[1]=value2[1]/1000;                     //B(2)=B(2)*(10^3)
   value2[2]=value2[2]/10000;                    //B(3)=B(3)*(10^4)
   value2[4]=value2[4]*(-1);                     //A(2)=A(2)*(-1)
   value2[5]=value2[5]/10;                      //A(3)=A(3)*(10)
   b[0]=value2[0];
   b[1]=value2[1];
   b[2]=value2[2];
   a[0]=value2[4];
   a[1]=value2[5];
   for(i=0;i<7;i++)
   {
    Serial.println(value2[i],8);
    Serial.flush();
   }
    flag=true;
    flag2=true;
  }
 else if((int)value2[6]==2)
  {
    Serial.println('message signal frequency is between 6kHz and 10kHz');
    value2[0]=value2[0]/1000;                    // B(1)=B(1)*(10^3)
    value2[1]=value2[1]/1000;                    //B(2)=B(2)*(10^3)
    value2[2]=value2[2]/1000;                    //B(3)=B(3)*(10^3)
    value2[4]=value2[4]*(-1);                    //A(2)=A(2)*(-1)
    value2[5]=value2[5]/10;                     //A(3)=A(3)*(10)
    b[0]=value2[0];
    b[1]=value2[1];
    b[2]=value2[2];
    a[0]=value2[4];
    a[1]=value2[5];
    for(i=0;i<7;i++)
    {
    Serial.println(value2[i],8);
    Serial.flush();
```

```
  }
 flag=true;
 flag2=true;
 }}
```

SENDING AUDIO SAMPLES TO MATLAB THROUGH SERIAL COMMUNICATION:

```
#define pininput PIOC
unsigned short valuesIn[40000];
unsigned int i,ab,z,cd,j,pin;
double p,q,r;
void setup()
{
 Serial.begin(9600);
 SerialUSB.begin(9600);
 pinMode(13,OUTPUT);
 pinMode(33, INPUT) ;
 pinMode(34, INPUT) ;
 pinMode(35, INPUT) ;
 pinMode(36, INPUT) ;
 pinMode(37, INPUT) ;
 pinMode(38, INPUT) ;
 pinMode(39, INPUT) ;
 pinMode(40, INPUT) ;
 pinMode(41, INPUT) ;
 pinMode(51, INPUT) ;
 pinMode(50, INPUT) ;
 pinMode(49, INPUT) ;
 pinMode(2,INPUT);
 pininput ->PIO_CODR = 0xFFFFFFFF;
 pininput ->PIO_OWER = 0xFFFFFFFF;
}
void loop()
{
 pin = digitalRead(2);
 for(i=0;i<40000;i++)
    {
      valuesIn[i]=pininput ->PIO_PDSR & 65535;
       delayMicroseconds(124);
```

```
    }
  for(ab=0;ab<40000;ab++)
    {
        z = (((valuesIn[ab]>>1) & 511) + ((valuesIn[ab]>>3) & 3584));
        valuesIn[ab]=z;
    }
  if(pin==0)
  {
  for(i=0;i<40000;i++)
   {
     SerialUSB.println(valuesIn[i]);
     Serial.begin(1);
   }
  }
}
```

MATLAB CODE:

CALCULATION OF FILTER COEFFICIENTS AND SENDING THEM TO ARDUINIO USING SERIAL COMMUNICATION:

```
arduino=serial('COM3','BaudRate',9600);
message_frequency=input('enter the message frequency between 5kHz and 10kHz');
frequency=message_frequency;  %convert to MHz
[B,A]=butter(2,frequency/333.333);  %calculate the filter coefficients

%based on the range of the message frequency select the scaling coefficients
if (message_frequency>=5 && message_frequency<=6 )

B(1)=B(1)*(10^4);
B(2)=B(2)*(10^3);
B(3)=B(3)*(10^4);
A(1)=A(1)+0.00001;
A(2)=A(2)*(-1);
A(3)=A(3)*(10);
M = [B,A,1.1];
fopen(arduino);
fprintf(arduino,'%s',M);
fclose(arduino);
```

```
elseif(message_frequency >6 && message_frequency<=10)

B(1)=B(1)*(10^3);
B(2)=B(2)*(10^3);
B(3)=B(3)*(10^3);
A(1)=A(1)+0.00001;
A(2)=A(2)*(-1);
A(3)=A(3)*(10);
M = [B,A,2.2];
fopen(arduino);
fprintf(arduino,'%s',M);
fclose(arduino);

else
   disp('invalid input');
end
```

READING AUDIO FILES:

```
clear all;
[a,fs1] = audioread('sound1.wav');
[b,fs2] = audioread('sound2.wav');
[c,fs3] = audioread('sound3.wav');
[d,fs4] = audioread('sound4.wav');

n=length(a);
xdft=fft(a);
xdft=xdft(1:n/2+1);
psdx=(1/(8000*n))*abs(xdft).^2;
psdx(2:end-1)=2*psdx(2:end-1);
freq=0:8000/length(a):8000/2;
figure(1);
plot(freq,10*log10(psdx))
grid on
title('Periodogram Using FFT')
xlabel('Frequency Hz')
ylabel('Power/Frequency(db/Hz)');
```

RECEIVING AUDIO SAMPLES FROM ARDUINO THROUGH SERIAL COMMUNICATION:

```
clear all;
clc;
delete(instrfindall);
if ~isempty(instrfind)
   fclose(instrfind);
   delete(instrfind);
end
y=int16([]);
z=double([]);
cd=double([]);
ab=double([]);
arduino = serial('COM3','BaudRate',9600,'DataBits',8,'StopBits',1);
fopen(arduino);
i=1;
while(i<=40000)
   y(i)=fscanf(arduino,'%d');

   i=i+1;
end
disp(y);
fclose(arduino);
z=double(y);
w=double(z/2048);
cd=w-1;
ab=cd*0.65;
audiowrite('sample.wav',ab,8000);
[abc,Fs]=audioread('sample.wav');
sound(abc);
```