POLYDYNE DISPLACEMENT INTERFEROMETER USING FREQUENCY-MODULATED LIGHT

by

Masoud Arablu

A dissertation submitted to the faculty of
The University of North Carolina at Charlotte
in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in
Mechanical Engineering

Charlotte

2019

ABSTRACT

MASOUD ARABLU. Polydyne displacement interferometer using frequency-modulated light (Under the direction of DR. STUART T. SMITH)

A novel multi-frequency (polydyne) optical interferometry method is introduced and a prototype developed to evaluate its performance in a Michelson displacement interferometer configuration. The polydyne interferometer contains three main parts of 1) synchronous Radio-Frequency Frequency-Modulated (RF-FM) electrical signal generation, 2) electrical-to-optical signal patching by an Acousto-Optic Modulator (AOM), and 3) signal detection and processing unit. Each of these three parts are discussed briefly in a separate chapter as well as the basic concept of phase extraction from the interference of FM light beams.

A novel synchronous RF-FM signal generation method is introduced that uses an atomic clock for timing all frequency components of the signal. The RF-FM signal generator uses a modulated, voltage-controlled time delay to correspondingly modulate the phase of a 10 MHz sinusoidal reference from the atomic clock. This modulated reference signal is, in turn, used to clock a Direct Digital Synthesizer (DDS) circuit resulting in an FM signal at its output. The modulating signal that is input to the voltage-controlled time delay circuit is generated by another DDS that is synchronously clocked by the same 10 MHz sine wave signal before modulation. Therefore, all the digital components are timed from a single sinewave oscillator that forms the basis of all timing.

The resultant output electrical signal comprises a center, or carrier, frequency plus a series of phase-synchronized sidebands having exact integer harmonic frequency separation.

The RF-FM electric signal is transferred into a He-Ne laser beam by diffraction of the beam through an Acousto-Optic Modulator (AOM). The first diffraction side-beam emerging from the AOM is selected by a slit to be used in a Michelson interferometer topology. Frequency spectra of the interfered FM light beams contains the harmonics of modulation frequency. The displacement measurement is derived from the phase measurement of selected modulation harmonic pairs. Individual modulation harmonic amplitudes are measured using Fourier transform applied to the signal from a single photodetector. Lock-in amplifiers were first used to perform Fourier transform on the detected signal. Displacement of the moving target was measured by harmonic pairs chosen from harmonics 1 to 5 of the modulation signal.

Since the analog Lock-In amplifiers use low-pass filters to detect the signal of interest, they limit the bandwidth of measurement to one-tenth of measured signal frequency. This limit reduces the speed of measurement to a few $\mu m \cdot s^{-1}$ in our system. Therefore, after validating the feasibility of displacement measurement using the changes in the amplitudes of harmonic pairs detected by Lock-in amplifiers, a Discrete Fourier Transform (DFT) algorithm was developed on a Field Programmable Gate Array (FPGA) microchip to increase the bandwidth and speed of measurement as well as to miniaturize the signal processing unit. The timing of the FPGA microchip was developed from a Phase Locked Loop (PLL) synchronous to the same 10 MHz sine wave from the atomic clock that is timing all the frequency components of the RF-FM signal. This synchronizes all the timings in the signal generation and detection units. The developed synchronous DFT provides measurements with speeds up to 10 mm·s^{-1}, a limit that can be always improved using larger FPGA microchips and faster analog-to-digital convertors on the photodetector output. In the developed synchronous DFT algorithm, displacement-related-phase-change is derived from the amplitudes of harmonics 1 and 2 of the modulation signal. The measured displacements are compared with a commercial heterodyne interferometer being used as a reference for these studies. Displacements of the moving mirror of the interferometer over ranges up to 10 μm with speeds up to 10 mm·s^{-1} all show differences of less than 50 nm between the polydyne interferometer and the reference interferometer measurements. A drift test is also used to evaluate long-term stability and repeatability of measurements with the developed polydyne interferometer.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my professor Dr. Stuart T. Smith. It has been a great honor to be his student. His idiosyncratic approach toward his research was contagious and motivational for me, even during tough times in my Ph.D. pursuit. Not only he educated me in different subjects of mechatronics, but also reshaped my personality to be eternally passionate to learn new technologies. I appreciate all his contributions of time, ideas, and funding to make my Ph.D. experience productive.

I gratefully acknowledge the funding sources that made my Ph.D. work possible. I was funded by the Graduate Assistantship Program for all my five years of education at UNC Charlotte and was honored to conduct a fruitful research at the Center for Precision Metrology (CPM) that funded my project for about three years. The professors of the CPM have contributed immensely to my personal and professional time at UNC Charlotte. The center has been a source of friendships as well as good advice and collaboration. I am especially grateful to Dr. Jimmie Miller who helped me with his laboratory facilities as well as his insightful comments especially on the drift test setup and results that were very crucial for evaluating long-term stability of the instrument developed in my thesis. Thanks to Jennifer Chastain for her extensive helps to procure the required material used in my setup. I would like to acknowledge Dr. Christopher Evans who has supported my project as the head of the center. I would like to thank Mr. John Brien for his valuable advice during the development of the frequency-modulator circuit. I also thank Dr. Thomas Weldon from electrical engineering department at UNC Charlotte, for use of his laboratory facilities to test my electronic boards. For this dissertation I would like to thank my reading committee members: Dr. Falaggis, Dr. Her, Dr. Miller, Dr. Smith, and Dr. Tarbutton for their time, interest, and helpful comments.

My time at UNC Charlotte was enjoyable especially due to many friends who have become a part of my life. I am grateful for time spent with friends, for our memorable trips to the mountains, for Farid Javidpour, Amirreza Sahami, Faraz Hedayati, Vahid Hemmati, Saeed Heysiattalab, Fatemeh Mirzaee’s hospitality. Many thanks to my great lab-mates, Jeffery Thousand, Kumar Arumugam, and Sajad Kafashi who helped me in programming a lot of codes. I would like to thank Pedram Poshtiban from IUST for teaching me the VHDL language for programming FPGAs.

Last but not least, I am grateful to my family for all their love and encouragement. For my parents who raised me with a love of science and supported me in all my pursuits. Special thanks to my dad for assigning me tough tasks as a kid to prepare me for the rainy days of my life. For my beloved mom who has truly devoted herself to our family. For my elder brothers Yousef and Naser who shed a light into the path that I took as an engineer as well as my lovely sister, Nahid, who has always supported me as her little brother. Thank you all.

Masoud Arablu
UNC Charlotte
May 2019

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABREVIATIONS

AOM	Acousto-Optic Modulator
ADC	Analog to Digital Convertor
BJT	Bipolar Junction Transistor
CLCK	Clock frequency, (Hz)
DAC	Digital to Analog Converter
dBm	decibel milliwatts
DDS	Direct Digital Synthesizer
DSP	Digital Signal Processing
FET	Field Effect Transistor
FPGA	Field Programmable Gate Arrays
FM	Frequency Modulated
FSK	Frequency Shift Keying
HP	Hewlett-Packard
LiDAR	Light Detection And Ranging
LUT	Look-Up Table

PCB	Printed Circuit Board
PLL	Phase Locked Loop
PSK	Phase Shift Keying
PZT	Lead Zirconate Titanate
RF	Radio Frequency
SCLCK	System clock frequency inside the DDS after multiplier, (Hz)
SFDR	Spurious-Free Dynamic Range
SMA	SubMiniature version A
VHDL	Very high speed integrated circuit Hardware Description Language
VHF	Very High Frequency

CHAPTER 1 INTRODUCTION

This chapter consists of two sections that present a summary of literature about displacement measurement interferometers in section 1.1, and objectives of this work in section 1.2.

1.1 Literature Review

The essential idea behind optical interference is that since light acts as an electromagnetic wave, optical interference results from the superposition of the electric field of two or more light beams as waves. Optical interference has played a prominent role in modern technology. It was first used by Thomas Young to discover the wave nature of light via the well-known double-slit experiment [1]. Albert A. Michelson and Edward W. Morley used white light interference through the well-known Michelson interferometer to show that there was no foundation for the existence of an aether [2]. Since then, there have been numerous advances in optics, interferometry, and supporting fields like; stabilized long coherence lasers, low-noise photodetectors, and ultrahigh speed, ultralow-noise electronic microchips for signal generation, detection and processing. All these advances along with Michelson's technique have produced many great technologies such as spectroscopy [3], displacement measuring interferometry [4], absolute distance measuring interferometry [5], Light Detection And Ranging (LiDAR) velocimetry [6], optical gyroscope [7], and so on.

Twyman and Green modified Michelson's interferometer by collimated light [8]. The method used by Twyman and Green is nowadays used in most displacement measurement interferometers for measuring the position change of a target object. A simple schematic of a displacement measuring interferometer is shown in Figure 1-1.

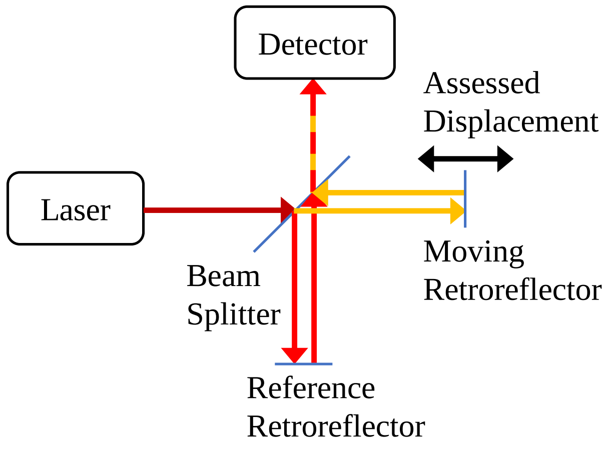


Figure 1-1- Schematic of a simple displacement measuring interferometer.

Typically, the fixed arm is called the reference arm and the moving arm is called the measurement arm. Any displacement in the measurement arm changes the optical path length of the measurement arm causing the intensity of the interfered beam at the detector to vary between destructive and constructive patterns. Assuming a wavelength of λ, displacement of ΔL results in a phase change of ϕ in the captured intensity of the interference:

$$\phi = 2\pi n N \Delta L / \lambda . \qquad (1\text{-}1)$$

where, n is the refractive index of air, N the path factor that is the number of times the light beam traverses the displacement path. Since the measured phase is a modulus of 2π, it should be unwrapped to determine the actual displacement.

The primary objective of any displacement measuring system is to perform linear and repeatable measurements. Repeatability as the maximum deviation between the measurements of a measurand under the same conditions and with the same measuring instrument refers to how stable the measurement will be over time. System measurement repeatability consists of two parts of short-term and long-term repeatability. Short-term repeatability is the measurement stability over a period shorter than one hour; long-term repeatability is stability over a period longer than one hour. Repeatability error budgets of a displacement measurement interferometer are categorized as intrinsic (laser wavelength, electronics error, optics nonlinearity), environmental (atmospheric compensation, material thermal expansion, optics thermal drift), and installment (dead-path error, Abbé error, cosine error). Intrinsic error components are the main focus of instrument manufacturers [9].

Displacement interferometers, just like any other displacement measurement systems, can be classified based on their dynamic range and precision. Some applications require high resolution (0.1 nm to 10 nm), low dynamic range interferometers [10] such as metrology for semiconductor fabrication [11], large optics fabrication [12], stellar interferometry [13], and precision robotics in medicine [14]. Another significant group of applications like machine tool metrology [15] require coarser resolutions (about 10 nm to 50 nm) displacement measurement interferometers with measurement ranges of a few meters and speeds of greater than 10 mm·s^{-1}.

Through decades of research effort, a wide variety of methods have been developed to overcome the challenges on building a robust, repeatable, linear interferometer [10]. All of these methods are generally categorized as either homodyne

[16] or heterodyne systems [17, 18]. In homodyne interferometers, the phase change is measured from DC-level fluctuations of the intensity of the interference. The major disadvantages of homodyne interferometers are their inherent lack of directional sensitivity and dependence on a constant maximum intensity of signal. The later has been solved by automatic compensation of the changes of the signal level [19]. The former problem has been solved by two different methods of adding directional sensitivity to the homodyne interferometers using: 1) phase quadrature measurement by applying known phase shift to the detected signal [20, 21], 2) amplitude demodulation technique by modulating the laser intensity with a known carrier frequency [4].

Heterodyne interferometers use a laser light containing two distinct frequencies that have a known frequency difference, i.e. beat/heterodyne frequency. These two frequencies are orthogonally polarized so that they can perform direction sensitive displacement detection. Each of the orthogonally polarized optical waves construct one of the arms, i.e. the reference arm and the measurement arm, of the interferometer. Since orthogonally polarized waves do not interfere, the interference is achieved by passing orthogonal beams through a polarizer aligned at 45 degrees [4]. Since the heterodyne interferometers usually have a heterodyne frequency of a few megahertz, they require a high detection bandwidth which is usually achieved by flash photodetectors and other ultrahigh-speed electronics. These requirements make the heterodyne systems more complex and expensive in comparison with homodyne systems.

Required bandwidth of homodyne and heterodyne systems are shown in Figure 1-2. The interferometers detection bandwidth limit is drastically increasing by the increase of the electronics microchips speed so that the state-of-the-art miniature retroreflector homodyne interferometers provide 800 mm·s^{-1} with 20 pm resolution [22]. Moreover, state-of-the-art heterodyne interferometers provide extraordinary low noise of 7.1 fm·Hz^{-1} at 21 MHz [23].

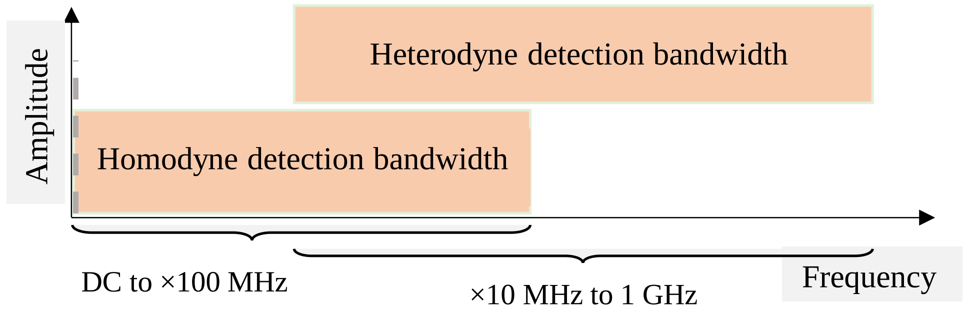


Figure 1-2- Typical detection bandwidth of homodyne and heterodyne interferometers.

Commercial laser-based displacement measurement interferometer systems typically use tunable [24], Zeeman split [10], or unmodulated Acousto-Optic Modulator (AOM) Doppler shifted lasers [25] to provide a frequency shifted light source. Rapidly tunable lasers are widely used for communication applications, typically with wavelength of 1550 nm, and controlled tuning frequencies up to 1.2 kHz. Consequently, this limits the measurement bandwidth to below 1 kHz [24, 26, 27]. Zeeman split and AOM Doppler shift methods provide only two frequencies thereby providing heterodyne sources.

A new generation of interferometers have recently been introduced that modulate the frequency of the laser beam by a sinusoidal phase. In this method, the phase information of interest is extracted from the harmonics of the modulation frequency as the beats. The sinusoidal phase can be injected into the laser beam by different methods. In the first method [28], the beam propagates in an optical fiber and a piezo applies a sinusoidal motion to the optical fiber; this method is called deep phase modulation interferometry. In the second method [29, 30], an electro-optical amplitude modulator modifies the laser frequency with a sinusoidal signal; this method is called deep frequency modulation interferometry. Highly compact extrinsic Fabry-Perot interferometers [26, 27] is another example of dynamic sinusoidal frequency modulation technique that uses the same method of deep frequency modulation interferometry for injecting the sinusoidal phase into the laser beam.

The dynamic frequency modulation technique is different from that of phase-shifting interferometry [31-33]. The phase-shifting interferometry applies a sequence of static phase shifts into the light beam to generate a sequence of phase-shifted fringe patterns to measure the profile of smooth surfaces [33]. Since the applied phase-shift is quasi-static, it does not change the frequency band of the measurement. Contrarily, the dynamic frequency modulation techniques change the temporal frequency of the laser beam continuously so that the optical phase of interest is measured in the beat frequencies that are resulted from this dynamic frequency change. This later technique is not as sensitive to low frequency environmental noise as the phase-shifting interferometry.

Deep frequency modulation-based interferometers combine strong sinusoidal laser frequency modulation in unequal arm length interferometers with a non-linear fit algorithm for phase extraction. The demodulation algorithm or the non-linear fit is implemented in a FPGA to miniaturize the system [28]. Extrinsic Fabry-Perot interferometer uses self-homodyning (also used in our method) by sinusoidal frequency modulation of the laser source wavelength to achieve sub-nanometer resolution displacement measurement in a compact sensing volume. Application of compact interferometers in simple position sensors and high-resolution inertial sensors has been recently reviewed in Ref. [34]. This review provides a summary of variety of optical interferometry techniques and their pros and cons in terms of measurement linearity and resolution, compactness of the interferometer, cost, and finally their inherent susceptibility to shot noise and length noise that couples to their readout [34].

Two methods to generate frequency or phase modulated laser light are: 1) by applying a modulation directly to the light source (direct method), 2) by passing the beam through a frequency shifter (indirect method). Optical serrodyne frequency translation [35] used in LiDAR velocimetry [6] and in optical gyroscopes [7] are examples of an indirect frequency shifting method, whereas changing the drive current through a laser diode to produce FM light is a direct method [36, 37].

1.2 Objectives of this research

There is always need for low cost, portable displacement measurement interferometers. The main objective of this research is to design and implement a low cost, portable displacement measurement interferometer with sub 50 nm resolution and measurement speed of about 10 mm·s^{-1} for metrology applications. This is achieved by generating a stabilized FM light source for interferometry. This method combines the homodyne and heterodyne interferometry detection bandwidth to extract the phase information from harmonics of modulation frequencies at an update rate of a few hundred kHz. Therefore, it does not require ultrafast flash photodetectors and ultrahigh-speed electronics resulting in a substantial drop in the final cost of the system.

We have developed a new method to transfer the precision of atomic clocks to dynamic frequency modulation interferometry. This method that is being named as polydyne interferometry technique comprises three main parts: 1) generating a synchronous RF-FM electrical signal, 2) transferring the FM signal into a laser beam using an AOM, and 3) signal detection and processing. The advantages of this approach are that measurement for any single degree of freedom requires only one detector, ultrahigh-speed ultrafast flash detectors and polarization optics are unnecessary, and all the frequency components are synchronous and they all follow the same path around the interferometer paths. More broadly the ability to create multiple, phase-synchronous reference beams enables an arbitrary large number of synchronous interference measurements. Using only a single AOM, multiple diffraction orders are relatively straightforwardly extracted. Each of these diffraction orders contain the same modulation frequency harmonics.

CHAPTER 2 INTERFERENCE OF FM LIGHT BEAMS

Interference of FM waves are the foundation of the polydyne interferometry technique presented in this thesis. In unequal arm length interferometers such as a Michelson interferometer, the resultant optical signal at the photo diode incorporates the superposition of the beams travelling through the short and the long arm that have an optical path difference of ΔL (corresponding to a time delay of $\tau = \Delta L/c$, where c is the speed of light in the medium). Assuming an FM light with carrier frequency ω_0, amplitude E_0, and modulation frequency ω_m that is split by a 50/50 beam splitter to produce the signals of each arm of a Michelson interferometer, u_1 and u_2:

$$\frac{u_1(t)}{E_0/2} = \sin\left(\omega_0 t - \beta\cos\left(\omega_m t\right) + \phi_0\right) = -\frac{i}{2}\left\{e^{i\left[\omega_0 t - \beta\cos(\omega_m t) + \phi_0\right]} - e^{-i\left[\omega_0 t - \beta\cos(\omega_m t) + \phi_0\right]}\right\}, \tag{2-1}$$

$$\begin{aligned}\frac{u_2(t)}{E_0/2} &= \sin\left(\omega_0\left(t-\tau\right) - \beta\cos\left(\omega_m\left(t-\tau\right)\right) + \phi_0\right)\\ &= -\frac{i}{2}\left\{e^{i\left[\omega_0(t-\tau) - \beta\cos(\omega_m(t-\tau)) + \phi_0\right]} - e^{-i\left[\omega_0(t-\tau) - \beta\cos(\omega_m(t-\tau)) + \phi_0\right]}\right\}\end{aligned}, \tag{2-2}$$

where, β is the modulation depth, ϕ_0 is the initial phase of light, phase @ $t = 0$, at the photo diode. The intensity of optical signal produced in the photo diode, $I\left(t,\tau\right)$, is

$$\frac{I(t,\tau)}{E_0^2/4} = \frac{\left(u_2(t)+u_1(t)\right)^2}{E_0^2/4} = \begin{pmatrix} \sin\left(\omega_0(t-\tau)-\beta\cos\left(\omega_m(t-\tau)\right)+\phi_0\right) \\ +\sin\left(\omega_0 t-\beta\cos\left(\omega_m t\right)+\phi_0\right) \end{pmatrix}^2 . \qquad (2\text{-}3)$$

Assuming the initial phase equal to zero, $\phi_0 = 0$, the intensity of optical signal is

$$\begin{aligned} \frac{I(t,\tau)}{-E_0^2/2} &= \begin{pmatrix} \left(e^{i\left[\omega_0 t-\beta\cos(\omega_m t)\right]} - e^{-i\left[\omega_0 t-\beta\cos(\omega_m t)\right]}\right)\Big/2 \\ +\left(e^{i\left[\omega_0(t-\tau)-\beta\cos\left(\omega_m(t-\tau)\right)\right]} - e^{-i\left[\omega_0(t-\tau)-\beta\cos\left(\omega_m(t-\tau)\right)\right]}\right)\Big/2 \end{pmatrix}^2 \\ &= \begin{pmatrix} \left(e^{2i\omega_0 t}e^{-2i\beta\cos(\omega_m t)} + e^{-2i\omega_0 t}e^{2i\beta\cos(\omega_m t)} - 2\right)\Big/4 \\ +\left(e^{2i\omega_0(t-\tau)}e^{-2i\beta\cos\left(\omega_m(t-\tau)\right)} + e^{-2i\omega_0(t-\tau)}e^{2i\beta\cos\left(\omega_m(t-\tau)\right)} - 2\right)\Big/4 \\ +\left(e^{2i\omega_0 t}e^{-i\omega_0\tau}e^{-i\beta\cos(\omega_m t)}e^{-i\beta\cos\left(\omega_m(t-\tau)\right)} + e^{-2i\omega_0 t}e^{i\omega_0\tau}e^{i\beta\cos(\omega_m t)}e^{i\beta\cos\left(\omega_m(t-\tau)\right)}\right)\Big/2 \\ -\left(e^{i\omega_0\tau}e^{-i\beta\cos(\omega_m t)}e^{i\beta\cos\left(\omega_m(t-\tau)\right)} + e^{-i\omega_0\tau}e^{i\beta\cos(\omega_m t)}e^{-i\beta\cos\left(\omega_m(t-\tau)\right)}\right)\Big/2 \end{pmatrix} . \end{aligned} \qquad (2\text{-}4)$$

The intensity equation (2-4) represents some fluctuations at frequencies of $2\omega_0 \pm p\omega_m$, in which p is an integer number, and other fluctuations at harmonics of ω_m. In practice, fluctuations at $2\omega_0 \pm p\omega_m$ occur too fast to measure so that they average out to zero, after which the measured intensity reduces to

$$\begin{aligned} I(t,\tau) &= \frac{E_0^2}{2}\left(1+\left(e^{i\omega_0\tau}e^{-i\beta\cos(\omega_m t)}e^{i\beta\cos\left(\omega_m(t-\tau)\right)} + e^{-i\omega_0\tau}e^{i\beta\cos(\omega_m t)}e^{-i\beta\cos\left(\omega_m(t-\tau)\right)}\right)\Big/2\right) \\ &= \frac{E_0^2}{2}\left[1+\cos\left(\omega_0\tau+\beta\cos\left(\omega_m\tau\right)-\cos\left(\omega_m(t-\tau)\right)\right)\right] \\ &= \frac{E_0^2}{2}\left[1+\cos\left(\omega_0\tau+\beta\sin\left(\omega_m\tau/2\right)\sin\left(\omega_m(t-\tau/2)\right)\right)\right] \end{aligned} . \qquad (2\text{-}5)$$

For typical metrology applications, the length of both arms of the interferometer is, at most, of the orders of a meter, from which $\tau \approx 1\text{m}/c = 1/299792458 \approx 3.3\times10^{-9}$ s.

Therefore, for $\omega_m \approx 10^5$ Hz, $\omega_m \tau \approx 3.3\times10^{-4} << 1$ so that $\sin\left(\omega_m \tau/2\right) \approx \omega_m \tau/2$ and $\sin\left(\omega_m\left(t-\tau/2\right)\right) \approx \sin\left(\omega_m t\right)$; this leads to the simplified form of measured irradiance:

$$\begin{aligned} I\left(t,\tau\right) &= \frac{E_0^2}{2}\left[1+\cos\left(\omega_0\tau+\beta\tau\sin\left(\omega_m t\right)\right)\right] \\ &= \frac{E_0^2}{2}\left[1+\cos\left(\omega_0\tau\right)\cos\left(\beta\tau\sin\left(\omega_m t\right)\right)-\sin\left(\omega_0\tau\right)\sin\left(\beta\tau\sin\left(\omega_m t\right)\right)\right] \end{aligned} . \tag{2-6}$$

Setting $\gamma = \beta\tau$ as the effective modulation depth, $\phi = \omega_0 \tau$ as displacement-related optical phase change, and using Jacobi-Anger expansion [38, 39] for $\sin\left(\gamma\sin\left(\omega_m t\right)\right)$ and $\cos\left(\gamma\sin\left(\omega_m t\right)\right)$ gives

$$I\left(t,\tau\right) = I_0 \begin{bmatrix} 1+\cos\left(\phi\right)\left(J_0\left(\gamma\right)+2\sum_{p=1}^{\infty} J_{2p}\left(\gamma\right)\cos\left(2p\omega_m t\right)\right) \\ +\sin\left(\phi\right)\sum_{p=1}^{\infty} -2J_{2p-1}\left(\gamma\right)\cos\left(\left(2p-1\right)\omega_m t\right) \end{bmatrix} . \tag{2-7}$$

In which, $J_p\left(\gamma\right)$ are the Bessel functions of first kind, $I_0 = E_0^2/2$ is the average intensity on the detector, $I_0 J_0\left(\gamma\right)\cos\left(\phi\right)$ is the homodyne equivalent intensity that will be equal to $I_0\cos\left(\phi\right)$ for $\gamma = 0$ as it is expected from the intensity of unmodulated signals interference, $\sum_{p=1}^{\infty} J_{2p-1}\left(\gamma\right)\cos\left(\left(2p-1\right)\omega_m t\right)$ is the odd harmonics of the modulation frequency whose amplitude changes with odd part of the phase, and $\sum_{p=1}^{\infty} J_{2p}\left(\gamma\right)\cos\left(2p\omega_m t\right)$ is the even harmonics of the modulation frequency whose amplitude changes with even part of the phase. Equivalent expressions have been presented in different studies [30, 40] in which all of them have used the same simplifications to derive an expression for the harmonics signals. Increasing the effective modulation depth, γ, spreads the energy of interference among the harmonics of modulation frequency. This can be observed in the absolute value of Bessel functions of first kind, $J_i(\beta)$, for i = 0, 1, … , 8, see Figure 2-1.

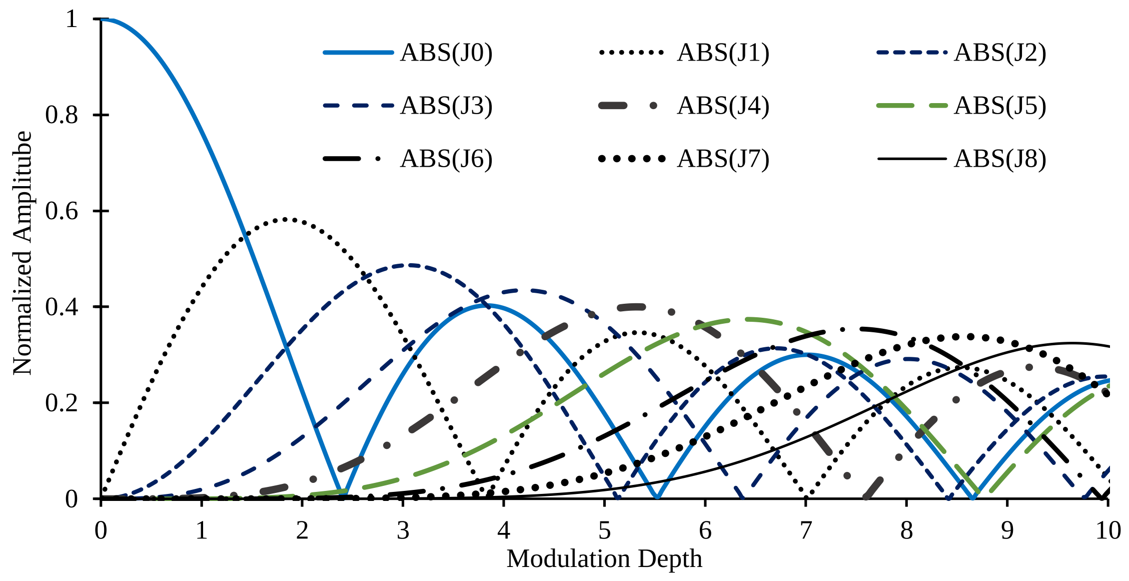


Figure 2-1- Absolute value of Bessel functions of first kind, $J_i(\beta)$, for i = 0, 1, … , 8 in a normalized scale.

The amplitudes of the odd and even harmonics are commonly called in-phase, I_{2p-1}, and quadrature, I_{2p}, signals, respectively. Applying Fourier transform to the measured intensity, equation (2-7), results in following expressions for the in-phase and quadrature signals from which phase can be extracted [41, 42]:

$$\begin{aligned} I_{2p-1} &\approx I_0 J_{2p-1}(\gamma)\sin(\omega_0\tau) \\ I_{2p} &\approx I_0 J_{2p}(\gamma)\cos(\omega_0\tau) \end{aligned} \quad . \qquad (2\text{-}8)$$

In order to extract the phase from the measured in-phase and quadrature signals, these signals are first normalized and phase-adjusted to fit in a unity circle after which the phase is calculated as

$$\phi = \tan^{-1}\left(\frac{I_{2p-1,u}}{I_{2p,u}}\right), \qquad (2\text{-}9)$$

where, $I_{2p-1,u}$ and $I_{2p,u}$ are the normalized, phase adjusted I_{2p-1} and I_{2p}, respectively. Finally, the displacement of the moving arm of the interferometer is derived from the measured phase as

$$\Delta L = \frac{\phi\lambda}{2\pi nN} \ . \qquad (2\text{-}10)$$

All the equations presented in this chapter represent the ideal condition of perfect FM signals interference. If the interfering beams are not perfect FM waves, the distribution of energy in different frequency components will be different from equation (2-7). In such circumstances, each of the odd and even harmonics of the modulation frequency might carry both $\sin(\omega_0\tau)$ and $\cos(\omega_0\tau)$. This provides the opportunity to calculate the optical phase information, $\phi = \omega_0\tau$, by extracting the amplitude of sine and cosine of each of the modulation signal harmonics. The proof of this hypothesis is left as a suggestion for future work.

CHAPTER 3 SYNCHRONOUS FM SIGNAL GENERATION

This chapter consists of four sections that present a summary of literature about digital FM signal generation methods in section 3.1, the principle operation mechanism of a new FM signal generation method introduced in this work in section 3.2, the new FM circuit design in section 3.3, and the testing and results of spectrum of generated FM signals by different prototypes of the new FM signal generator in section 3.4.

3.1 Digital FM signal generation background

Frequency modulation in instrumentation and communication include a broad range of sensing strategies and information transmission methods. Such strategies range from single frequency transmission and reception with amplitudes representing the DC value, through to phase/frequency modulation and phase locking and, ultimately, multiple phase and frequency transmission for synchronous, parallel, digital data transfer. All of the modulation techniques share the main concept through which signals are mixed to operate at more conveniently measured harmonic frequencies and intermediates as is the case for heterodyne and super-heterodyne techniques [43]. Such processes find applications in both optical processes and radio transmission [43-45]. Of particular recent interest is the implementation of these modulation techniques for development of optical interferometric instrumentation [24, 26, 46]. In these techniques, the interferometric phase is extracted using multiple sidebands which has led to novel sensing methods.

Until the last few decades, most radio frequency modulators were implemented by analog circuits. Since digital phase/frequency modulators are compact and often provide superior frequency stability, they have now evolved as replacements for the bulky analog circuits. One such, digital approach, are Direct Digital Synthesizers (DDS) that provide low distortion signals with frequencies exceeding 400 MHz, a limit that is continuously increasing [47]. The method used in commercial DDS chips to generate modulated signals is based on regularly updating a phase offset word in a look-up table via a serial port [48-50] and is limited by the speed of the port. Furthermore, this method does not necessarily lead to synchronous FM signals, since the serial port is not necessarily timed with the main clock of the system. Recent studies have been focused on high speed DDS technology with FM capabilities over GHz frequencies [51-57]. One such development is the DDS that has been manufactured in Ref. [58] as the first DDS that generates FM signals at frequencies above 1 GHz. The aim reported in Ref. [58] is to provide all functionality of over-GHz FM signal generation in a single microchip. Their DDS architecture uses carry-look-ahead or ripple-carry-adder to boost the speed of registering frequency control word of a precursor design that uses bipolar plus negative metal oxide semiconductor adder architecture [59], and uses a 5 GHz clock to time the data transfer buses. This DDS has been designed to be used in radar technology where the best spurious free dynamic range at higher frequencies, i.e. over-GHz frequencies, is the main functional figure of merit. Also, this DDS provides 38 dBc Nyquist band and 83 dBc narrowband spurious-free dynamic range for over GHz carrier frequencies, see Figure 3-1 [59].

Another research group has produced an all-digital, direct frequency synthesizer that provides direct modulation capabilities at 20 Mbps with 3 μs startup latency [60]. In this work, they have claimed that a synthesizer can generate desired frequencies over a wide range by digitally manipulating time-shifted copies of a temperature-compensated film bulk acoustic resonator signal based on the outputs from a sigma-delta modulator. Performance of a working circuit has not been reported yet.

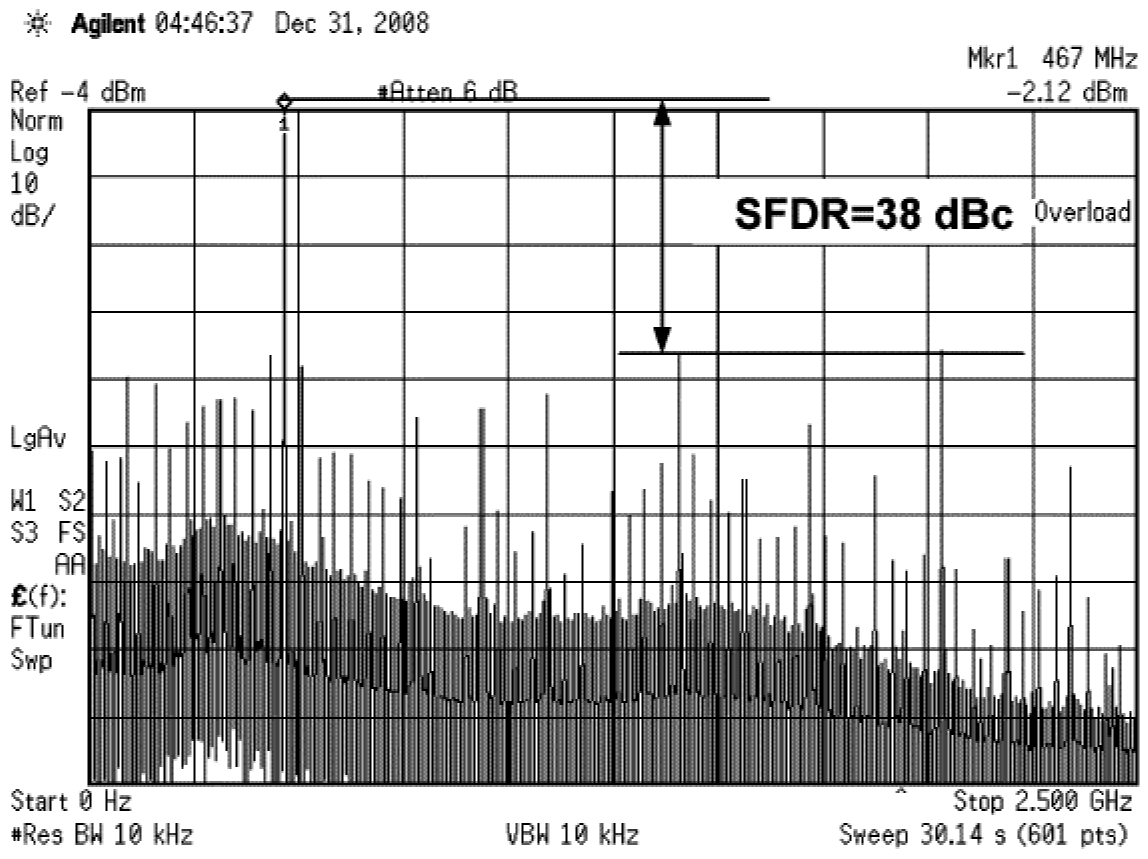


Figure 3-1- Frequency spectrum of DDS output with a 469.360351 MHz output showing a 38 dBc Nyquist band spurious-free dynamic range [59].

In principle, it is possible to generate a phase modulated signal by dynamically updating the phase offset control of some commercial DDS chips (such as AD9834, and the AD9951 used in our design). Phase Shift Keying (PSK) and Frequency Shift Keying

(FSK) are two common data encoding methods that are used in these DDS architectures for modulating the phase and frequency of the desired output signal, respectively. In PSK, the phase of the transmitted signal is changed, while the frequency of the carrier remains constant. FSK allows a binary shifting of the frequency of a continuous carrier to one or the other of two discrete frequencies. Two frequency registers are required to achieve a fast tuning into these two frequencies as it is used in AD9834 [50]. The range and precision of modulation by FSK and PSK is inherently limited, primarily by the speed of the serial peripheral interface and the accuracy of timing for triggering FSK and PSK registers. This limits the achievable values of the modulation depth so that side-band amplitudes of only about -40 dB are achievable for different modulation frequencies, while the noise level is about -70 dB to -80 dB.

Early attempts to create frequency modulated signals using the methods outlined in the previous paragraph experienced numerous problems in dynamically updating the phase accumulator of commercial DDS boards to generate phase/frequency modulated electric signal so that our efforts resulted in a synchronous FM signal with spurious glitches that could not be removed. Therefore, we designed and manufactured a novel digital FM signal generator that produces electrical FM signal in which the modulation frequency is phase synchronized with the carrier frequency. This has the benefit of phase synchronization in all generated frequencies using an atomic clock. This method uses two DDS chips, one for generating the carrier frequency and the other one for producing the modulation through an analog time-delay circuit. This removes the necessity to update the phase offset word and eliminates the limitation of serial data transfer speed, though the modulation and carrier frequencies are limited by the frequency limits of the corresponding DDS chips. This FM signal generator also solves another common problem of noise spikes in the generated signals caused by dynamic effects of the register updates. Using this particular method, frequencies of carrier and harmonics are not influenced by the analog circuit variations; however, drifts in the delay circuit frequency will affect relative amplitudes of the harmonics either side of the carrier, i.e. the modulation depth. Since the electrical FM signal is aimed to drive AOMs with 60 and 70 MHz operation frequencies, the manufactured FM signal generator was designed to provide an output frequency band of 1 Hz to 80 MHz. This is achieved by two stages of passive and active low-pass filters at the output of the board outlined in section 3.3.

3.2 Principle of operation

3.2.1 Modulated clocking of the high bandwidth DDS

DDS microchips comprise a processor that uses a pointer to increment through a Look-Up Table (LUT) to determine values that are sent to a Digital to Analog Converter (DAC) through which the analog signal is generated. Regularly clocking the pointer at equal time intervals as it steps through a LUT of sinewave values results in a digital output equivalent to the function represented in the LUT. Dependent on the values in the LUT and the frequency of the clock, periodic shapes of any function can be produced. Consequently, to produce a sinewave, a DDS is implemented using a reference clock, an address counter, a digital sine LUT, and a DAC. An amplitude word is picked from the sine LUT by the address counter and is sent to the DAC to generate the equivalent analog sinewave. The output frequency depends on the clock frequency and the LUT jumper value set by the user.

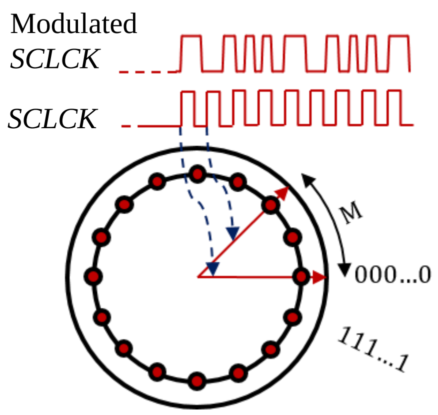


Figure 3-2- Digital Phase Wheel with corresponding normal and modulated clock.

A “phase wheel” representation shown in Figure 3-2 illustrates the DAC transfer process in a DDS. As shown, generation of sinewave oscillation can be visualized as a vector rotating around a circle containing values for the DAC. Each designated point on the phase wheel corresponds to data that transforms to an equivalent point on a cycle of a sinewave. Each revolution of the vector around the phase wheel, at a constant speed, generates one complete cycle of the sinewave the frequency of which is determined by the basic tuning equation

$$f = (M \times CLCK)/(2^{Q}). \qquad (3\text{-}1)$$

where, f is the output frequency; M, the register vector jump size; Q is the bit resolution of the LUT; 2^{Q} is the number of data-points in the LUT and *SCLCK* is the system clock frequency inside the DDS. In some DDS microchips, the external clock is first multiplied by an integer value through a multiplier to generate higher frequency clocks for processing the data inside the DDS structure. In such DDS architectures, the *SCLCK* is the frequency of clock signal after multiplier.

The main idea behind the electrical FM signal generator presented in this study is that a DDS will output frequency/phase modulated sinewave if the phase of its clock signal is dynamically being adjusted. The FM signal generator developed here uses two DDS processors: one for adjusting the phase of the clock signal and the other one for generating the FM signal from the modulated clock. To do this, a voltage-controlled time-delay circuit is adopted to modulate the reference clock frequency before feeding it into the high frequency DDS, DDS2, also indicated in Figure 3-2. Figure 3-3 shows a block diagram indicating the major components of this method. DDS1 is used to generate a sinusoidal voltage that controls the time-delay circuit. Importantly, the clock for both DDS1 and DDS2, are derived from the same source, in our experiments a 10 MHz rubidium oscillator (SRS model PRS10 [61]).

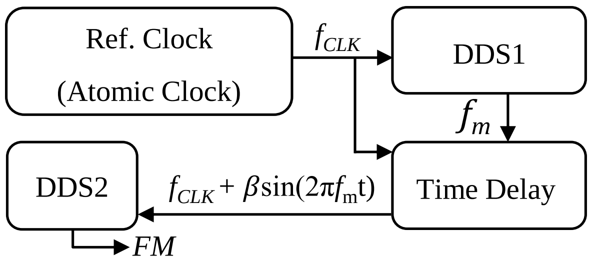


Figure 3-3- Block diagram of the circuit containing the signal path from reference clock to the DDS2.

As seen in Figure 3-3, the reference clock is simultaneously timing both DDS circuits. Being directly timed from the clock source, the DDS1 output signal is used to modulate the reference clock signal via the time-delay sub-circuit. The output of the time-delay circuit is then used as a clock for the DDS2 circuit. Definitions of the mathematical parameters in this figure are discussed in more detail in the following section. To achieve clock speeds suitable for RF generation, the 10 MHz clock signal is multiplied by 20 (or any integer from 4 to this value) through a Phase Locked Loop (PLL) multiplier within the DDS2 chip. Because the multiplier output, *SCLCK*, follows the phase of the input clock through the PLL, it is assumed that the phase modulations, being substantially slower than the 10 MHz clock signal, will be similarly transmitted.

3.2.2 Theory of Synchronous FM Signal Generation

As shown in Figure 3-2 (page 19), each rising edge of the clock signal triggers the LUT to send a digital value to the DAC through which an analog equivalent is generated. For a constant clocking speed, this results in a perfect sine wave. Under such an assumption, the sine wave at the DDS output is

$$V\left(t_r\right)=V_0\sin\left(\omega_c t_r\right)=V_0\sin\left(\omega_c r\Delta t_0\right), \qquad (3\text{-}2)$$

where, V_0 is the amplitude of the output signal; ω_c, the angular frequency of the signal that is equal to $2\pi f_c$; r, an integer number increasing by each pulse of the *SCLCK* (i.e. a clock count); and Δt_0 is the clock timing interval. For the modulated reference clock, the instantaneous timing interval reads as:

$$\Delta t=\Delta t_0+\frac{\beta}{r\omega_c}\sin\left(\omega_m r\Delta t_0\right), \qquad (3\text{-}3)$$

where, ω_m is the angular modulation frequency, and β is the modulation depth. Substituting Δt_0 in (3-2) with Δt gives

$$V\left(t_r\right)=V_0\sin\left[\omega_c r\Delta t_0+\beta\sin\left(\omega_m r\Delta t_0\right)\right], \qquad (3\text{-}4)$$

where, $r\Delta t_0$ is the quantized time approximating to a continuous parameter, t_r, as Δt_0 approaches zero. Equation (3-4) represents an FM signal with corresponding carrier frequency of $f_c = \omega_c / 2\pi$, modulation frequency of $f_m = \omega_m / 2\pi$, and modulation depth β. For the purpose of computing frequency content, this FM signal, $V(t)$, can be represented in the complex exponential form:

$$V(t)/V_0 = \sin\left(\omega_c t + \beta \sin\left(\omega_m t\right)\right) = i\left[e^{i\left(\omega_c t + \beta\sin\left(\omega_m t\right)\right)} - e^{-i\left(\omega_c t + \beta\sin\left(\omega_m t\right)\right)}\right]\Big/2. \qquad (3\text{-}5)$$

Equation (3-5) can be rewritten in terms of Jacobi-Anger expansions in the form of following series [38, 39]

$$e^{-i\left(\omega_c t + \beta\sin\left(\omega_m t\right)\right)} = e^{-i\omega_c t} J_0(\beta) + \sum_{p=1}^{\infty}\left(\left(-1\right)^p e^{-i\left(\omega_c - p\omega_m\right)t} + e^{-i\left(\omega_c + p\omega_m\right)t}\right) J_p(\beta), \qquad (3\text{-}6)$$

$$e^{i\left(\omega_c t + \beta\sin\left(\omega_m t\right)\right)} = e^{i\omega_c t} J_0(\beta) + \sum_{p=1}^{\infty}\left(e^{i\left(\omega_c + p\omega_m\right)t} + \left(-1\right)^p e^{i\left(\omega_c - p\omega_m\right)t}\right) J_p(\beta), \qquad (3\text{-}7)$$

If the signal phase is subject to a single modulation frequency, the modulation depth is a constant. Hence the output comprises the original carrier frequency plus an infinite series of sidebands at integer multiples of the modulation frequency. In practice, the number of significant modulation sidebands in the output spectrum is a function of the modulation depth [43, 62]. By taking the Fourier transform of (3-6) and (3-7), a discrete FM output spectrum with magnitude coefficients as a function of β is seen as presented in the equation below.

$$x_{FM} = \left(V_0/2\right) \times \sum_{p=0}^{\infty} J_p(\beta)\,\delta\left(f - \left(f_c \pm p f_m\right)\right), \qquad (3\text{-}8)$$

where, $\delta(-)$ is the Dirac delta.

3.3 Modulator Circuit Design

As stated, the modulation process requires a voltage-controlled time-delay circuit to generate controlled delays in the phase of a sinusoidal signal from a 10 MHz oscillator that is subsequently used to time a high frequency DDS circuit. The controlling voltage signal, which acts to modulate the time delay, is generated by a second DDS that is synchronously timed by the same, but unmodulated, 10 MHz carrier signal.Both the carrier and modulation harmonics are consequently timed by the same clock signal. The time-delay circuit is shown in Figure 3-4. This consists of a wide bandwidth JFET input operational amplifier model LF353 [63], an ultrafast precision comparator model LT1016 [64], and a retriggerable mono-stable multi-vibrator model SN74LS123 [65].

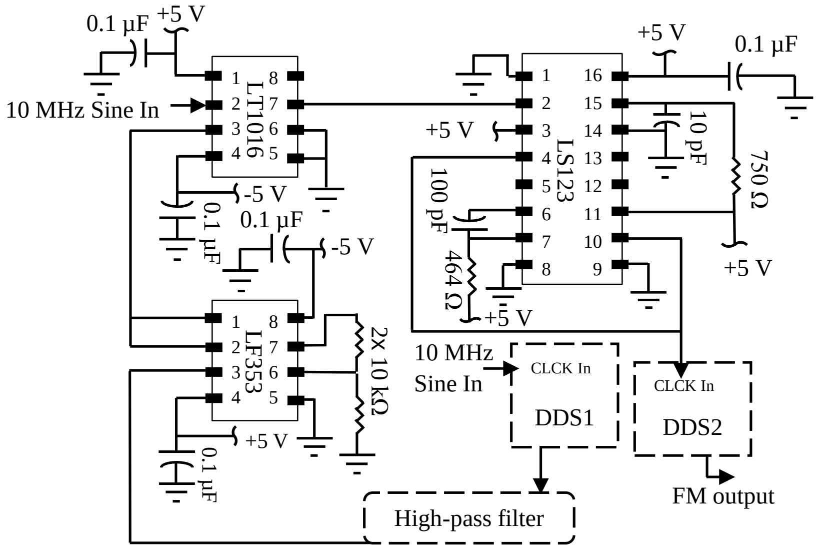


Figure 3-4- Details of the phase delay circuit in conjunction with major blocks of other DDS1 and DDS2 circuits.

The modulation signal coming from DDS1 is amplified to the desired amplitude and then applied to the voltage-controlled delay. There is a buffer between the DDS1 and the comparator of the time-delay circuit to eliminate possible loading effects due to impedance mismatch between them. The harmonically varying delay results in a dynamic shift in the reference clock going into the comparator. The net result of this is to produce a voltage-controlled pulse width generator where the pulse width is proportional to the modulation voltage from the DDS1. Finally, the output signal of the comparator is used to trigger the multi-vibrator. Since the commercial atomic clocks typically generate 10 MHz signal, the phase delay is designed to be linear with input voltage at this frequency.

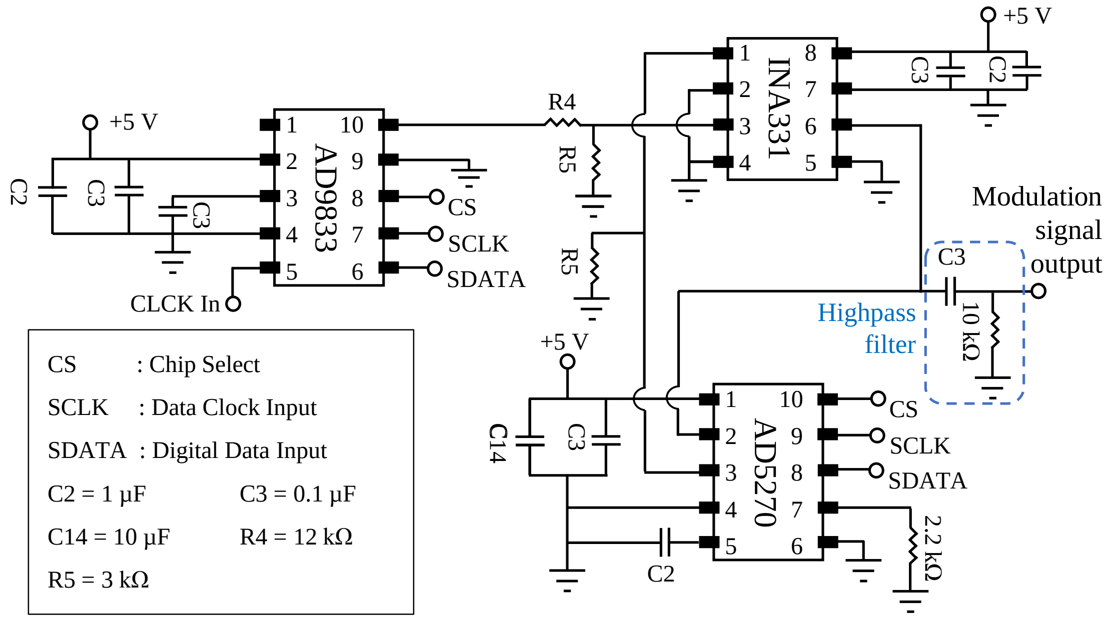
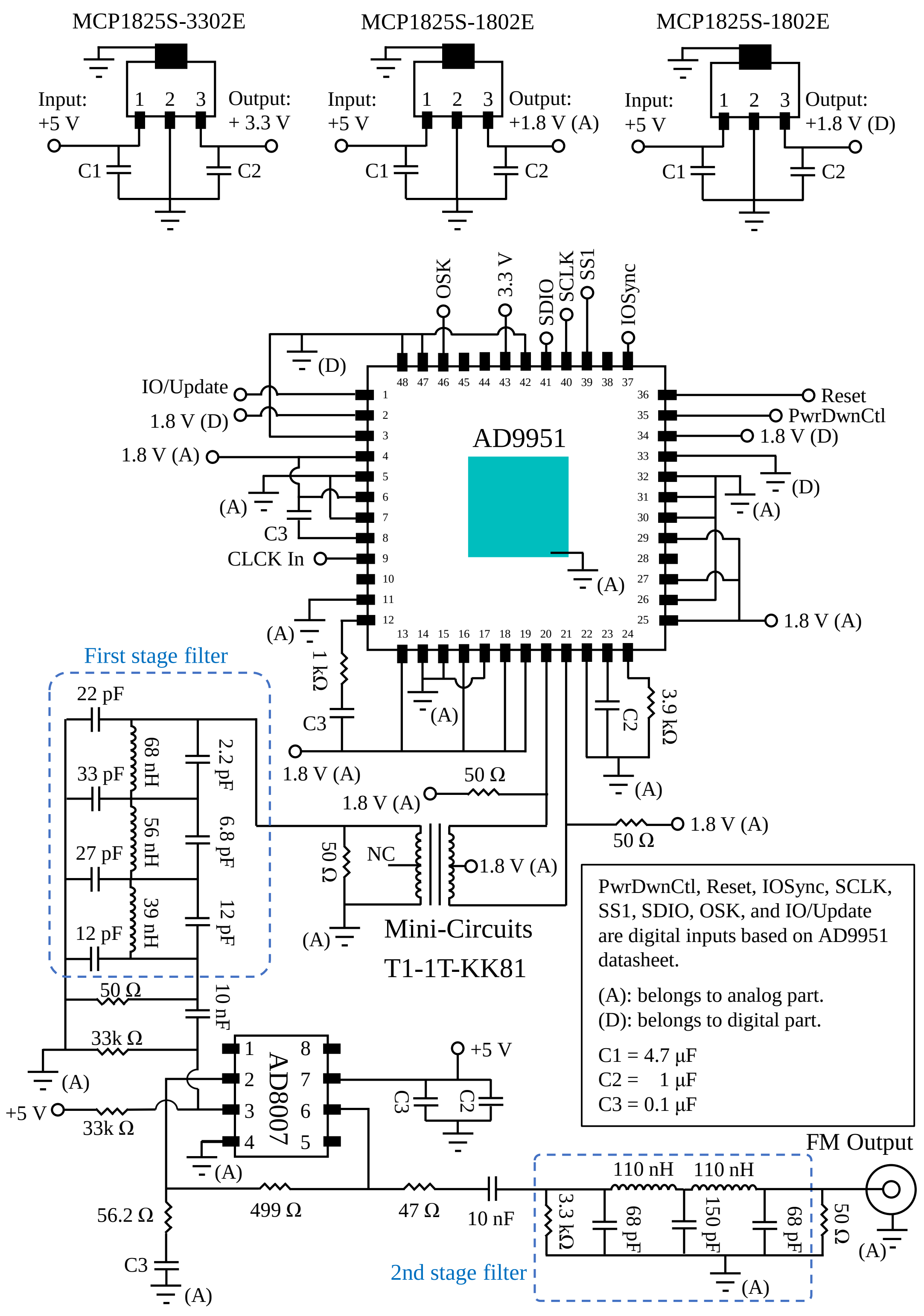


Figure 3-5- Schematic of the DDS1 circuit with a digital amplitude control.

The DDS1 circuit is shown in Figure 3-5. In the first prototype of the FM board, a potentiometer was used to control the amplitude of the modulation signal, V_m. This potentiometer was replaced with a digital rheostat, model AD5270 [66], to provide full-digital control over all the characteristics of the generated FM signal, see Figure 3-5. Due to the linear relationship between the delay generated in the time-delay circuit and V_m, this will subsequently control the modulation depth, β, equation (3.3). The modulation depth changes the energy distribution among the sidebands of the FM signal. This can be observed in the frequency content and relative side-band amplitudes based on Jacobi-Anger expansion. The high-pass filter between DDS1 and the time-delay circuit, shown in Figure 3-4, is a simple first-order RC filter located at the output of the DDS1 circuit in Figure 3-5. This filter eliminates the DC offset at the output of the DDS1 circuit so as to make it compatible with the input voltage range of the time-delay circuit as it is discussed in the testing section of the sub-circuits of the board. This also provides an AC-couples modulation signal that is compatible with the reference inputs of lock-in amplifiers.

The DDS2 circuit is shown in Figure 3-6. The DDS2 chip, AD9951, requires two separate 1.8 V supplies for analog and digital parts that were provided by small surface-mount, fixed-voltage regulators, model MCP1825S, made by Microchip Technology Inc [67]. Acceptable signal qualities were achieved by a two-layer board with separate digital and analog ground plates, and separate power supplies via fixed-voltage regulators. To decrease the noise in the circuit, the impedance of the traces used in the Printed Circuit Board (PCB) is matched to the required impedance for VHF signals (50 Ω). To further eliminate impedance mismatching that can attenuate and distort the signal and introduce noise, SubMiniature version A (SMA) connectors are used. For the best spectral purity possible, two stages of filtering are included on the output of the DDS2 circuit, shown in Figure 3-6. The first stage is a shunt-connected, 7th order, elliptic low-pass filter with a cutoff frequency of 160 MHz with component values based on a commercial DDS2 evaluation board [68]. The second stage of filtering is a π configuration, 5-pole, Butterworth low-pass filter with a cutoff frequency of 73 MHz [69].



PwrDwnCtl, Reset, IOSync, SCLK, SS1, SDIO, OSK, and IO/Update are digital inputs based on AD9951 datasheet.

(A): belongs to analog part.
(D): belongs to digital part.

C1 = 4.7 μF
C2 = 1 μF
C3 = 0.1 μF

Figure 3-6- Schematic of the DDS2 circuit.

3.4 FM signal generator testing and results

3.4.1 Experimental apparatus and test procedure

Since the modulator combines three main circuits of DDS1, time-delay, and DDS2 circuits, prior to evaluating the complete modulator, it is necessary to evaluate the performance of each of these individual sub-circuits. To determine time delay as a function of the input voltage, the DDS1 was bypassed and a DC voltage applied to pin 3 of the LF353 microchip shown in Figure 3-4. With a 10 MHz signal going into the circuit, the delay through the comparator versus different voltages was measured using an oscilloscope, see Figure 3-7. As observed in Figure 3-7, the generated delay varies linearly with the input voltage. Furthermore, it is seen that the delay circuit becomes nonlinear for input-voltages outside the range -2 V to 2 V, thereby limiting the amplitude of the modulation signal generated by DDS1. Over this range the response of the time delay circuit is -5.207 ns.V^{-1} with a coefficient of determination of 0.9994.

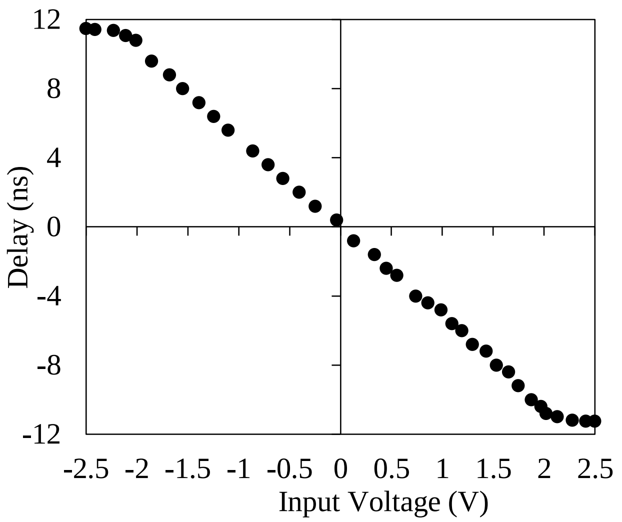


Figure 3-7 Variation of the generated delay from the LT1016 comparator as a function of the voltage applied to the LF353 amplifier.

The amplitude frequency responses of the DDS1 and DDS2 circuits as they are clocked by constant 10 MHz signal are shown in Figure 3-8 and Figure 3-9, respectively. The magnitude of DDS1 output attenuates at frequencies greater than 1 MHz. It also has a DC offset of about 2V. To eliminate this DC offset, a high-pass filter with a cutoff frequency of 1 kHz is used at its output, as shown in Figure 3-5. The two stages of low pass filters shown in Figure 3-6 provide a cut-off frequency of about 73 MHz. This cut-off frequency is shown in Figure 3-9.

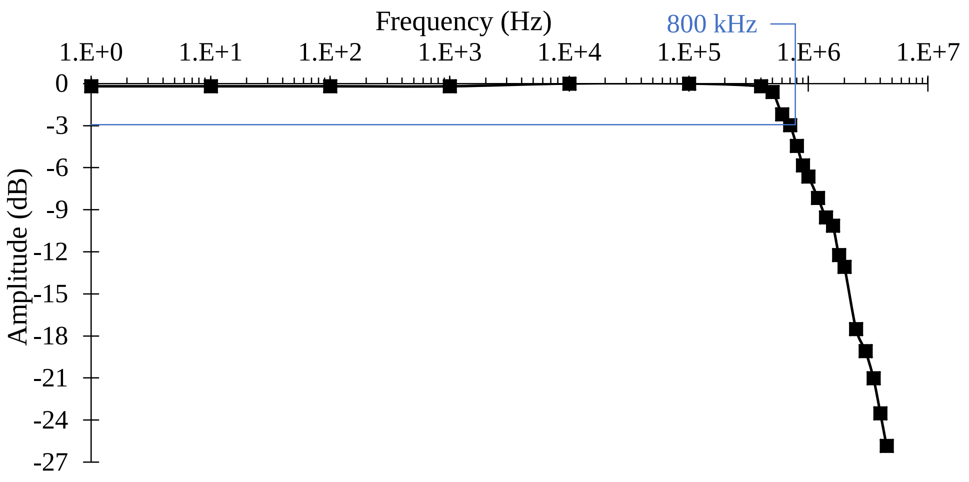


Figure 3-8- Frequency response of the DDS1 circuit over a 2 MHz band.

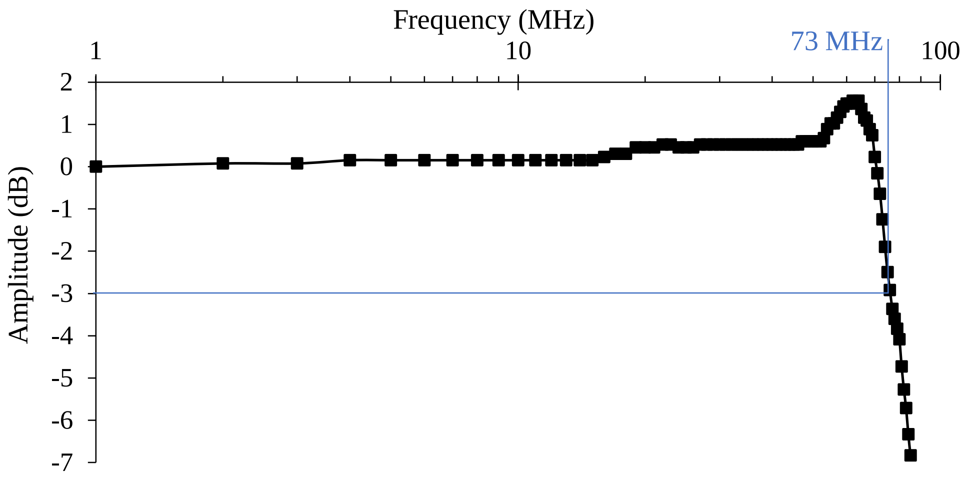


Figure 3-9- Frequency response of the DDS2 circuit from 1 to 80 MHz.

3.4.2 Performance tests on first prototype of FM board

Having designed and implemented the FM circuit, see Figure 3-10, an experiment was set up using an Arduino Mega™ microcontroller to program the microchips, and a rubidium oscillator (SRS model PRS10) for the clock, see Figure 3-11. This rubidium frequency standard provides a low phase noise of <-130 dBc/Hz at 10 MHz and a short-term stability of $<2\times10^{-11}$ s. Because this provides the clock signal for both DDS chips, the frequency components of all generated FM signals are synchronized to this reference. The frequency components of the FM signals are measured using a spectrum analyzer (Agilent CSA model N1996A-506). This analyzer has a frequency range up to 6 GHz with resolutions settable from 10 Hz to 5 MHz. The Arduino code used for controlling the DDS microchips are listed in APPENDIX A.

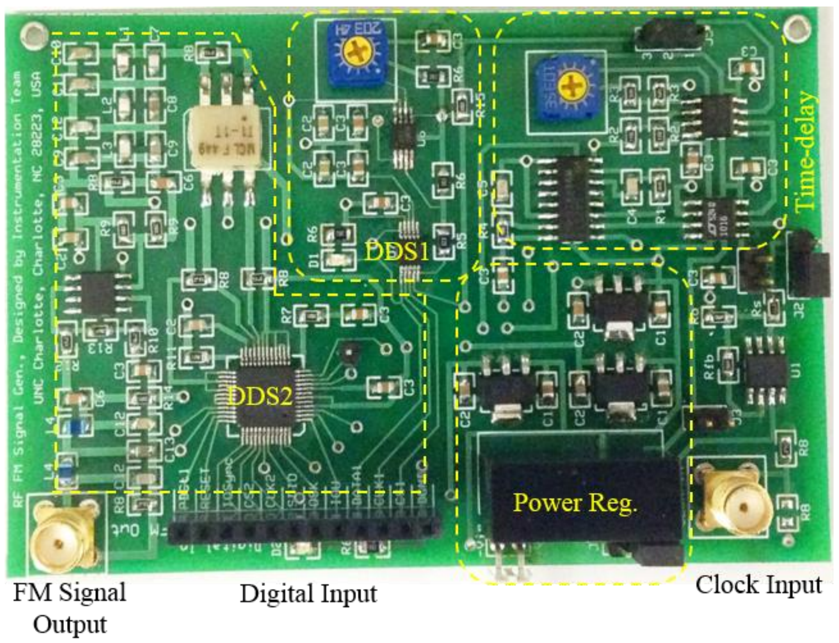


Figure 3-10- First prototype of the FM board.

The FM board shown in Figure 3-10 has only one high-pass filter between the atomic clock and the clock input of the board without any non-inverting amplifier. It also uses a potentiometer instead of a digital rheostat. Furthermore, this board uses a power regulator at its power input section to generate ±5 V DC lines from the supplied 5 V DC power. These are the main differences between this first prototype of the FM board and the final prototype which is presented in the next section. The reason for modifying the first prototype of the board to implement the final prototype is presented hereinafter.

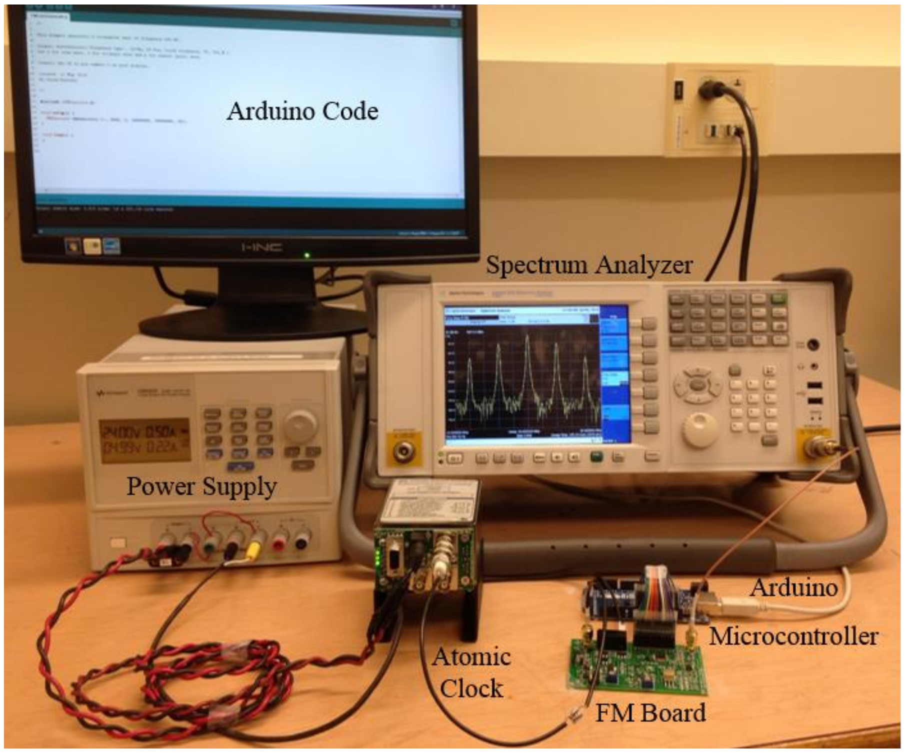


Figure 3-11- Experimental setup to test the first prototype FM signal generator performance.

Two samples of frequency spectra of the generated FM signals by the first prototype FM board are shown in Figure 3-12 and Figure 3-13. As seen in these figures, there is an instability in the distribution of energy in the harmonics of the modulation frequency so that the noise level is about -50 dBm.

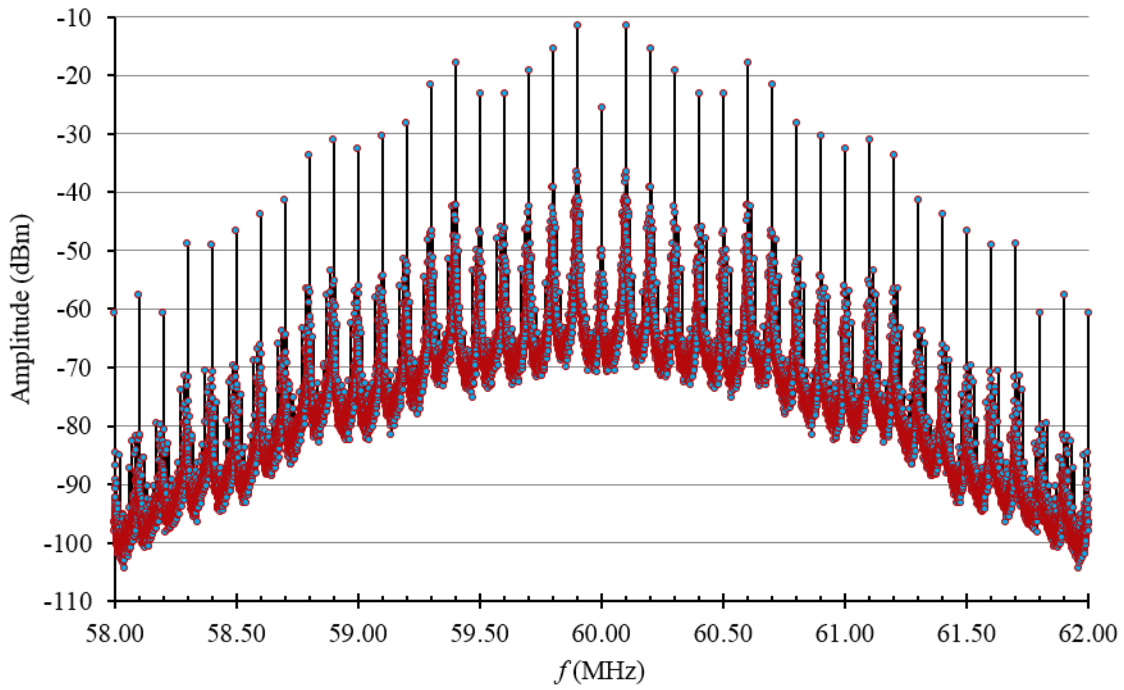


Figure 3-12- Frequency spectrum of FM signal of $f_c = 60$ MHz and $f_m = 100$ kHz generated by the first prototype board.

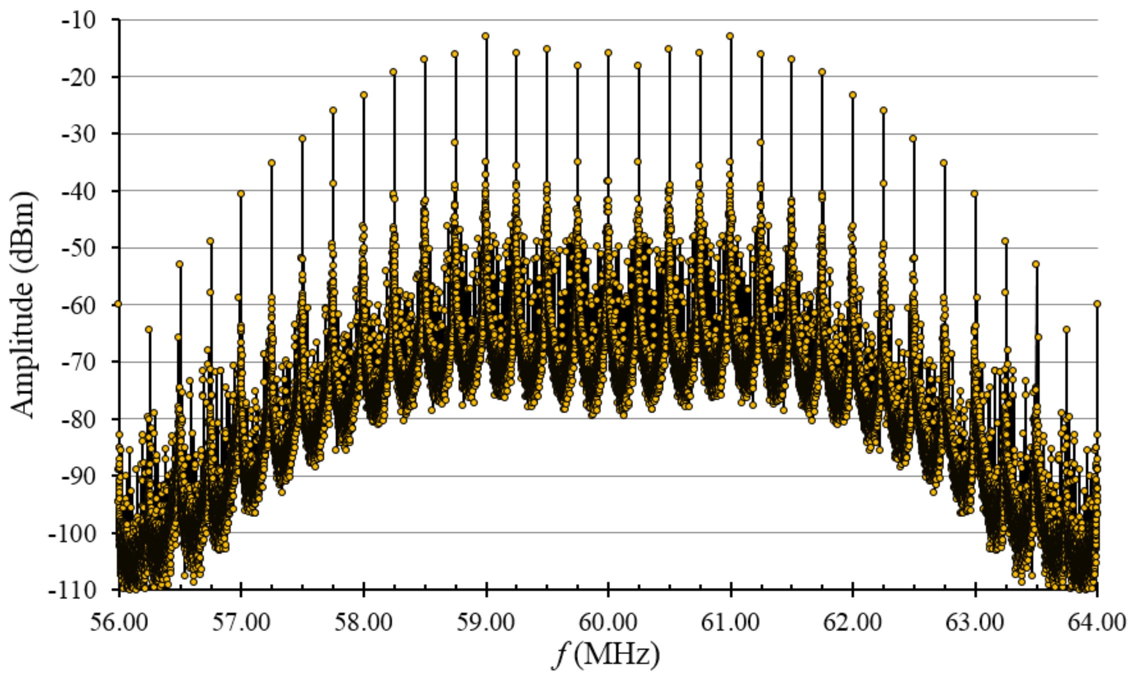
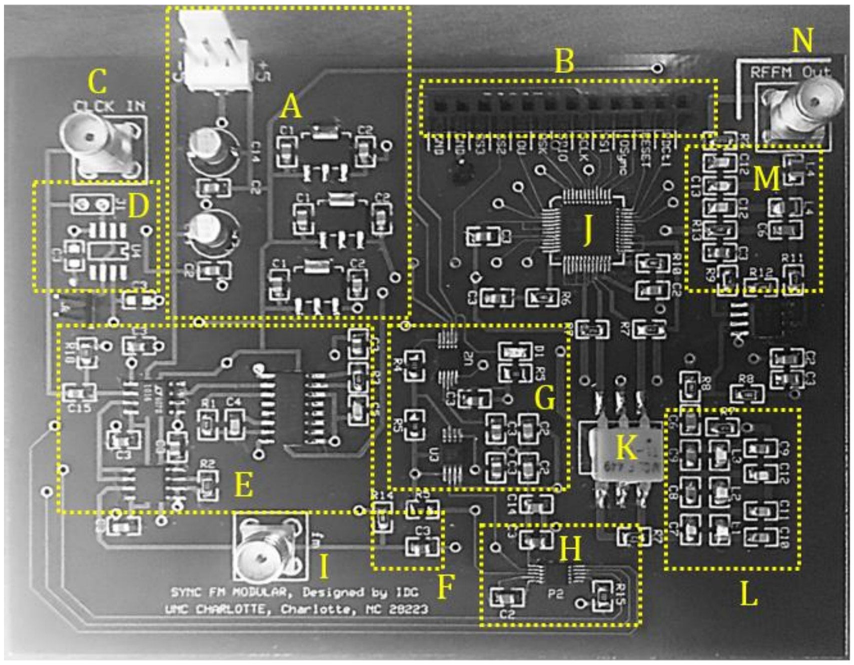


Figure 3-13- Frequency spectrum of FM signal of $f_c = 60$ MHz and $f_m = 250$ kHz generated by the first prototype board.

The tests performed on the board revealed that this instability is due to loading of both the comparator and the DDS1 clock input. To solve this issue, a non-inverting amplifier was used between the atomic clock output and the clock input port of the board. Also, to protect the atomic clock from signals reflecting from the modulator, a buffer was added between the DDS1 clock input and the atomic clock output. These modifications were implemented in the final prototype of the board presented in the next section.

3.4.3 Final prototype FM board

The final prototype of the FM board is shown in Figure 3-14. Since the ground plate of the clock line buffer was the same as the DDS sub-circuits analog ground, it was adding substantial noise to the signal. To solve this issue, the buffer was separated from the PCB as shown in Figure 3-15. The adjustable gain non-inverting amplifier provides control on the amplitude of the clock signal to set it to the required level for the DDS1. Experiments showed that these are the necessary circuits between the atomic clock and the FM board input to achieve the highest signal-to-noise ratio without loading the atomic clock. To test the spectral purity and precision of the frequency components of the final prototyped FM board, the experiment shown in Figure 3-11 was set up for this new board, see also Figure 3-16. A CXA signal analyzer model N9000B (9 kHz – 7.5 GHz) is used in this experiment and the resolution is set to 1 Hz for all the spectra presented in this chapter, except for those that are mentioned specifically.


A
B
C
D
E
F
G
H
I
J
K
L
M
N
CLCK IN
RFFM Out
+5
SYNC FM MODULAR, Designed by IDC
UNC CHARLOTTE, Charlotte, NC 28223

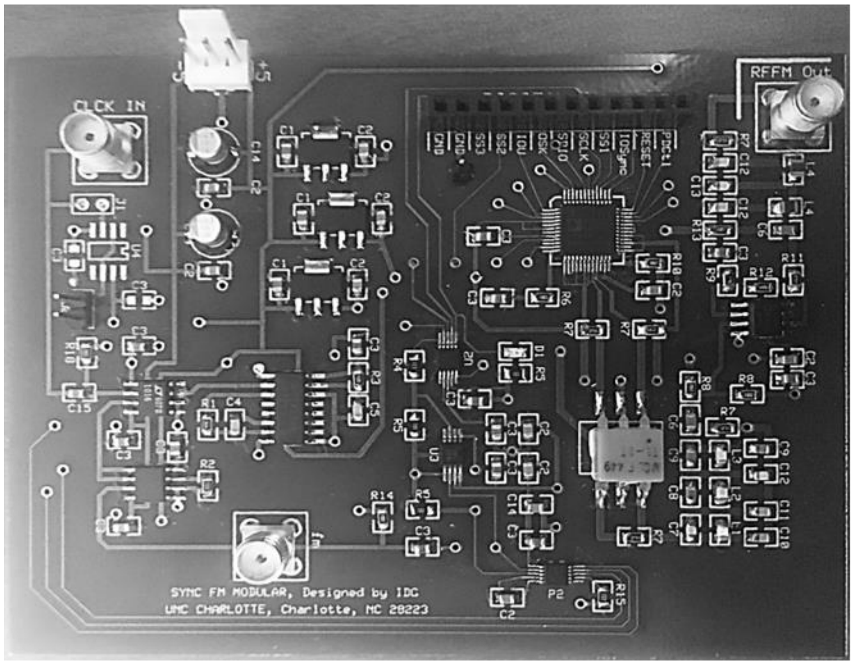

CLCK IN
RFFM Out
+5
SYNC FM MODULAR, Designed by IDC
UNC CHARLOTTE, Charlotte, NC 28223

Figure 3-14- Photograph of the final prototype PCB with different sub-circuits components; A) power input and regulators, B) digital input, C) clock input, D) omitted clock-line buffer layout, E) time-delay, F) high-pass filter, G) DDS1 circuit, H) digital rheostat to control modulation signal amplitude, I) modulation signal output, J) DDS2, K) transformer, L) low-pass filter with 160 MHz cutoff, M) low-pass filter with 75 MHz cutoff, N) FM output.

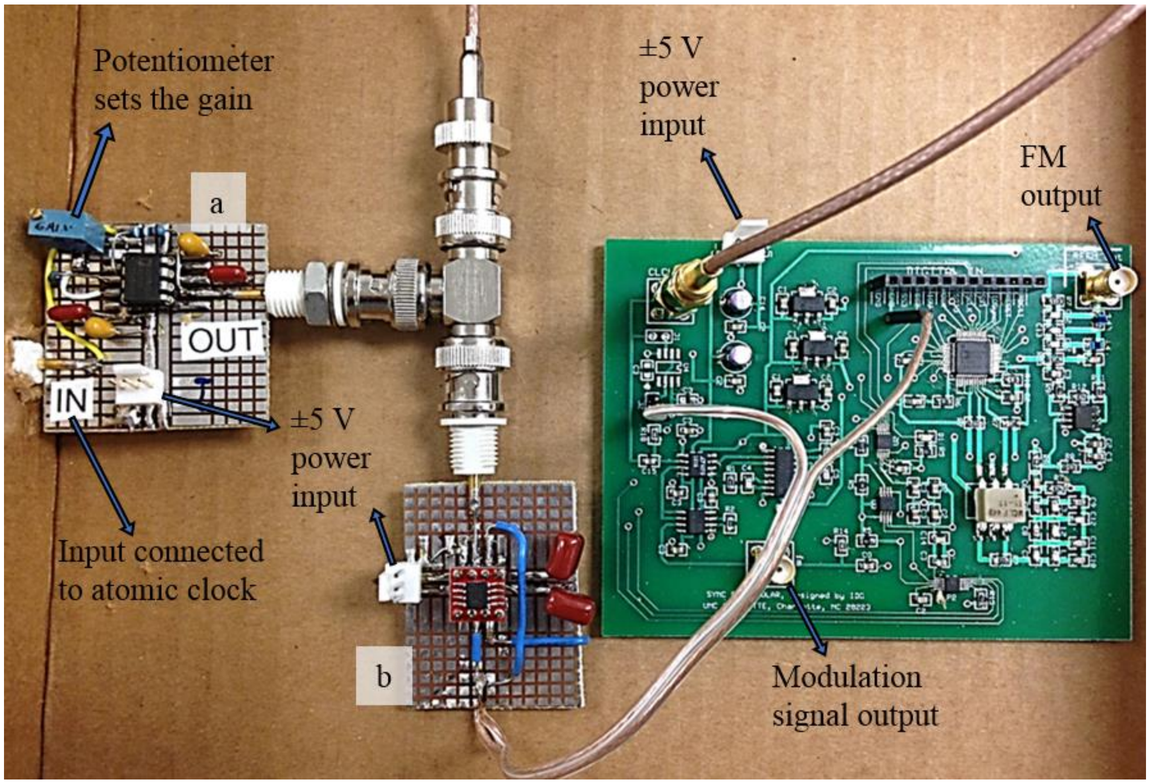


Figure 3-15- The FM board with separated buffers in the clock line; a) non-inverting adjustable gain amplifier; b) buffer between DDS1 and clock line.

There are three parameters in the final prototyped FM signal generator that can be varied, these being the modulation depth β (which is a linear function of V_m), the carrier frequency f_c and the modulation frequency f_m. To study the effects of β on the shape of the spectra, V_m is changed by setting different values of resistance for the digital rheostat in the last version board —that is identical to the manual potentiometer in the first version board— and the spectra captured for a variety of carrier and modulation frequencies.

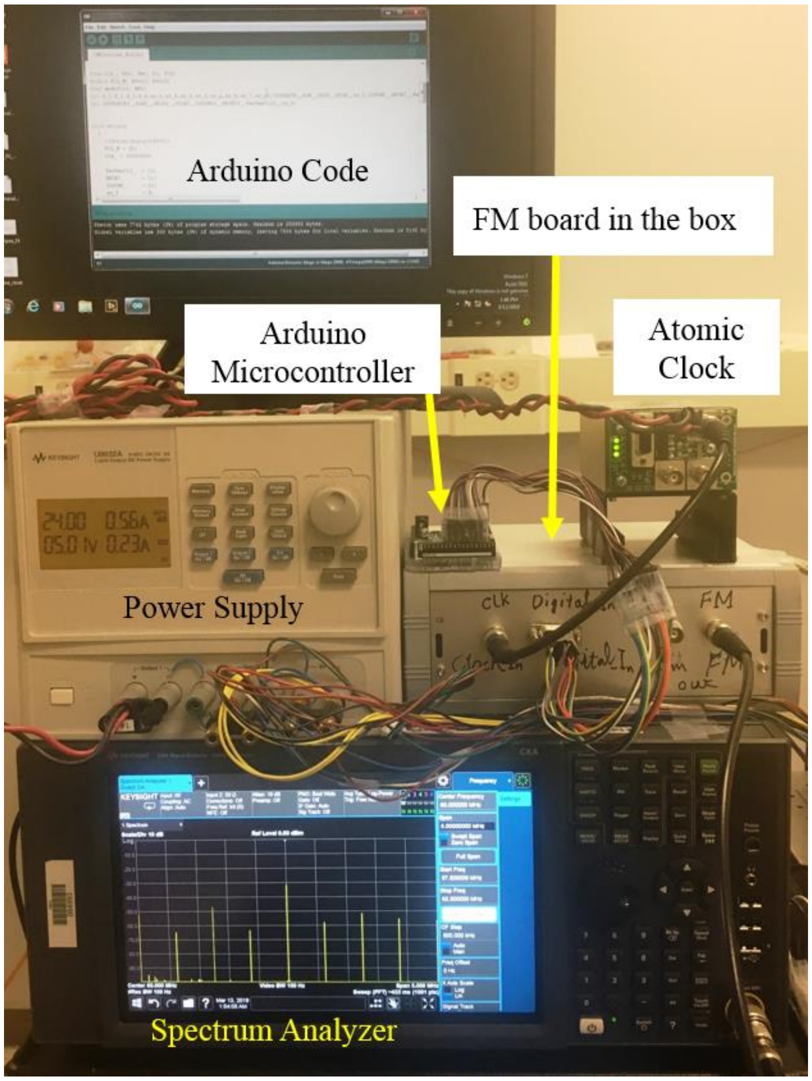


Figure 3-16- Experimental setup to test the final prototype FM signal generator performance.

The frequency spectra of signals with $f_c = 30$ MHz, $f_m = 250$ kHz, and various V_m values are plotted in Figure 3-17 and Figure 3-18. For the data presented in these figures, the resolution of the spectrum analyzer was set to 100 Hz and frequency components of the spectra spanning 27 to 33 MHz with frequency steps of 6 kHz were captured. The reason for setting the resolution higher than 10 Hz, which is the finest possible resolution in the Agilent CSA spectrum analyzer used in this study, is to generate more visible (broader) sidebands for visualization in these 3D figures. The lines plotted on top of the sidebands if Figure 3-17 represent the absolute value of Bessel function of first kind of integer orders -6 to 6 at the same decibel magnitude (dBm) scale used in 3D spectra. To further illustrate the Bessel function dependence of amplitude with modulation depth, the absolute values of Bessel functions of the first kind for integer orders of 0 to 8 are compared with the sidebands magnitudes in Figure 3-18.

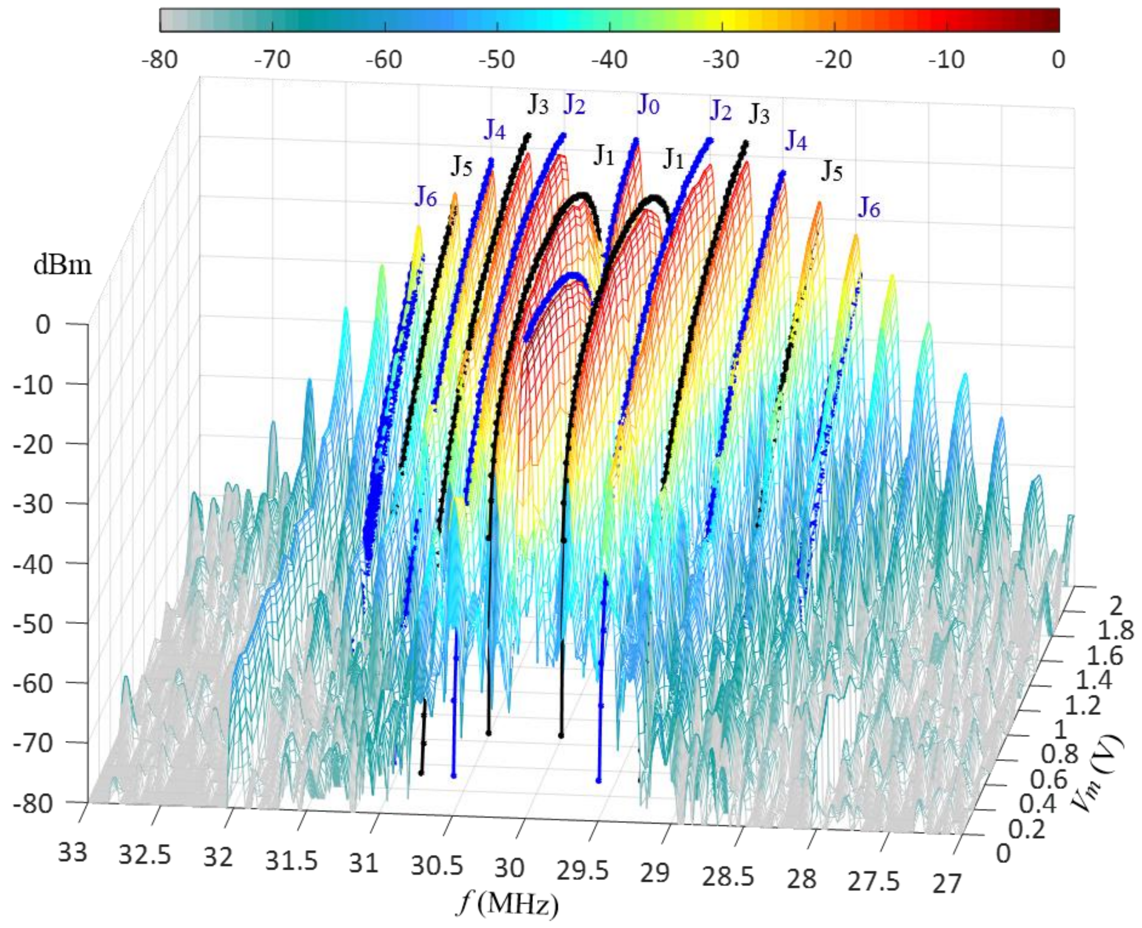


Figure 3-17- Measured frequency spectra of signals of $f_c = 30$ MHz and $f_m = 250$ kHz with varying $\beta(0-3.7)$ indicated by $V_m(0-2)$, compared with modulation amplitude resulted from Jacobi-Anger expansion.

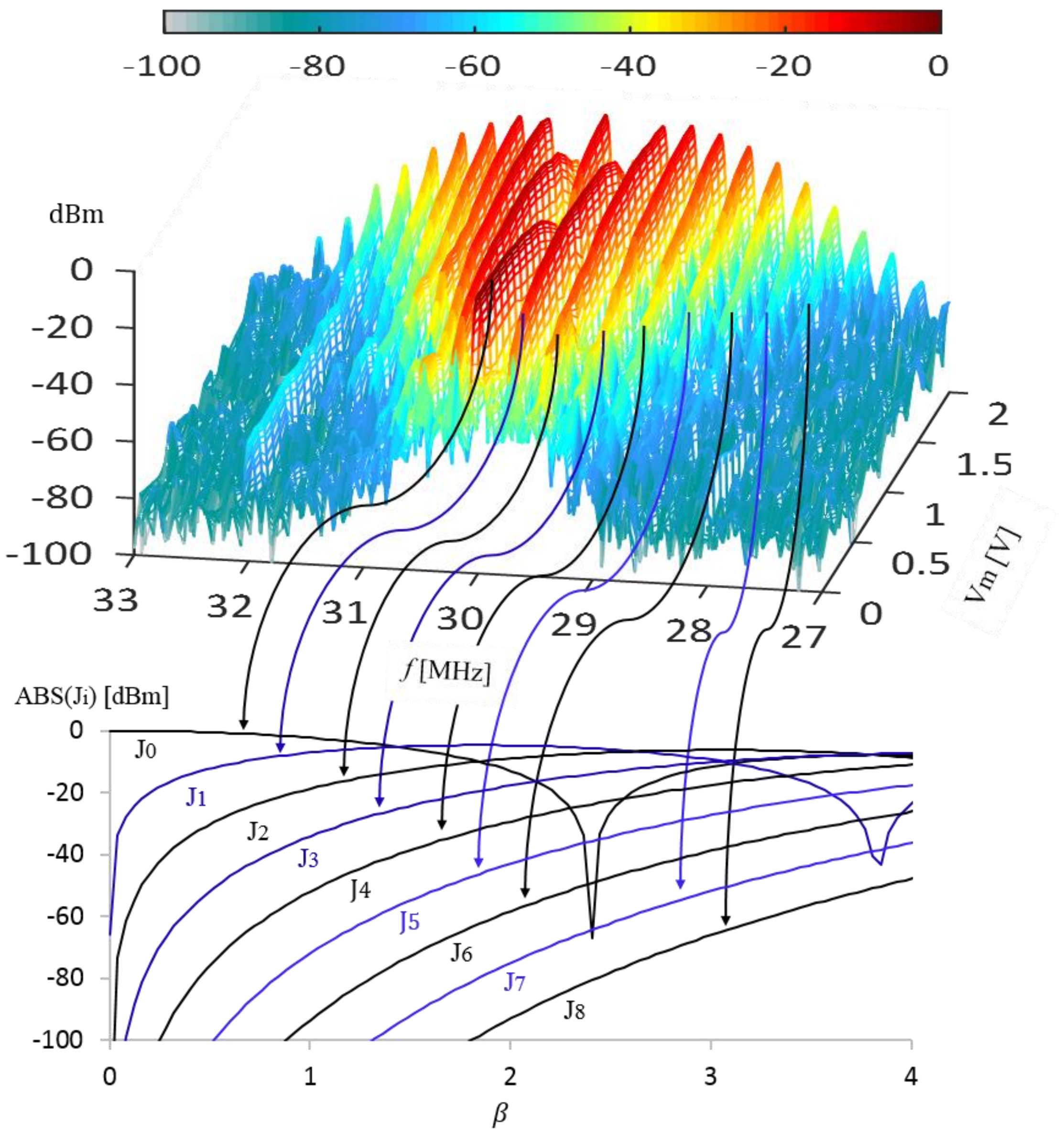


Figure 3-18- Measured frequency spectra of signals with $f_c = 30$ MHz and $f_m = 250$ kHz with varying V_m values of 0 to 2 V compared with the absolute value of Bessel functions of first kind, $J_i(\beta)$, for i = 0, 1, … , 8 in a dBm scale.

As seen in Figure 3-17 and Figure 3-18, the generated signals are consistent with the characteristic predicted by the Jacobi-Anger expansion for the relationship between

β and V_m. The same behavior was observed for signals with other f_c and f_m values ranging from 10 MHz to 70 MHz and 10 kHz to 300 kHz, respectively.

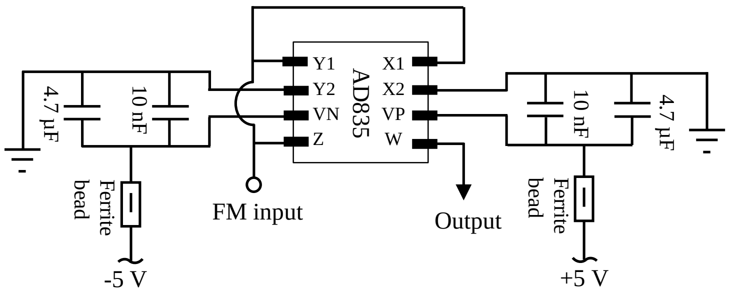


Figure 3-19- Schematic of the mixer, AD835, circuit.

Theoretically, successive modulation frequency harmonics have alternating parity with respect to the carrier of the FM signal, i.e. successive sidebands should have 90° phase difference. To evaluate this phase difference, a mixer (AD835) [70] with two lock-in amplifiers (SRS model SR850) were used to detect the phase difference between the sidebands. Schematic diagrams of the AD835 mixer is shown in Figure 3-19 and the experimental setup to measure the phase difference between different harmonics of f_m is shown in Figure 3-20. Both lock-in amplifiers were externally referenced by locking to the f_m output of the FM board. One of the lock-in amplifiers is locked to the first harmonic at f_m and the other to the second harmonic at $2f_m$. The phase difference between these harmonics was measured for different f_m values. The DSP lock-in amplifiers used here can measure any of the first to the tenth harmonics of the reference signal as long as the frequency is smaller than 100 kHz. Therefore, this measurement was limited to $f_m < 50$ kHz.

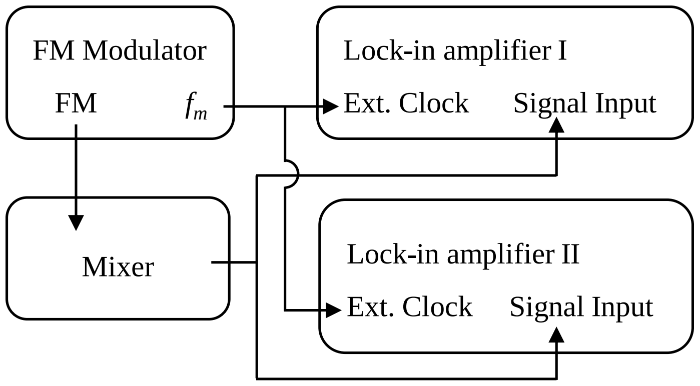


Figure 3-20- Schematic diagram of the experimental setup for measuring the phase difference between odd and even sidebands.

Table 3-1- Measured phase differences between sidebands.

f_m (kHz)	Harmonic I phase (deg.)	Harmonic II phase (deg.)	Phase difference (deg.)
10	-8.31	83.39	91.7
20	0.48	86.88	87.36
30	3.48	91.46	87.98
30	3.09	91.20	88.11
30	2.63	95.94	93.31
35	3.2	92.71	89.51
40	2.87	89.71	86.84
50	-8.41	78.25	86.66

The measured phase differences between the first and second harmonics of different f_m values are presented in Table 3-1. The measurements show phase differences of 90º ± 4º for different frequencies and connecting cables. A practical challenge with this measurement was caused by the reactance of the cables necessary to connect the output of the mixer to the lock-in amplifiers. Because the frequencies of the signals being measured are also different (i.e., one frequency is twice that of the other), it is expected that this also causes small phase shifts of the signals arriving at each lock-in.

Consequently, to verify that these phase variations can be caused by cables, for the 30 kHz modulation frequency, three measurements were made; the second is the same measurement as the first but with the cables to the lock-in amplifiers switched, and the third was the measurement using different cables. For these three measurements at the 30 kHz modulation frequency, it is observed that phase variations of 6º or more can occur. Notwithstanding this cabling issue, the harmonic pairs appear to have the expected phase difference of 90º to within uncertainties due to capacitance of the cables and possible variations of the mixers circuit at different frequencies.

Precision of the generated signal in terms of coherent frequency components with stability governed by that of the atomic clock has been the main motivation of developing the polydyne interferometry technique in this study. Therefore, the spectra of signals with different carrier and modulation frequencies were measured with the analyzer set to its limiting resolution to evaluate the quality, i.e. high signal to noise ratio and spurious free FM signal, of frequency components in the generated FM signals. For the rest of the spectral measurements presented in this chapter, the experimental conditions were: 1) $V_m = 2$ V which translates to different β values for different f_c and f_m values; 2) the resolution bandwidth of the spectrum analyzer was set to the finest value of 10 Hz and the spectra were measured with frequency steps of 1 Hz. To illustrate the coherence of the frequency components, the spectra of signals of $f_c = 10$ MHz, and f_m of 50 kHz, 100 kHz, and 150 kHz are plotted in Figure 3-21. Inset in this figure are two sidebands and the carrier frequency enlarged to illustrate the linewidth of the sidebands and the carrier.

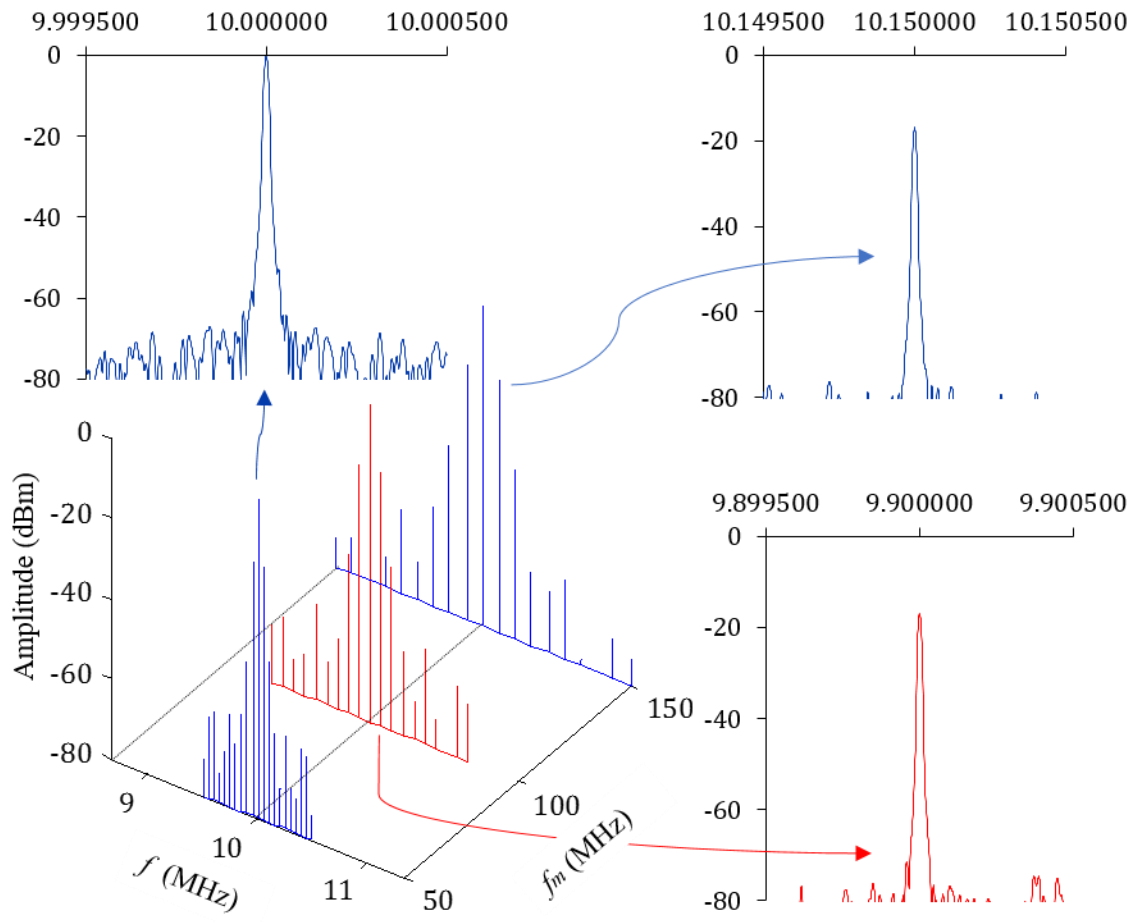


Figure 3-21- Spectra of signals of $f_c = 10$ MHz and $f_m =$ 50, 100, and 150 kHz measured with frequency steps of 1 Hz.

The sidebands shown in Figure 3-21 are supposed to occur at multiple integers of f_m and have been measured to be within 1 or 2 Hz at most which is due to the difference between the timing of the internal clock of the spectrum analyzer and the atomic clock. Although the spectrum analyzer does not use an atomic clock, it has quite acceptable short-term stability that might be inferred from the constant shift in all of the harmonics. However, the captured data are adequate to demonstrate that the harmonics are occurring at exact integer multiples of f_m. Also, the amplitude of the sidebands can be changed by the potentiometer used in the DDS1, as seen in Figure 3-5, to provide the user with the desired modulation depth values. Comparing the frequency spectrum of the Ref. [59] DDS shown in Figure 3-1 with the spectra presented in Figure 3-21 reveals the superiority of our novel FM signal generation method to the digital FM generators. As seen in Figure 3-21, our FM board provides greater than 70 dBc Nyquist band spurious-free dynamic range over its entire operating range. This low level of noise improves the resolution of phase measurements in polydyne interferometer.

Table 3-2- Simulation of AD9951 DDS with PLL multiplier of 20 and AD9833 DDS with clock frequency of 10 MHz

AD9951 DDS		AD9833 DDS	
Target frequency (Hz)	Actual frequency (Hz)	Target frequency (Hz)	Actual frequency (Hz)
10000000.0000	10000000.0090	50000.000000	49999.9895690
20000000.0000	20000000.0190	100000.000000	100000.016391
30000000.0000	29999999.9810	150000.000000	150000.005960
40000000.0000	39999999.9910	200000.000000	199999.995530
50000000.0000	50000000.0000	250000.000000	249999.985099
60000000.0000	60000000.0090	300000.000000	300000.011921
70000000.0000	70000000.0190	312500.000000	312500.000000

Since any DDS uses a finite look-up table, there will be a truncation error associated with the conversion of digital data to analog values at the output DAC of the DDS architecture. This results in a small difference between the target frequencies and the frequency of the signal that is generated. The difference between programmed frequency and that of the generated signal with the AD9951 and AD9833 are provided by Analog Devices. Examples of these values are presented in Table 3-2. Therefore, knowing the truncation error values for any target frequency, systematic errors in the uncertainty budget of precision measurements that are using this FM modulator can be determined. Alternatively, frequencies may be selected to minimize or, in some cases, remove this error. If one of the harmonics of the modulation frequency intersects with the clock frequency, the image of the Digital to Analog Convertor (DAC) used in the AD9833 will not add spurs to the output f_m signal, i.e. the programmed frequency will be exactly generated without any error due to truncation of the lookup table as seen for frequency of 312.5 kHz in Table 3-2. Since this frequency is $\frac{1}{32}\times 10$ MHz, $\frac{1}{32}$ of the atomic clock frequency, it is a perfect match for detection by 32 sample discrete Fourier transform algorithm timed by any harmonic of the same atomic clock signal. This is the scenario used in the discrete Fourier transform algorithm developed in FPGA microchip in CHAPTER 5.

Spectral responses of signals with f_c of 10, 30, 60, and 70 MHz under different f_m values of 50, 100, 150, 200, 250, and 300 kHz have been measured, see APPENDIX B. The results of the captured spectra in the form of relative magnitude and frequency of the first ten harmonies showed that the repeatability and precision of the atomic clock is transferred into all the frequency components of the generated signals. To within the precision and accuracy of the spectrum analyzer, CXA N9000A, the sidebands are located at the programmed frequencies that are the harmonics of modulation frequency either side of the carrier. Another point about these spectra is the low coherence and noise. Apparent in these figures, the level of noise in the spectra is less than -80 dBm. Finally, the spectra of signal of different carrier and modulation frequencies presented in APPENDIX B validate the conclusions presented above based on the spectra shown in Figure 3-21 and lookup table truncation errors shown in Table 3-2

CHAPTER 4 FM LIGHT GENERATION BY AOM

This chapter consists of three sections that present a summary of literature about AOMs in section 4.1, diffraction of light by FM acoustic waves through an AOM in section 4.2, and the experimental results of diffraction of light by FM waves through an AOM in section 4.3.

4.1 Acousto-Optic Modulators

Acousto-optics is a field of study that investigates the interaction of optical waves with acoustic waves in material media. Research began in 1922 with Brillouin predicting the diffraction of light by an acoustic wave that is being propagated through a transmitting medium [71]. A decade later, in 1932 the acousto-optic effect was experimentally verified by Debye and Sears [72], and Lucas and Biquard [73]. Since then, this phenomenon has been extensively investigated in terms of differing conditions: a) the angle of incidence of light into the acoustic wave, b) the wavelength of the ultrasonic wave, c) the wavelength of incident light, d) the amplitude of the ultrasonic wave, and e) the width of the ultrasonic beam [74].

In general, there are two types of diffraction, commonly referred to as normal and abnormal. In 1936, Raman and Nath designed a general model of the interaction between optical and acoustic waves that included multiple order diffraction [75]. Normal diffraction generates multiple order side-beams each of comparable intensity (also called

Raman-Nath diffraction), whereas for abnormal diffraction the intensity of first diffraction orders are dominant (also called Bragg diffraction). In 1937, David followed Brillouin's power series-based analysis for the first and second diffraction orders and gave explicit expressions for the intensities of the two beams on the assumption that the intensities of the higher orders are negligible [76].

In practice, a piezoelectric actuator coupled to an AOM crystal is used to excite it, typically using radio-frequency excitation. To achieve the required stress amplitudes, the piezoelectric actuator and driving circuits are designed to drive the PZT in either its fundamental mode or one of the harmonics of its fundamental mode [77, 78]. This causes the AOMs having an operational frequency band that is dependent on the designed actuator, and the size, geometry, and mechanical properties of the crystal. The frequency of the excitation controls the spatial wavelength of the refractive index variation in the crystal and angle of diffraction in the light wave, whereas the acoustic power controls the optical power distribution in the diffraction orders.

The polarization of the light beam traversing through the AOM medium is affected by the specific crystal used. This is due to different strain-optic coefficients in crystals with different symmetry in their structures [79]. The AOM used in this study (IntraAction model AOM-602AF1) uses dense flint glass for the acousto-optic material [80]. Since glass is an amorphous material, its modulus of elasticity is the same in all directions. This leads to a polarization insensitive AOM, i.e. the diffraction will not affect the polarization of the light.

All the experimental and theoretical models derived so far have considered pure sinusoidal acoustic waves for diffraction of light through AOMs. To the best of our knowledge, there is no experiment or theoretical model in available literature related to diffraction of light by AOMs using FM waves; therefore, the current work is the first study that uses FM acoustic waves to produce FM light by an AOM. A synoptic theoretical model of FM wave propagation through the AOM and its impact on the diffracted light is presented to demonstrate the source of harmonics in the optical signal.

4.2 Diffraction of Light by FM Waves using an AOM

The polydyne interferometer is a single-detector system that uses light from a He-Ne laser diffracted through an AOM that is being driven by an FM signal. A block diagram indicating major components of this system is shown in Figure 4-1. The FM signal generator was discussed in CHAPTER 3. The interaction of the laser beam with the FM signal in the AOM crystal is discussed in this section. Finally, the photodetector and signal processing unit, i.e. the Fourier transform, will be discussed in CHAPTER 5. To study interaction between the light transmitted through an AOM excited with an FM signal, an isotropic homogeneous medium traversed by a plane compression wave propagating in x direction is modeled in Figure 4-2.

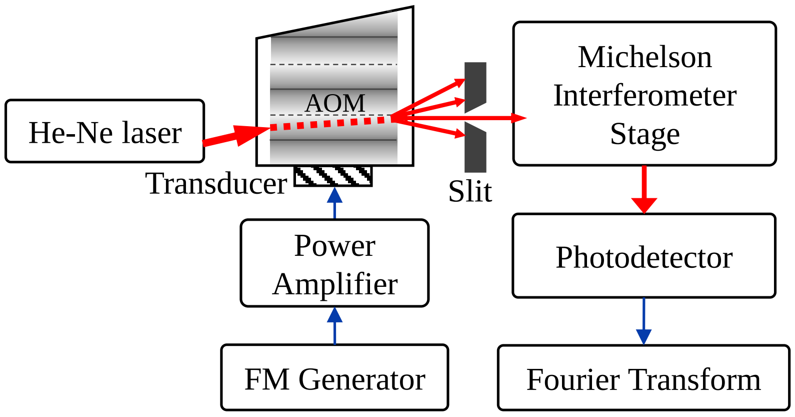


Figure 4-1- Block diagram of the major components of the polydyne interferometer.

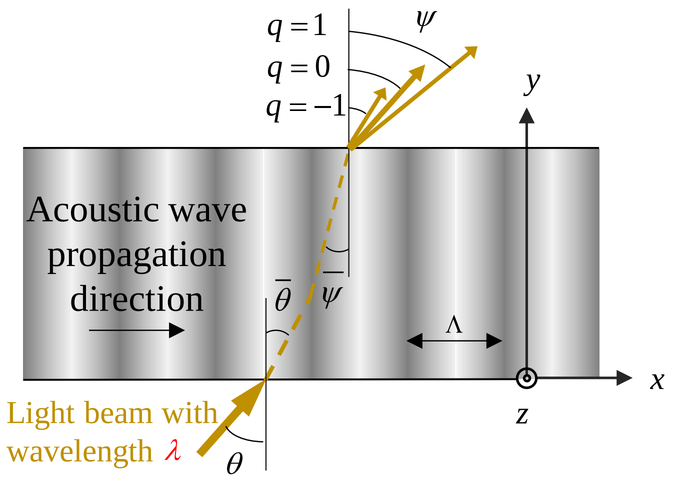


Figure 4-2- Acoustic waves diffracting light beam like a diffraction grating.

As seen in Figure 4-2, the light beam traverses through a particular path due to refraction and diffraction inside the AOM crystal. Refraction bends the light beam at the interfaces of the crystal and air with angles equal to $(\theta-\bar{\theta})$ or $(\psi-\bar{\psi})$; whereas, diffraction adds specific angular shifts to the output beam if the following condition is satisfied [74]:

$$\Lambda\left(\sin\left(\overline{\psi_q}\right)-\sin\left(\bar{\theta}\right)\right)=q\bar{\lambda}, \quad \left(q=0,\pm1,\pm2,...\right). \tag{4-1}$$

where, q represents the diffraction orders, Λ is the acoustic wavelength. Therefore, as it has been explained by Debye and Sears [72] for the angular separation between successive orders equation (4-1) can be approximated by

$$\sin\left(\psi_q\right)-\sin\left(\psi_{q-1}\right)\approx\psi_q-\psi_{q-1}=\lambda/\Lambda\ . \tag{4-2}$$

The photo-elastic effect in material couples the mechanical strains caused by acoustic waves (i.e. pressure waves) to the refractive index of the material [79]. Assuming the refractive index, η, variations of the AOM medium due to FM signal propagation through it are described by:

$$\eta\left(\frac{x}{V}-t\right)=\eta_0+\eta_1\cos\left(\omega_c\left(\frac{x}{V}-t\right)+\beta\sin\left(\omega_m\left(\frac{x}{V}-t\right)\right)\right), \qquad (4\text{-}3)$$

where, V is the velocity of acoustic wave inside the AOM crystal, and η_0 and η_1 are the average and the amplitude of changes of the refractive index of the medium, respectively. Setting $\omega_c\left(\frac{x}{V}-t\right)=\alpha_c$ and $\omega_m\left(\frac{x}{V}-t\right)=\alpha_m$, equation (4-3) can be simplified as:

$$\eta\left(\frac{x}{V}-t\right)=\eta_0+\eta_1\cos\left(\alpha_c+\beta\sin\left(\alpha_m\right)\right). \qquad (4\text{-}4)$$

Using the Jacobi-Anger expansion the cosine term in (4-4) can be expressed by

$$\cos\left(\alpha_c+\beta\sin\left(\alpha_m\right)\right)=\frac{1}{2}\sum_{n=-\infty}^{\infty}\left(e^{i\left(\alpha_c+n\alpha_m\right)}+e^{-i\left(\alpha_c+n\alpha_m\right)}\right)J_n\left(\beta\right), \qquad (4\text{-}5)$$

and

$$\eta\left(\frac{x}{V}-t\right)=\eta_0+\frac{\eta_1}{2}\sum_{n=-\infty}^{\infty}\left(e^{i\left(\alpha_c+n\alpha_m\right)}+e^{-i\left(\alpha_c+n\alpha_m\right)}\right)J_n\left(\beta\right). \qquad (4\text{-}6)$$

Substituting the refractive index variations in Maxwell's equation for a linearly (z axis) polarized monochromatic plane electromagnetic wave gives:

$$\frac{\partial^2 E_z}{\partial x^2}+\frac{\partial^2 E_z}{\partial y^2}-\frac{\partial^2}{c^2\partial t^2}\left[\left(\eta_0+\frac{\eta_1}{2}\sum_{n=-\infty}^{\infty}\left(e^{i\left(\alpha_c+n\alpha_m\right)}+e^{-i\left(\alpha_c+n\alpha_m\right)}\right)J_n\left(\beta\right)\right)E_z\right]=0. \quad (4\text{-}7)$$

Following David's analysis [76] a form of solution can be considered for the z component of the electric field:

$$E_z=\sum_q\sum_n E_{z,i}\left(y\right)e^{i\left(\alpha+q\left(\alpha_c+n\alpha_m\right)\right)}, \qquad (4\text{-}8)$$

where, $E_{z,\mathrm{q}}\left(y\right)$ is the amplitude of the z component of the electric field in the q^{th} diffraction order and will vary as a function of propagation distance y through the AOM crystal. Substituting (4-8) into equation (4-7) yields the series

$$\begin{aligned}&\sum_q\sum_n\left(\eta_0\left(\frac{\omega+q\omega_{cnm}}{c}\right)^2-\left(k\sin(\theta)+qK_{cnm}\right)^2\right)E_{z,q,n}(y)e^{i(\alpha+q(\alpha_c+n\alpha_m))}\\&+\sum_q\sum_n\frac{d^2E_{z,\mathrm{q},n}(y)}{dy^2}e^{i(\alpha+q(\alpha_c+n\alpha_m))}\\&=-\frac{\eta_1}{2c^2}\sum_q\sum_n J_n(\beta)E_{z,\mathrm{q},n}(y)\left(\omega+(q+1)\omega_{cnm}\right)^2e^{i(\alpha+(q+1)(\alpha_c+n\alpha_m))}\\&-\frac{\eta_1}{2c^2}\sum_q\sum_n J_n(\beta)E_{z,\mathrm{q},n}(y)\left(\omega+(q-1)\omega_{cnm}\right)^2e^{i(\alpha+(q-1)(\alpha_c+n\alpha_m))}\end{aligned}\quad. \qquad (4\text{-}9)$$

where, $K_{cnm}=K_c+nK_m=\frac{2\pi V}{\omega_c}+n\frac{2\pi V}{\omega_m}$. Equating the coefficients of each exponential term in equation (4-9), the following recurrence relation can be obtained for $E_{z,\mathrm{q},n}(y)$:

$$\begin{aligned}&\frac{d^2E_{z,\mathrm{q},n}(y)}{dy^2}+\left\{\eta_0\left(\frac{\omega+q\omega_{cnm}}{c}\right)^2-\left(k\sin(\theta)+qK_{cnm}\right)^2\right\}E_{z,\mathrm{q},n}(y)\\&=-\frac{\eta_1J_n(\beta)}{2c^2}\left\{\left(\omega+(q+1)\omega_{cnm}\right)^2E_{z,\mathrm{q}+1,n}(y)+\left(\omega+(q-1)\omega_{cnm}\right)^2E_{z,\mathrm{q}-1,n}(y)\right\}\end{aligned}\quad, \qquad (4\text{-}10)$$

in which, $\omega_{cnm}=\omega_c+n\omega_m$. Equation (4-8) represents a superposition of waves of frequencies $\omega_{\mathrm{q},n}=\omega+q\omega_{cnm}$ for $(q,n=0,\pm1,\pm2,...)$. Moreover, the *x*-component of the wave vector for the wave of frequency $\omega_{\mathrm{q},n}$ is $k\sin(\theta)+qK_{cnm}$. Therefore, the sine of the angle $\psi_{\mathrm{q},n}$ which the wave of frequency $\omega_{\mathrm{q},n}$ makes with the y axis beyond the scattering medium is given by

$$\sin(\psi_{\mathrm{q},n})=\frac{c\left(k\sin(\theta)+qK_{cnm}\right)}{\omega+q\omega_{cnm}}\sim\sin(\theta)+q\frac{\lambda}{\Lambda_{cnm}}\quad\Leftrightarrow\quad\frac{\omega_{cnm}}{\omega}\ll1. \qquad (4\text{-}11)$$

Assuming $\left|E_{z,\mathrm{q}=0}\right|>>\left|E_{z,\mathrm{q}=\pm1}\right|>>\left|E_{z,\mathrm{q}=\pm2}\right|>>...$ and $\begin{cases}E_{z,\mathrm{q}=0}(y=0)=\left|E_z(y=0)\right|=B\\E_{z,\pm q,n}(y=0)=0\quad for\quad q\geq1\end{cases}$ as the proper boundary condition [74], a solution to equation (4-10) is as follows:

For $q=0$:

$$\frac{d^2E_{z,q=0}(y)}{dy^2}+\left\{\eta_0\left(\frac{\omega}{c}\right)^2-k^2\sin^2(\theta)\right\}E_{z,q=0}(y)=0\,. \tag{4-12}$$

A general solution for equation (4-12) is sought in the form

$$E_{z,q=0}(y)=Be^{iky\sqrt{\eta_0-\sin^2(\theta)}}\,. \tag{4-13}$$

For $q=\pm1$:

$$\begin{aligned}&\frac{d^2E_{z,q=\pm1,n}(y)}{dy^2}+\left[\eta_0\left(\frac{\omega\pm\omega_{cnm}}{c}\right)^2-\left(k\sin(\theta)\pm K_{cnm}\right)^2\right]E_{z,q=\pm1,n}(y)\\&=-\frac{\eta_1}{2}J_n(\beta)k^2E_{z,q=0}(y)\end{aligned}\quad, \tag{4-14}$$

the general solution becomes

$$E_{z,q=\pm1,n}(y)=\frac{\eta_1J_n(\beta)\left(Be^{iky\sqrt{\eta_0\left(1\pm\frac{\omega_{cnm}}{\omega}\right)^2-\left(\sin(\theta)\pm\frac{K_{cnm}}{k}\right)^2}}-E_{z,q=0}(y)\right)}{\left[\pm\frac{4\eta_0\omega_{cnm}}{\omega}+2\eta_0\left(\frac{\omega_{cnm}}{\omega}\right)^2\mp\frac{4K_{cnm}}{k}\sin(\theta)-2\left(\frac{K_{cnm}}{k}\right)^2\right]}\,. \tag{4-15}$$

Equations (4-13) and (4-15) are the first step of David's method for an approximate solution from which a further trial and error solution can be developed. These solutions can be corrected by putting equation (4-15) into equation (4-10) and resolving for $E_{z,q=0}$ and also repeating the solution for $E_{z,q=\pm1}$. This procedure can be used iteratively to derive the exact solutions for the intensity and frequency components of higher orders of diffraction by FM waves. A similar procedure is presented by David [76] for diffraction by a single frequency sine wave. Equation (4-15) shows that the diffraction side-beams, $q\geq1$, contain all the frequency components of the FM signal.

Therefore, each of these diffracted beams can be extracted using a slit and utilized as an independent FM light source in interferometry.

4.3 Experimental Results of Diffraction

An experiment was implemented to test the effects of angle of incidence of light into the acoustic wave, θ, and modulation frequency and depth of the FM acoustic wave on the diffraction of the light through an AOM. The experimental apparatus for these test comprised, an IntraAction power amplifier model PA-4, an IntraAction AOM model 602AF1, a single pivot angle-adjustment notch-hinge flexure stage, and a He-Ne laser, see Figure 4-3.

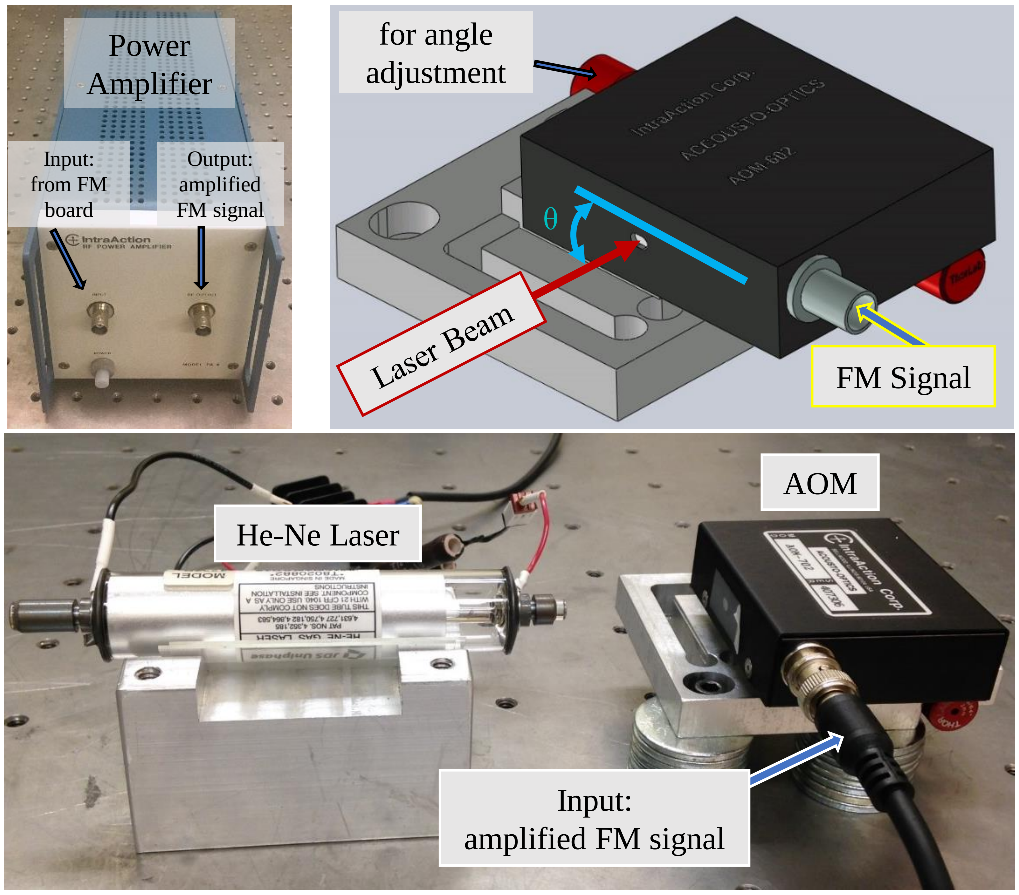


Figure 4-3- Experimental setup to test the light diffraction by FM waves via an AOM.

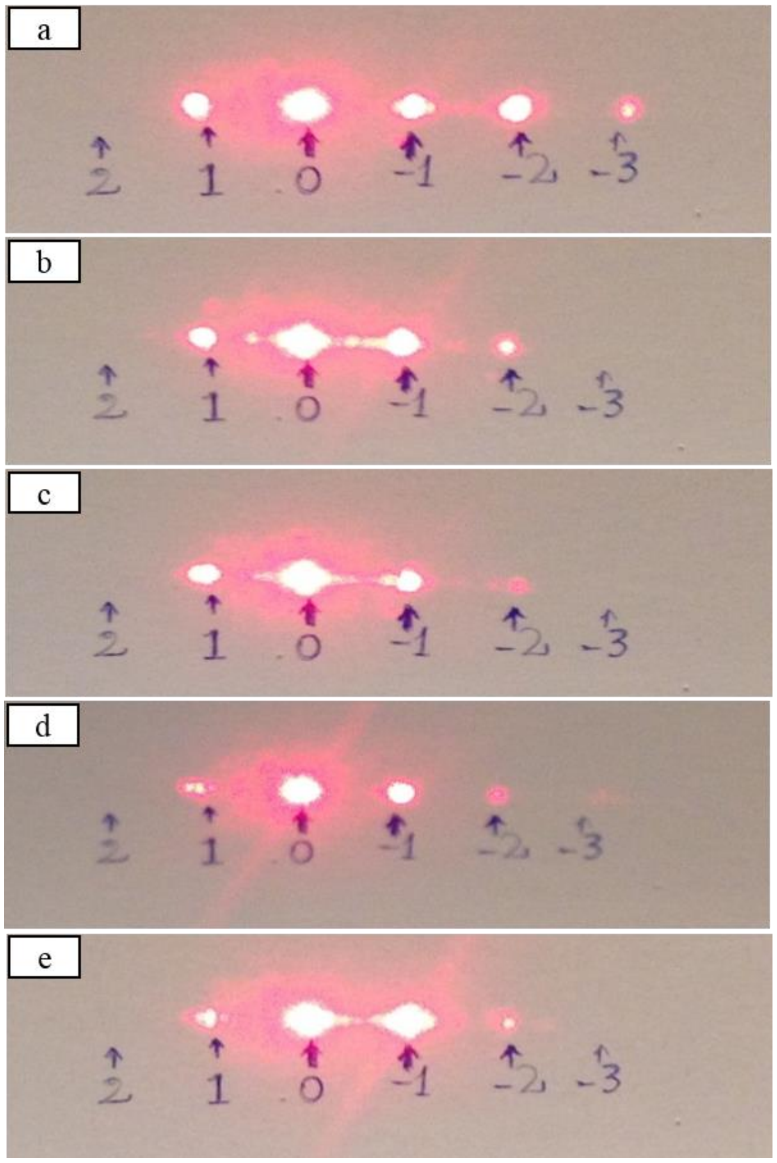


Figure 4-4- Different diffraction patterns resulted from different incident angles.

Distribution of energy among the diffraction side-beams is dependent on the angle of incidence of the light entering into the AOM crystal. To demonstrate the effects of incident angle on the diffraction pattern, the diffracted light beam is directed on a screen 2 meters away from the AOM output. The intensity distribution in the side-beams is determined while changing the angle of incidence of light into the AOM crystal. The diffraction patterns for an FM signal of $\omega_m = 10$ kHz and $\omega_a = 60$ MHz are shown in

Figure 4-4. All the different regimes of diffraction shown in Figure 4-4 occur within 1° of incidence angle change.

Figure 4-4-a belongs to Raman-Nath diffraction which contains multiple orders of side-beams with comparable intensities. Figure 4-4-b to Figure 4-4-e show the Bragg diffraction where the angle into the AOM is varied over a relatively large range of approximately ±1 degree. Existence of weak higher order side-beams in Bragg diffraction as well as finite angular range for generating constructive interference that produces diffraction has been previously demonstrated by other researchers in Refs. [72, 73]. Since diffraction side-beam number -1 shown in Figure 4-4-e has the highest intensity among all the diffraction side-beams, it was extracted using a slit for use during interferometric measurements. The extracted diffraction side-beam is shown in Figure 4-5.

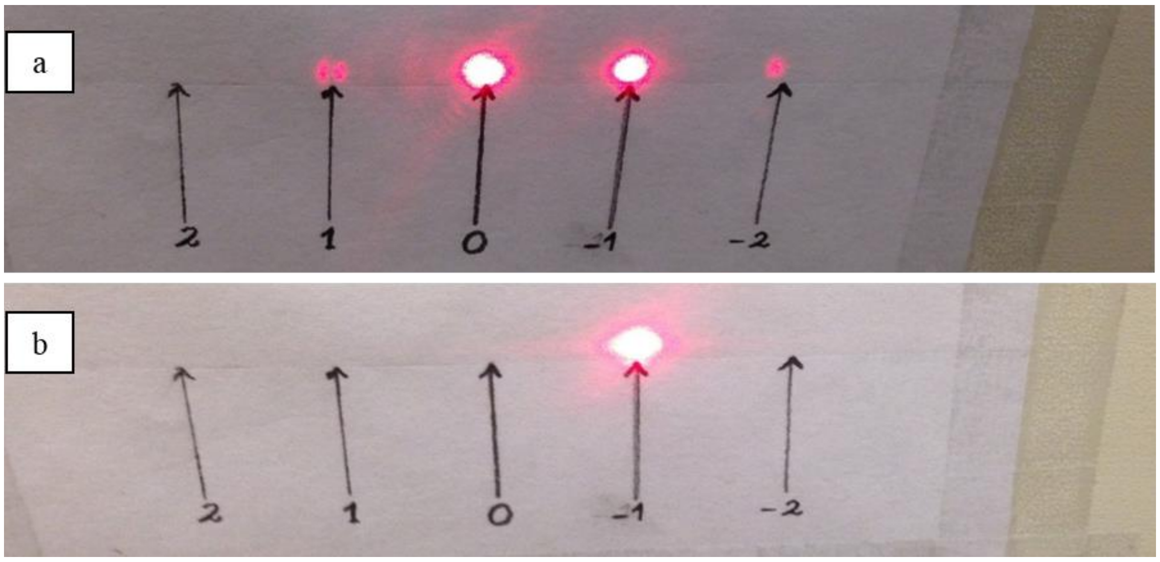


Figure 4-5- Diffraction side-beam extraction by a slit; a) before using slit; b) after passing through the slit.

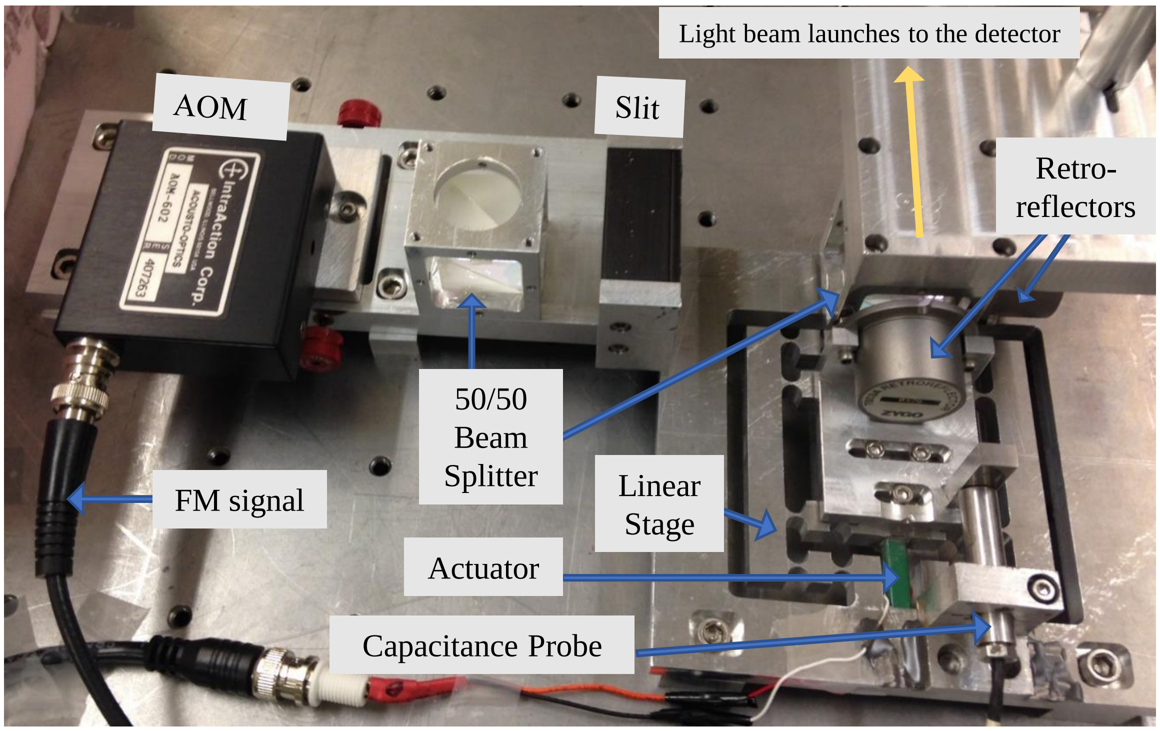


Figure 4-6- Experimental setup to evaluate the frequency spectra of diffraction side-beams.

To evaluate the frequency content of each of the diffraction side-beams, they were extracted by a slit and used in a Michelson interferometry stage, see Figure 4-6. In this experimental set-up, a He-Ne laser is diffracted through the AOM that is being driven by an electrical FM signal. The non-polarizing 50/50 beam splitter divides the diffracted light into two parts. One part is used in Michelson stage and the other part can be later used for different goals like stabilizing the laser or assuring that the diffraction side-beams are stable throughout the measurements. One of the diffraction side-beams is extracted by a slit from all the diffraction side-beams. This side-beam is also divided into two parts by another non-polarizing 50/50 beam splitter to construct the reference and moving arm of the Michelson interferometer topology. These beams traverse through different arms of the interferometer and reflect back to interfere with each other on the point they reach the splitting surface of the beam splitter. The interferometry stage is explained in more detail in CHAPTER 6. The interfered light launch into a photodetector and the photodetector output was monitored in an oscilloscope and a spectrum analyzer. The retroreflector of the moving arm is implemented on a linear stage that is actuated by a piezo-electric actuator. The photodetector used in this experiment is explained as the first prototype photodetector in CHAPTER 5.

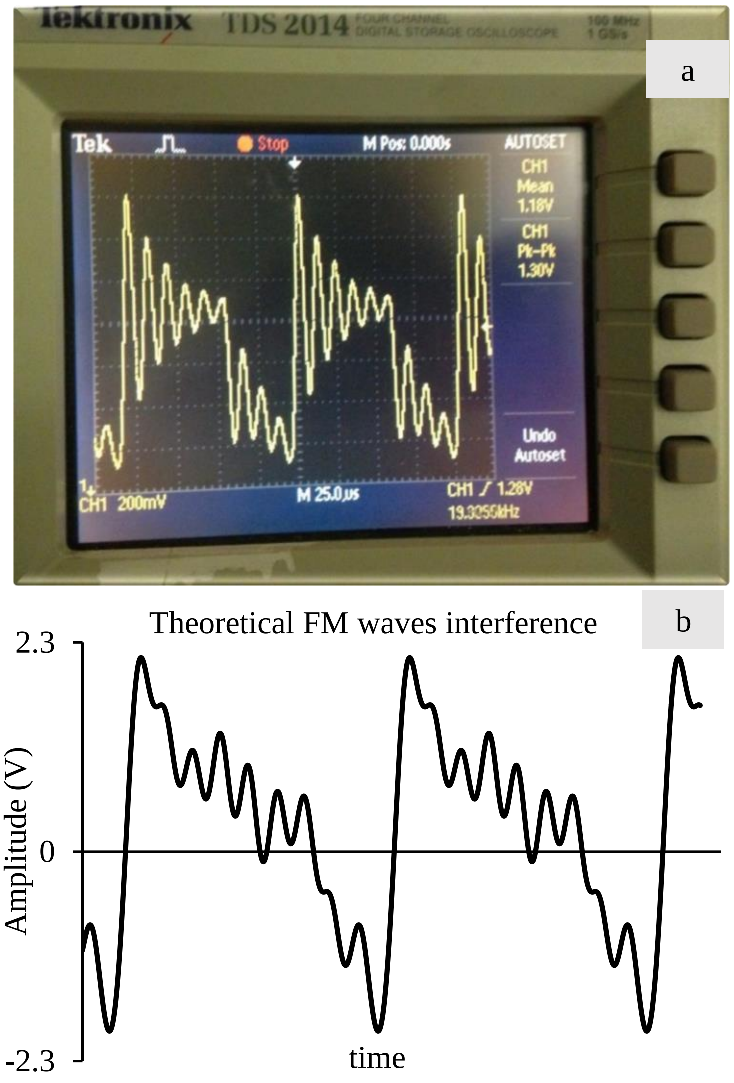


Figure 4-7- FM light interference resulted from a) experimental setup shown in Figure 4-6 monitored in an oscilloscope, and b) theoretical FM waves interference.

The output of the photodetector is monitored in an oscilloscope, see Figure 4-7-a. This signal is similar to the theoretical FM waves interference signal shown in Figure 4-7-b. The theoretical FM wave interference is simulated by accounting first 10 orders of the modulation harmonics with different amplitudes as

$$V(t) = \sum_{k=1}^{10} V_k \sin\left(2\pi k f_m t + \varphi_k\right) \qquad \text{(4-16)}$$

where, V_k and φ_k are the k^{th} harmonics amplitude in volts and phase in radians that are considered to be constant with values $\left\{\begin{matrix} V_1 = 1, V_2 = 0.7, V_3 = 0.48, V_4 = 0.32, V_5 = 0.28, \\ V_6 = 0.2, V_7 = 0.15, V_8 = 0.05, V_9 = 0.2, V_{10} = -0.2, \\ \varphi_1, \varphi_2 = -0.5, \quad \varphi_3, ..., \varphi_{10} = -1.5 \end{matrix}\right\}$.

To demonstrate that the interference of each of the diffraction side-beams in the Michelson stage results in stable harmonics of the modulation frequency of the acoustic FM waves, the frequency spectra of the photodetector signal was studied for different modulation frequencies. Examples of these spectra are shown in Figure 4-8 for f_m of 5 kHz and 25 kHz. As seen in Figure 4-8, the interfered signal contains multiple harmonics of the modulation frequency as it was expected from the theoretical analysis of frequency content of FM signals interference. Results shown in Figure 4-7 and Figure 4-8 reveal that each of the diffraction side-beams contains the frequency components of the electric FM signal that is driving the AOM. Thus, the main goal of patching the electric FM signal into a laser beam has been achieved.

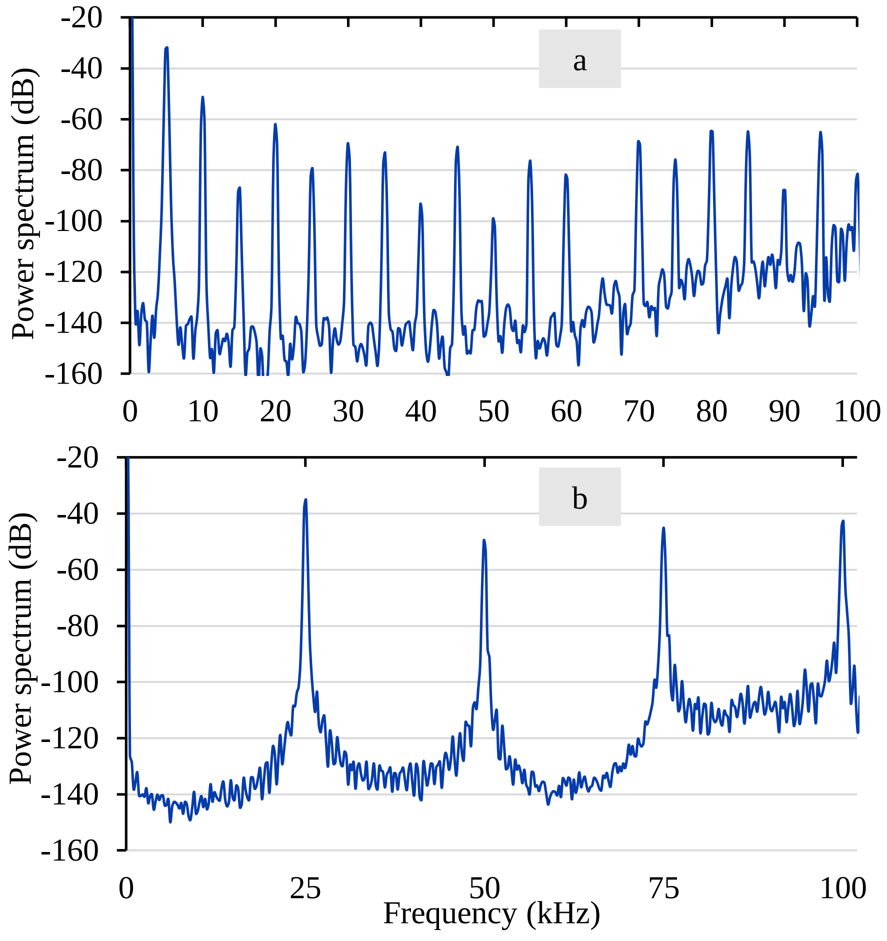


Figure 4-8- Frequency spectra of the photodetector signal for a) $f_m = 5$ kHz, and b) $f_m = 25$ kHz.

CHAPTER 5 SIGNAL DETECTION

This chapter consists of three sections. Section 5.1 presents the design of two photodetector circuits each of which with different detection bandwidth and noise characteristics. Section 5.1 presents the extraction of modulation harmonics from the interfered FM light by analog lock-in and demodulation techniques. Section 5.2 presents the application of FPGA-based synchronous discrete Fourier transform algorithm for extraction of modulation harmonics in the interfered FM light beams.

5.1 Photodetector Design

Photodiodes are high impedance sensors used to detect the intensity of light that operate either in photovoltaic mode (with zero bias) or in photoconductive mode (with a reverse bias). These two operation modes are presented in Figure 5-1. Photovoltaic mode provides the most precise linear operation, while the photoconductive mode results in higher switching speeds at the expense of linearity. Under reverse bias conditions, a small amount of dark current flows even when there is no illumination. The dark current leakage can be compensated by a second photodiode of the same type in the noninverting input of the op-amp, see Figure 5-2.

Three factors influence the response time of a photodiode: 1) the charge collection time of the carriers in the depleted region, 2) the charge collection time of the carriers in the un-depleted region, and 3) the RC time constant of the diode circuit combination.

Because photodiode junction capacitance is dependent on its diffused area and the applied reverse bias, faster rise times are obtained with smaller diffused area photodiodes and larger applied reverse biases.

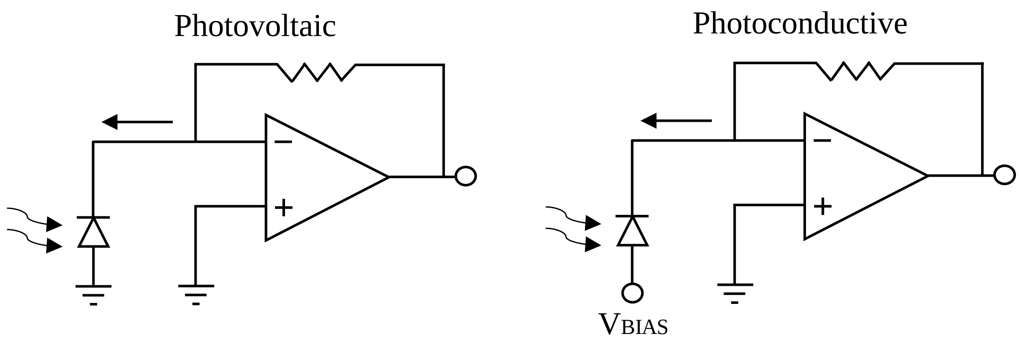


Figure 5-1- Photovoltaic and photoconductive modes.

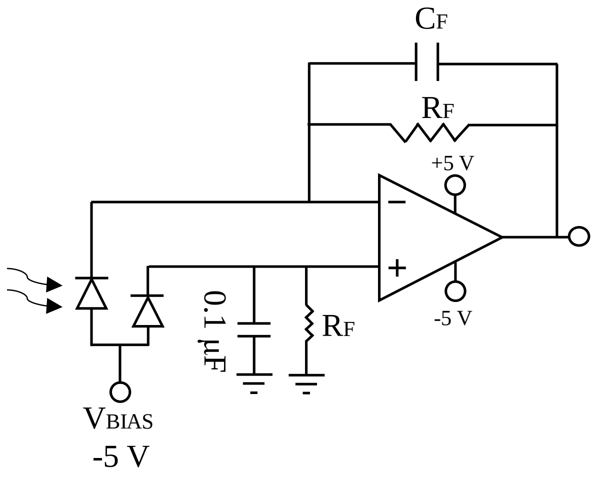


Figure 5-2- Dark current compensation in photoconductive mode [81].

There are several important amplifier characteristics that should be considered for achieving a desirable high signal to noise ratio when the incident light contains substantial noise. Such characteristics are: gain bandwidth, input offset voltage, input noise voltage, input bias current, input offset current, and input noise current [82]. There are two general amplifier circuits for photodetectors: pre-amplifier and transimpedance amplifiers [82], see Figure 5-3. The current to voltage conversion mechanisms in these two amplifier configurations are different. The pre-amplifier circuit converts the current to voltage through the resistor to ground, R_S, and since the op-amp is amplifying the voltage it is called a voltage amplifier. In the transimpedance circuit, the amplifier itself acts as a current-to-voltage convertor and this is why the circuit is often called current amplifier. Bipolar Junction Transistors (BJT) op-amps provide a better performance for pre-amplifier configuration because this configuration is less sensitive to current noise and highly sensitive to voltage noise. Contrary to the pre-amplifier circuit, the transimpedance configuration is sensitive to current noise. Therefore, Field Effect Transistor (FET) op-amps are a better choice for the transimpedance setup. In general a photoconductive transimpedance circuit provides the highest detection speed among all of the four possible configurations, i.e. photoconductive pre-amplifier, photoconductive transimpedance, photovoltaic pre-amplifier, and photovoltaic transimpedance configurations.

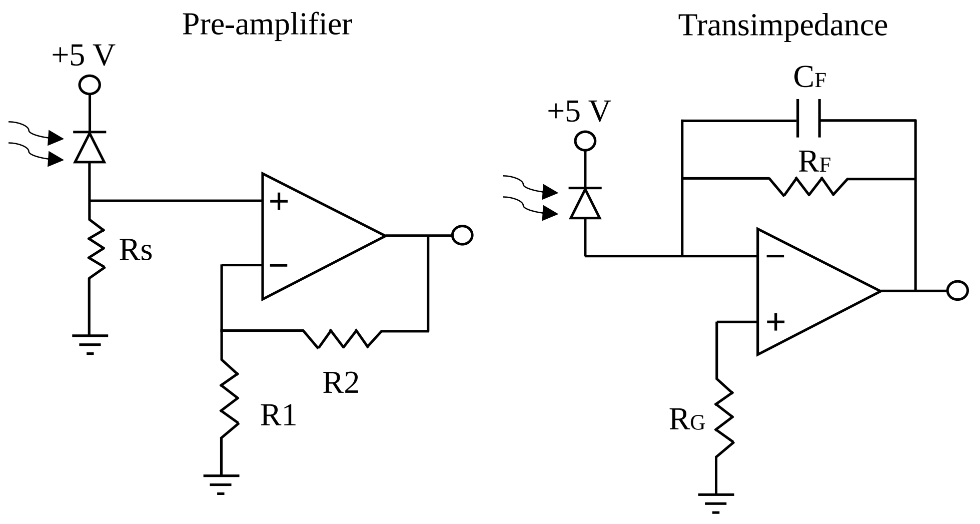


Figure 5-3- Pre-amplifier and transimpedance configuration of photodetector circuits.

Different harmonic detection methods, i.e. lock-in amplifiers and FPGA-based discrete Fourier transform, require different bandwidths. This leads to different photodetector design requirements. At the start of this project, two Stanford lock-in amplifiers model SR850 [83] were used for harmonics extraction from the detected signal; the photodetector designed for this step of the project is discussed in section 5.1.1. As discussed in detail in section 5.2, the lock-in amplifiers limit the speed of the displacement measurement because of their detection bandwidth limits. To overcome this bandwidth limitation a synchronous Fourier transform algorithm in an FPGA microchip was developed to boost the speed of measurement substantially. The FPGA-based harmonics extraction needed a photodetector with specific characteristics that are discussed in section 5.1.2.

5.1.1 First photodetector

The first photodetector was implemented by modifying different photovoltaic and photoconductive transimpedance configurations discussed in Refs. [84, 85], see Figure 5-4. All the prototypes that are shown in Figure 5-5 are based on the circuits presented in Figure 5-4 with different photodiodes and op-amps. Among all the photodetectors prototyped based on the circuits in Figure 5-4, the photovoltaic transimpedance photodetector with BPX61 photodiode [86] provided the best signal to noise ratio. The photodiode of this circuit is followed by a high-pass filter with cut-off frequency of 100 Hz to decrease the DC-level fluctuations that resulted from homodyne part of the light intensity from the Michelson interferometer experiments. Reducing the DC-level fluctuations enables the subsequent amplification stage to dedicate a greater portion of its gain to higher frequency components, including the harmonics of modulation frequency, which consequently increases the sensitivity of the photodetector circuit to fluctuations at harmonics of modulation frequency.

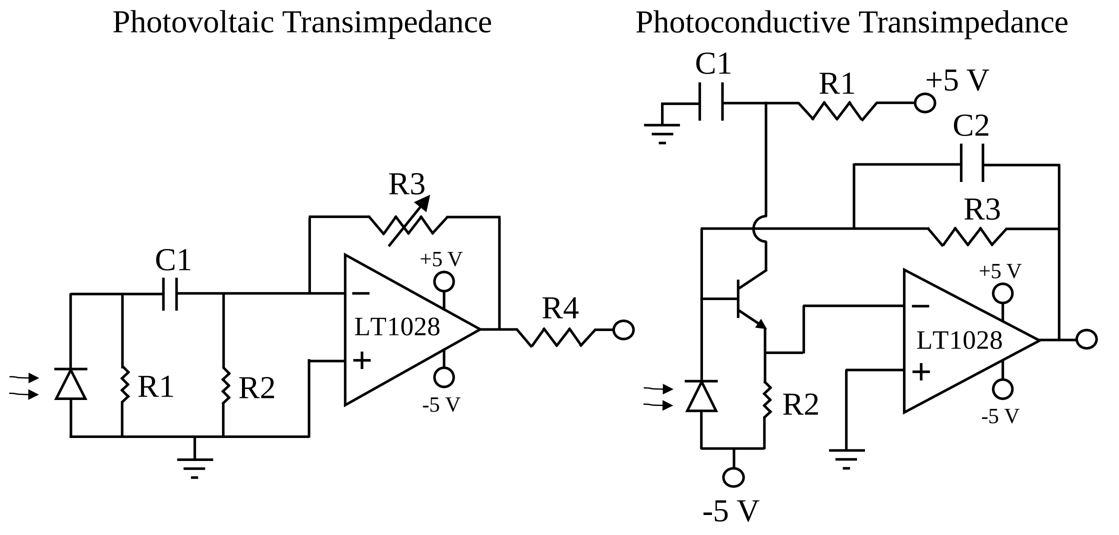


Figure 5-4- The photovoltaic and photoconductive transimpedance photodetector circuits.

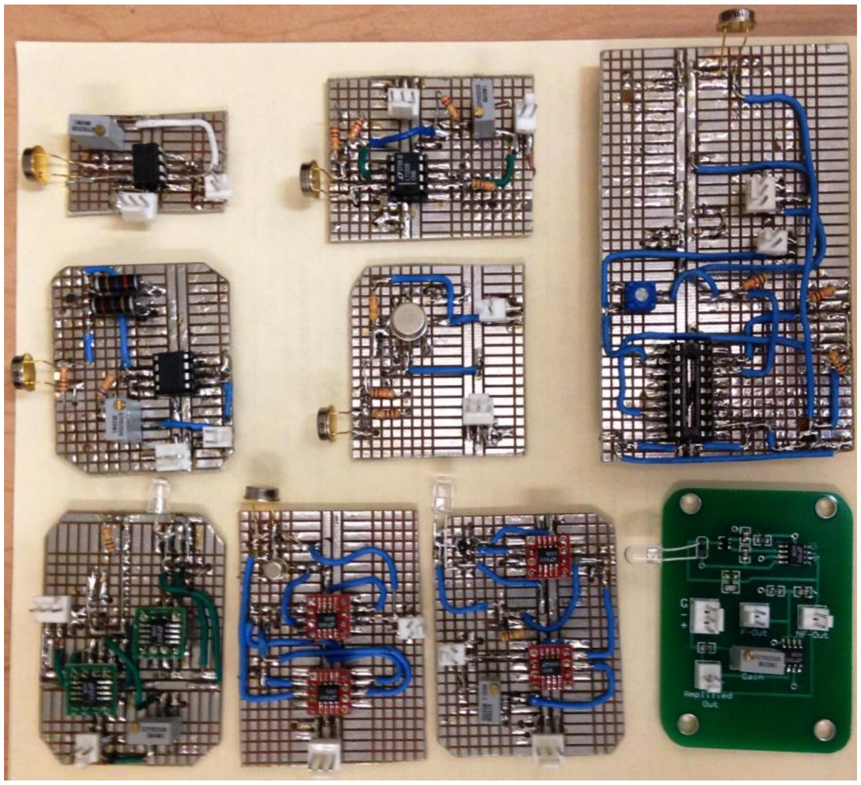


Figure 5-5- Different prototypes of the first photodetector.

5.1.2 Second photodetector

To achieve the desired displacement measurement speed of 10 mm·s^{-1} with 50 nm resolution, the phase signal should be detectable at a bandwidth greater than 200 kHz which requires the modulation frequencies of about 200 kHz. Furthermore, to extract the phase information from 10 harmonics of the modulation frequency, this requires 200 kHz detection bandwidth centered around 2 MHz. Therefore, the photodetector should be designed to provide a detection bandwidth of up to 2.1 MHz. This goal can be achieved by a preamp photodetector circuit using a photodiode with small junction capacitance followed by an op-amp with picoamps of input bias current and low input offset voltage to minimize the error. The output of the photodetector circuit should be fed into an Analog-to-Digital Convertor (ADC) with conversion speed of greater than 20 MHz to provide at least 10 samples per period for 2 MHz signal.

Looking for characteristics of different photodiodes, OSRAM Opto Semiconductors SFH 2701 PIN photodiode seems very promising as it has a small junction capacitance of 3 pF typical, 5 pF maximum for 0 V bias. The typical capacitance is 1.7 pF for 5 V reverse bias. Measurements in the circuit were all taken with a reverse bias of 5 V. The AD8065 op-amp with a very small input bias current and offset voltage of 2 pA and 400 µV, respectively, satisfies the required bandwidth and input characteristics. The AD9629-20 is a good candidate for a 20 MHz sampling rate ADC with 12 bits of resolution. This photodetector circuit followed by the ADC is shown in Figure 5-6. Detailed analysis of the noise and resolution matching for this photodetector is presented in [81]. The circuit presented in this study is a modified version of the circuit presented in [81] to cancel the DC part of the signal for preventing the ADC from overflowing by the unnecessary DC part of the detected signal. A photo of the modified photodetector is shown in Figure 5-7.

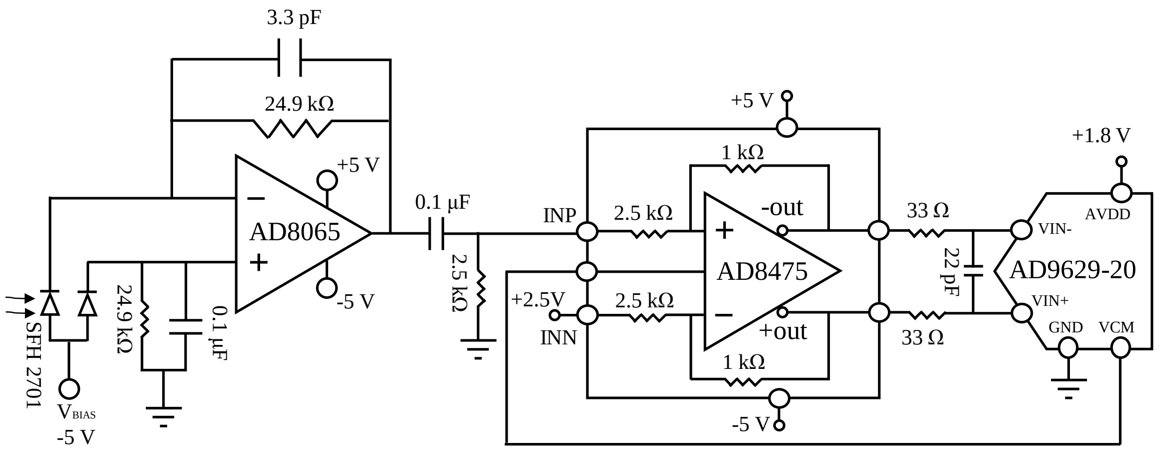


Figure 5-6- Photodiode preamp system with dark current compensation [81].

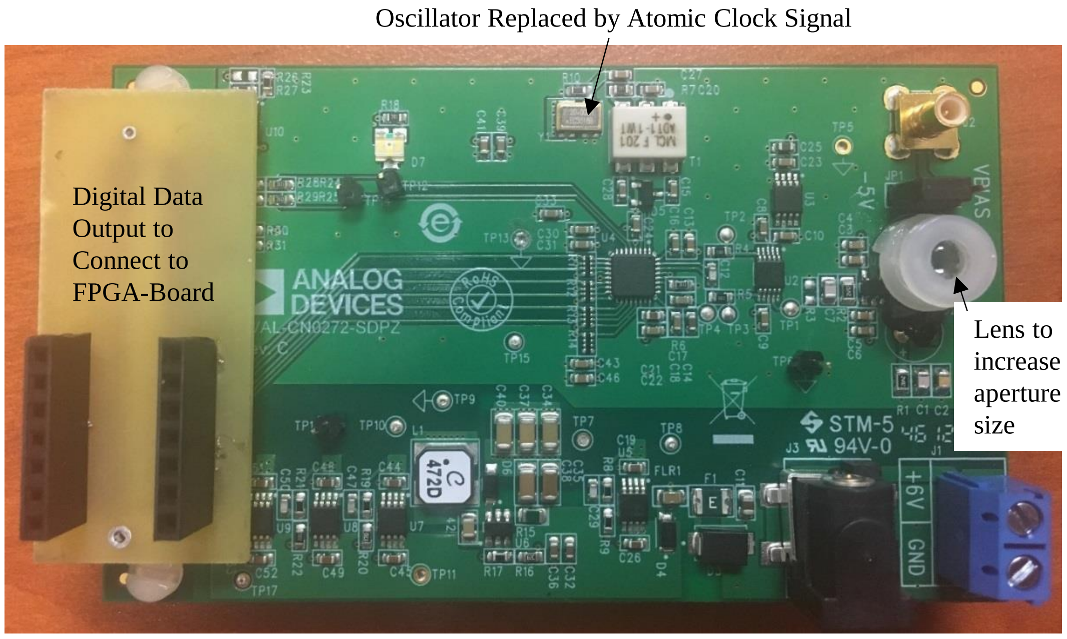

Oscillator Replaced by Atomic Clock Signal
Digital Data Output to Connect to FPGA-Board
Lens to increase aperture size

Figure 5-7- Modified photodetector for use with FPGA board.

5.2 Harmonics detection by analog lock-in amplifiers

As mentioned in CHAPTER 2, the phase information is extracted from the amplitudes of the harmonics of modulation frequency in the interfered FM light intensity measured by the detector. To this end, the detected signal is transferred to the frequency domain using Fourier transform. Lock-in amplifiers use analog demodulators to mix the harmonics of any particular frequency that is connected to their external clock to extract the Fourier coefficients of those harmonics in a signal. In fact, these analog demodulators apply fast Fourier transform to the signal. A schematic of this demodulation technique is represented in Figure 5-8. The low-pass filter, in Figure 5-8, after the mixer has a cut-off frequency of one-tenth of the reference input, i.e. desired frequency to be extracted. This limits the bandwidth of analog lock-in / demodulation circuits to one-tenth of the frequency to be extracted.

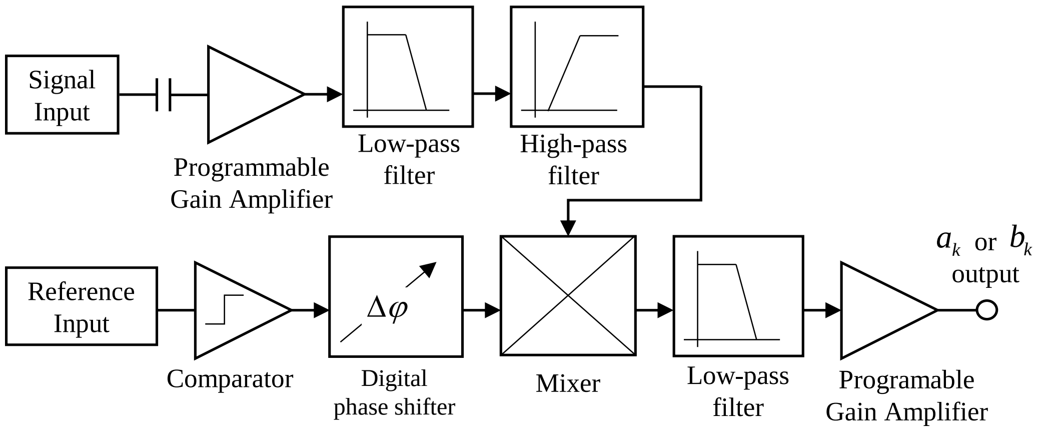


Figure 5-8- Schematic of Fourier transform implementation through analog demodulators.

At the onset of this research, two DSP lock-in amplifiers model SR850 [83] were used for extracting the modulation frequency harmonics from the photodetector output, see Figure 5-9. These lock-in amplifiers have a detection bandwidth of 1 mHz to 102 kHz for detection of any first 10 harmonics of the reference input. In an effort to increase demodulation bandwidth, AD630 demodulators [87] were used to create a compact lock-in amplifier, see Figure 5-10. This lock-in circuit provides 350 kHz bandwidth for extraction of amplitude of the reference signal frequency component present in the demodulation signal input. The results of measurements with these lock-in amplifiers are discussed in CHAPTER 6.

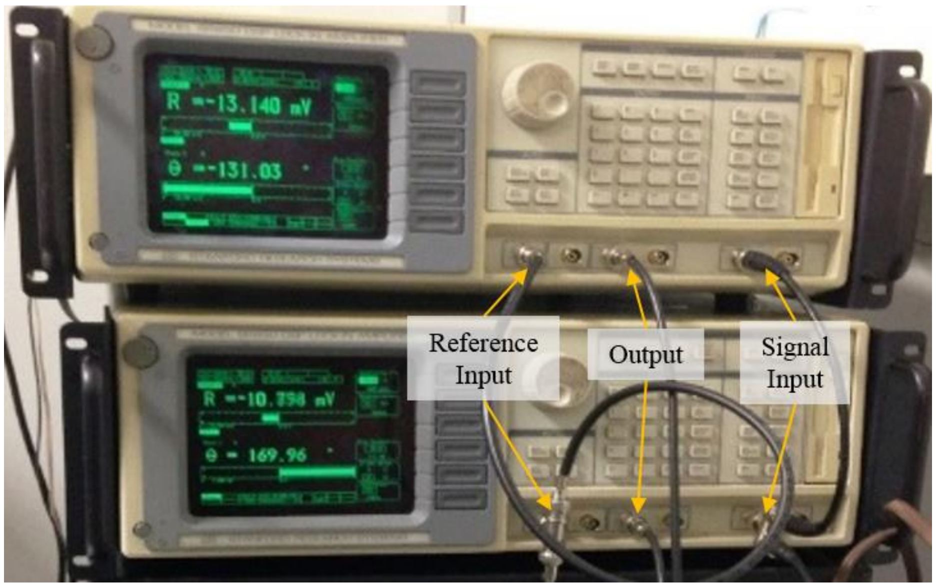


Figure 5-9- DSP lock-in amplifiers model SR850 [83].

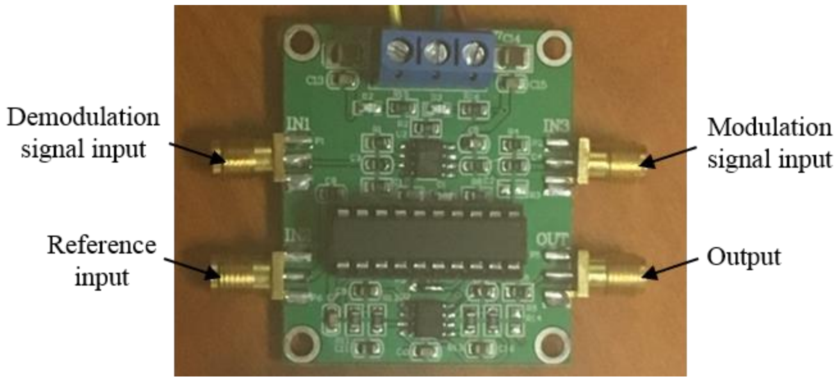


Figure 5-10- Compact lock-in amplifier circuit using AD630 microchip [87].

5.3 Harmonics detection by FPGA-based Fourier transform

FPGA technology provides a vehicle for ultrafast parallel signal processing. Computationally the special orthogonality property of Fourier transform over singular or multiple complete period/s of detection frequency [88] provides a unique opportunity to substantially increase the detection bandwidth in our system. Combination of orthogonality property of synchronous discrete Fourier transform and the unprecedented speed of parallel signal processing in FPGAs has been the main incentive to develop an FPGA-based synchronous discrete Fourier transform code to extract the harmonics of modulation frequencies in our system. Furthermore, as it was explained in section 3.4.3, the AD9833 DDS can generate modulation frequency of $f_m = 10^7/32 = 312500$ Hz without any truncation error due to its look-up table limits. Therefore, if a 32 sample Fourier transform is implemented in FPGA that is synchronously being timed with the same 10 MHz atomic clock, then this number of samples used in the Fourier transform covers exact periods of the modulation frequency. This guarantees the compatibility of the implemented Fourier transform in FPGA with the orthogonality feature of Fourier transform.

To calculate displacement from an interferometer sensor, the temporal phase of selected modulation harmonic pairs should be extracted from the measured interference intensity. Individual harmonic frequency amplitudes are measured by applying discrete Fourier transform algorithm to the signal from the photodetector. The discrete Fourier transform is developed by a VHDL code on an FPGA microchip that is being timed with the same atomic clock that is timing the FM signal generator to synchronize all the timings in the system. The discrete Fourier transform algorithm has been developed on the commercial FPGA board model NEXYS4 DDR [89], see Figure 5-11.

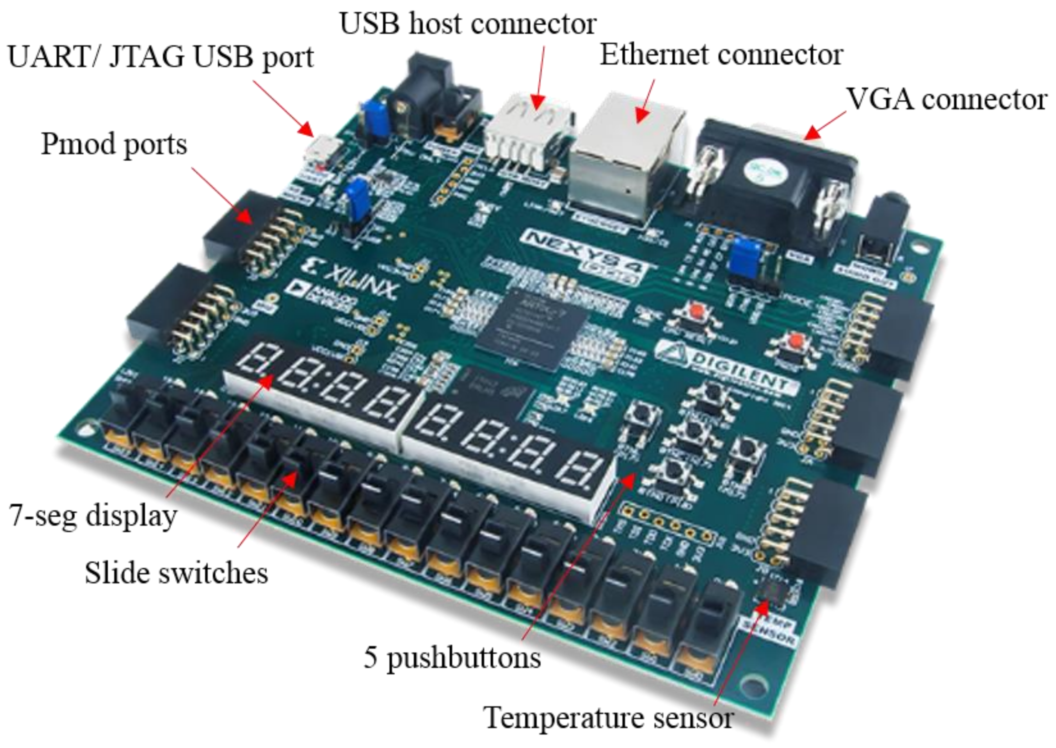


Figure 5-11- NEXYS 4 DDR board.

The idea behind Fourier transforms is that if $x(t)$ is a periodic function with period, T, it is possible to expand this into a series of harmonics with integer multiple frequencies:

$$x(t) = a_0 + 2\sum_{k=1}^{\infty}\left(a_k \cos\left(\frac{2\pi kt}{T}\right) + b_k \sin\left(\frac{2\pi kt}{T}\right)\right), \tag{5-1}$$

where,

$$a_k\,|_{k\geq 0} = \frac{1}{T}\int_0^T x(t)\cos\left(\frac{2\pi kt}{T}\right)dt, \quad \& \quad b_k\,|_{k\geq 0} = \frac{1}{T}\int_0^T x(t)\sin\left(\frac{2\pi kt}{T}\right)dt\ . \tag{5-2}$$

If we use complex notation, equations (5-2) can be combined into a single equation using

$$X_k = a_k - ib_k,$$
$$e^{-i(2\pi kt/T)} = \cos\left(\frac{2\pi kt}{T}\right) - i\sin\left(\frac{2\pi kt}{T}\right), \qquad (5\text{-}3)$$

to give

$$X_k\,|_{k\geq 0} = \frac{1}{T}\int_0^T x(t)e^{-i(2\pi kt/T)}dt \ . \qquad (5\text{-}4)$$

Now, if the continuous time-series $x(t)$ is not known and only equally spaced samples are available as $\{x_r\}, r = 0,1,2,...,(N-1)$, where $t = r\Delta$, and $\Delta = T/N$. In such case, the integral in equation (5-4) may be approximated by summation

$$X_k = \frac{1}{T}\sum_{r=0}^{N-1} x_r e^{-i(2\pi k/T)(r\Delta)}\Delta . \qquad (5\text{-}5)$$

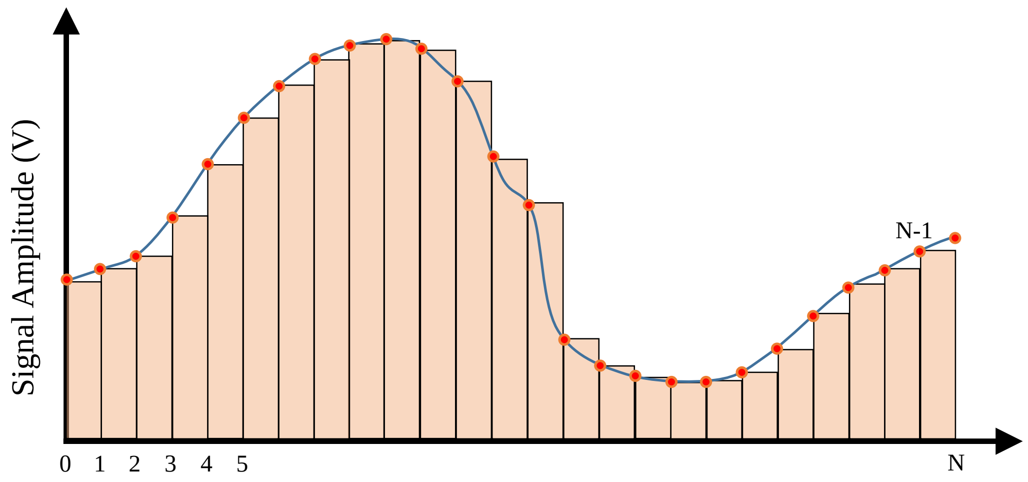


Figure 5-12- Riemann sum approximation involved in calculation of Fourier coefficients from a discrete series, x_r, rather than a continuous function, $x(t)$.

Assuming the equations (5-4) and (5-5) result in equal values for X_k is identical to assuming that the total area under the curve shown in Figure 5-12 is given by the sum of all the shaded strips [90]. Substituting $T = N\Delta$ into (5-5) gives the approximate formula for the Fourier transform, i.e. discrete Fourier transform, as

$$X_k = \frac{1}{N}\sum_{r=0}^{N-1} x_r e^{-i(2\pi kr/N)} \qquad (5\text{-}6)$$

The discrete Fourier transform that is implemented in the FPGA microchip uses the equation (5-6) at each time point and sweeps through the measured data, x_r to calculate the Fourier coefficients through the time. This means the coefficients calculated from the series $x_0, x_1, ..., x_{N-1}$ are considered to be at $t = t_0$, those calculated from the series $x_1, x_2, ..., x_N$ belong to $t_1 = t_0 + \Delta$, and those calculated from the series $x_2, x_2, ..., x_{N+1}$ belong to $t_2 = t_1 + \Delta$. This algorithm gives a method to calculate the Fourier coefficients with time interval of Δ if the FPGA microchip is fast enough to execute all the calculations for equation (5-6) in Δ timeframe. Schematic of the detected signal, x_r, path through different signal processing units of FPGA to extract the first harmonic of f_m, R_1, is shown in Figure 5-13.

In the schematic shown in Figure 5-13, the numbers $\{0,1,2,...,15\}$ represent $r = \{0,1,2,..., N-1\}$ in equation (5-6). Also, the harmonic frequency in this algorithm is determined by the update rate of x_r divided by the number of samples used in Fourier transform, N. For example, if the photodetector ADC output rate is 10 MHz and a 32 sample Fourier transform algorithm is implemented, the detection frequency will be equal to 312.5 kHz. Averaging the measured signal before applying the Fourier transform can reduce the high-frequency content of the signal in cost of a small time delay in the processed signal. Also averaging the calculated coefficients through the time can reduce some of the noise in the calculated Fourier coefficients.

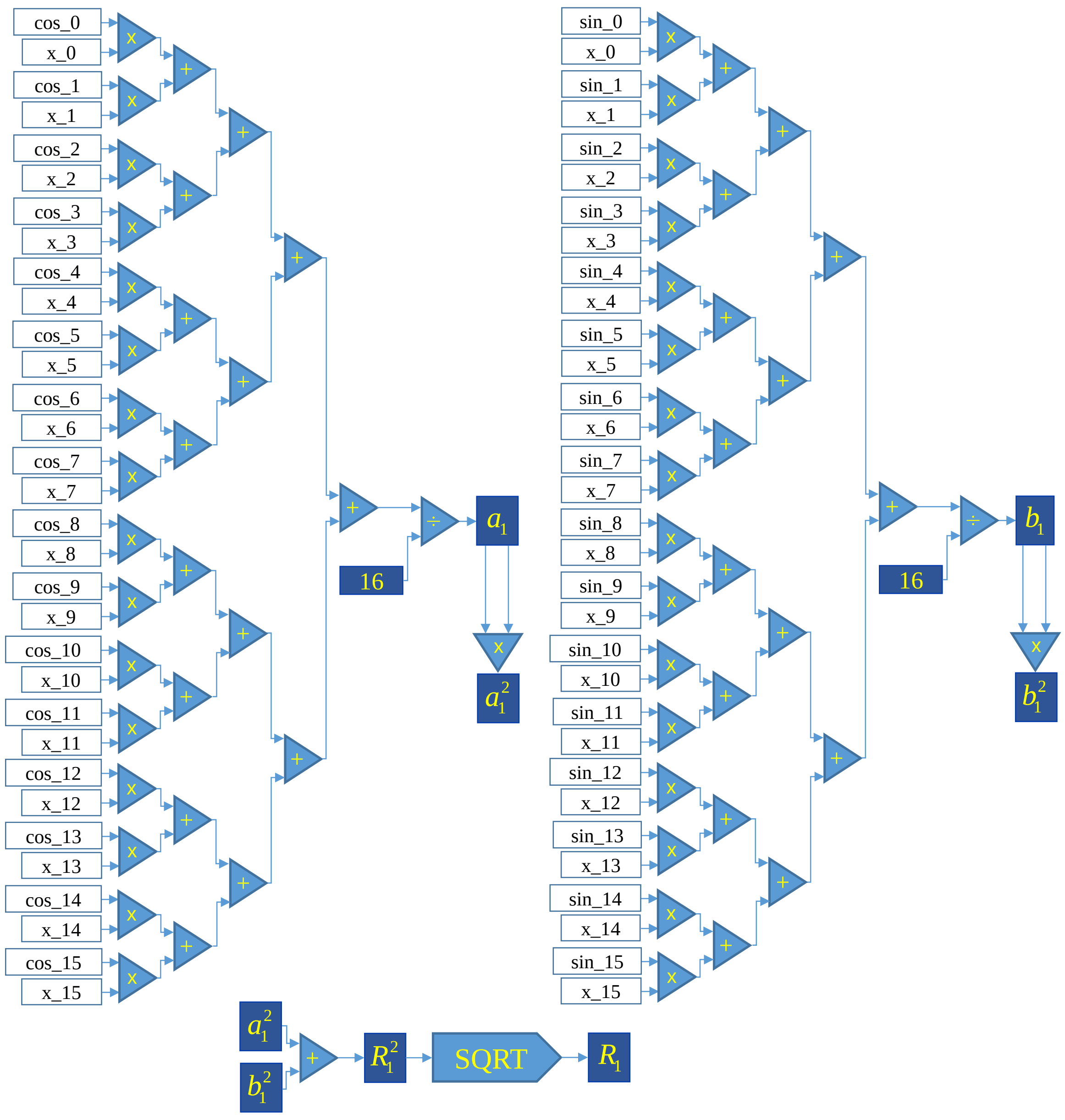


Figure 5-13- Schematic of FPGA-based, 16 sample, discrete Fourier transform for extraction of the amplitude of first harmonic in the detected signal, x_r .

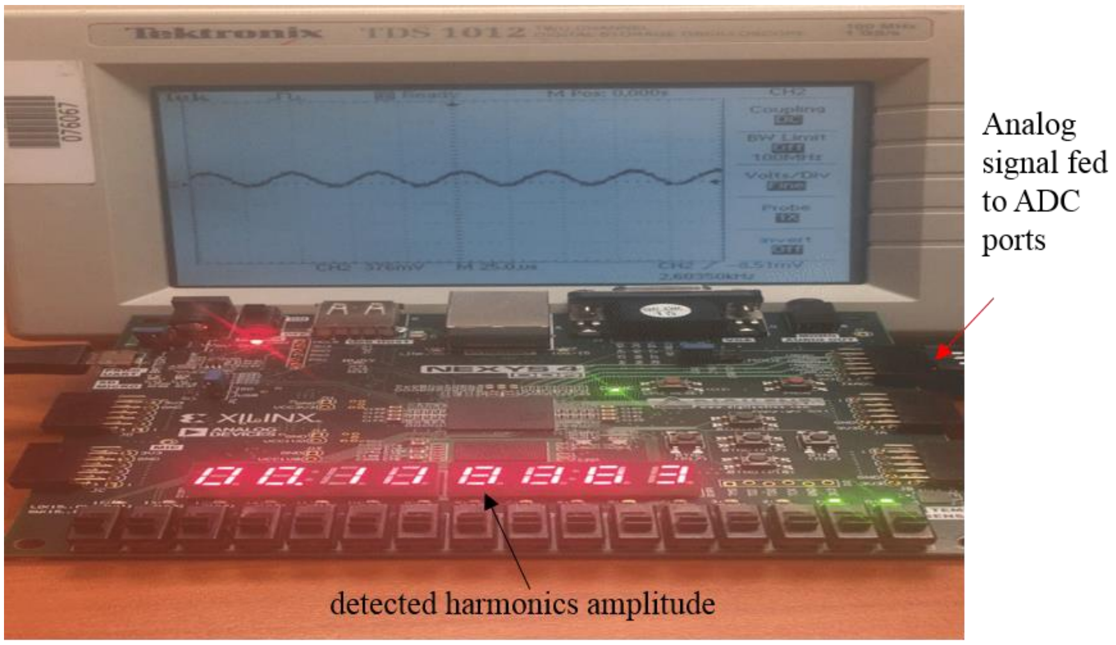


Figure 5-14- Experimental setup to measure the frequency response of discrete Fourier transform.

To verify that the Fourier transform algorithm that is being implemented in the NEXYS 4 DDR board's FPGA is correct, different VHDL codes for different sampling rates and numbers of Fourier transform algorithm were programmed in Xilinx Vivado software [91]. To evaluate the performance of the implemented Fourier transform algorithms, an experiment was set up, see Figure 5-14. Two important characteristics of the discrete Fourier transform algorithm are evaluated with this experimental setup: 1) frequency dependency of the detection bandwidth, 2) phase extraction speed limit. The frequency-dependency of the Fourier transform detection bandwidth is assessed by programming the on-board 1 MHz ADC of the FPGA to detect the amplitude of the signal that is connected to the analog Pmod pins of the NEXYS 4 DDR board. The frequency and amplitude of the signal are changed and the detected values are demonstrated in the 7-segment LEDs of the FPGA board. Results of this experiment are shown in Figure 5-15 and Figure 5-16 for detection frequencies of 25 kHz and 80 kHz, respectively.

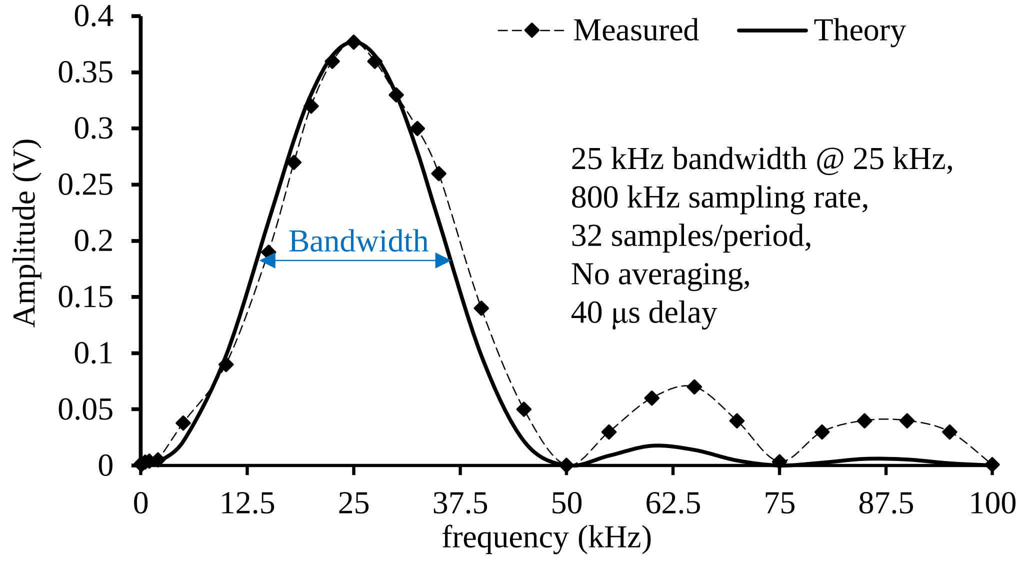


Figure 5-15- FPGA-based Fourier transform spectrum response for 25 kHz signal.

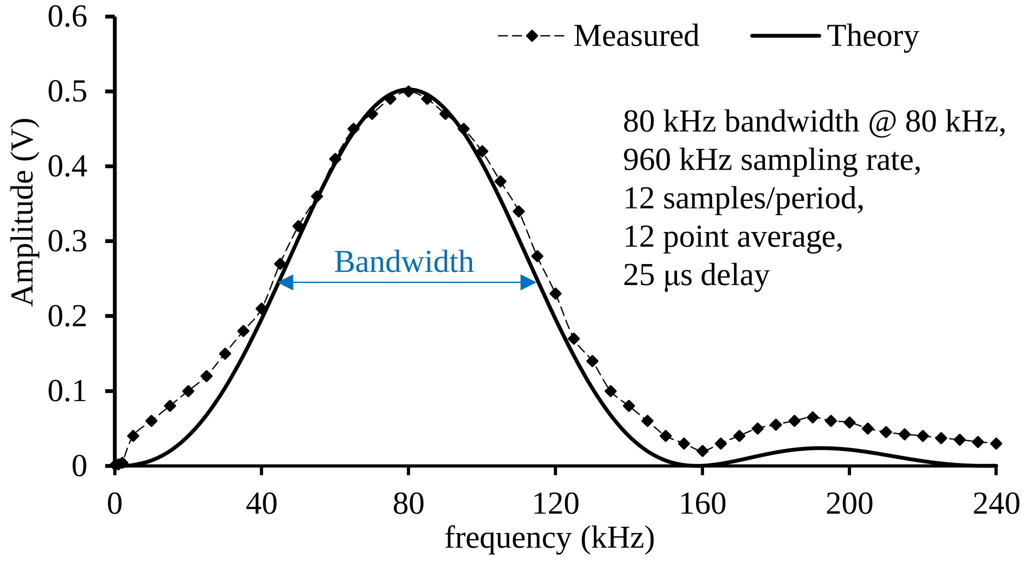


Figure 5-16- FPGA-based Fourier transform spectrum response for 80 kHz signal.

To evaluate the performance of the applied discrete Fourier transform algorithm, the theoretical response of a digital discrete Fourier transform algorithm is presented in

Figure 5-15 and Figure 5-16. As seen in these figures, the measured frequency response is compatible with theory. Also, the detection bandwidth of the discrete Fourier transform is highly dependent upon the detection frequency. The averaging after calculation of the Fourier coefficients affect the detection bandwidth if the number of averaging is higher than number of samples, N , as well. This averaging affects the phase of the detected frequency that can be determined from Lissajous curves showing the Fourier coefficients of first and second harmonic of modulation frequency versus each other, see Figure 5-17 to Figure 5-24. To evaluate the phase extraction speed limit of the Fourier transform, the simulation modules of the Vivado software was used as it is discussed here. First a 12-point sampling, 12-point averaging, 960 kHz sampling rate Fourier transform algorithm was programmed in Vivado software to detect two frequencies of 80 kHz and 160 kHz as the first and second harmonics of the desired modulation frequency, $f_m = 960/12 = 80$ kHz. Then, the ADC part of the code was deactivated and a ".txt" formatted simulation file was generated to mimic the detected signal inside the FPGA microchip. This signal contains a series of data generated by

$$x_r = \sin\left(\phi - \pi/8\right) \times \sum_{p=1}^{3} \sin\left(p\omega_m t_r\right) + \cos\left(\phi\right) \times \sum_{p=1}^{3} \cos\left(p\omega_m t_r\right) + Noise, \qquad (5\text{-}7)$$

where, ϕ is the displacement related optical phase that is a sinusoidal function of time represented by $\phi = \left(A/\omega\right) \times \sin\left(\omega t_r\right)$, ω_m is the angular modulation frequency, and *Noise* is a random noise with maximum signal-to-noise amplitude ratio of 0.1, i.e. -20 dBm. Results of this on-chip simulation for maximum speeds of 15 μm·s^{-1}, 1 mm·s^{-1}, 5 mm·s^{-1}, 7 mm·s^{-1}, and 10 mm·s^{-1} are shown in Figure 5-17 to Figure 5-21, respectively.

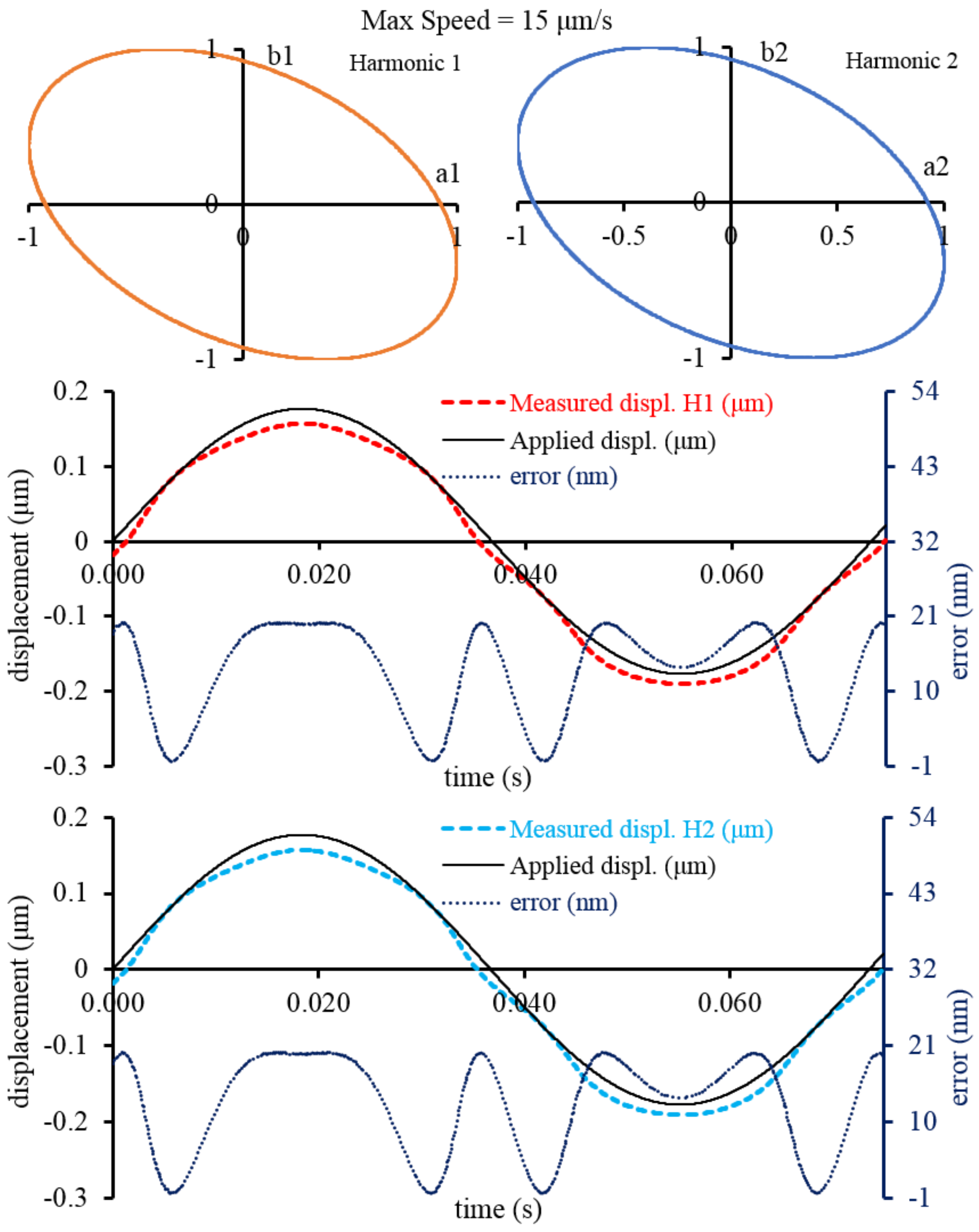


Figure 5-17- Lissajous of Fourier transform coefficients of first and second harmonics, and displacement calculated from these harmonics coefficients, compared with the applied displacement for maximum displacement speed of 15 µm·s^{-1}.

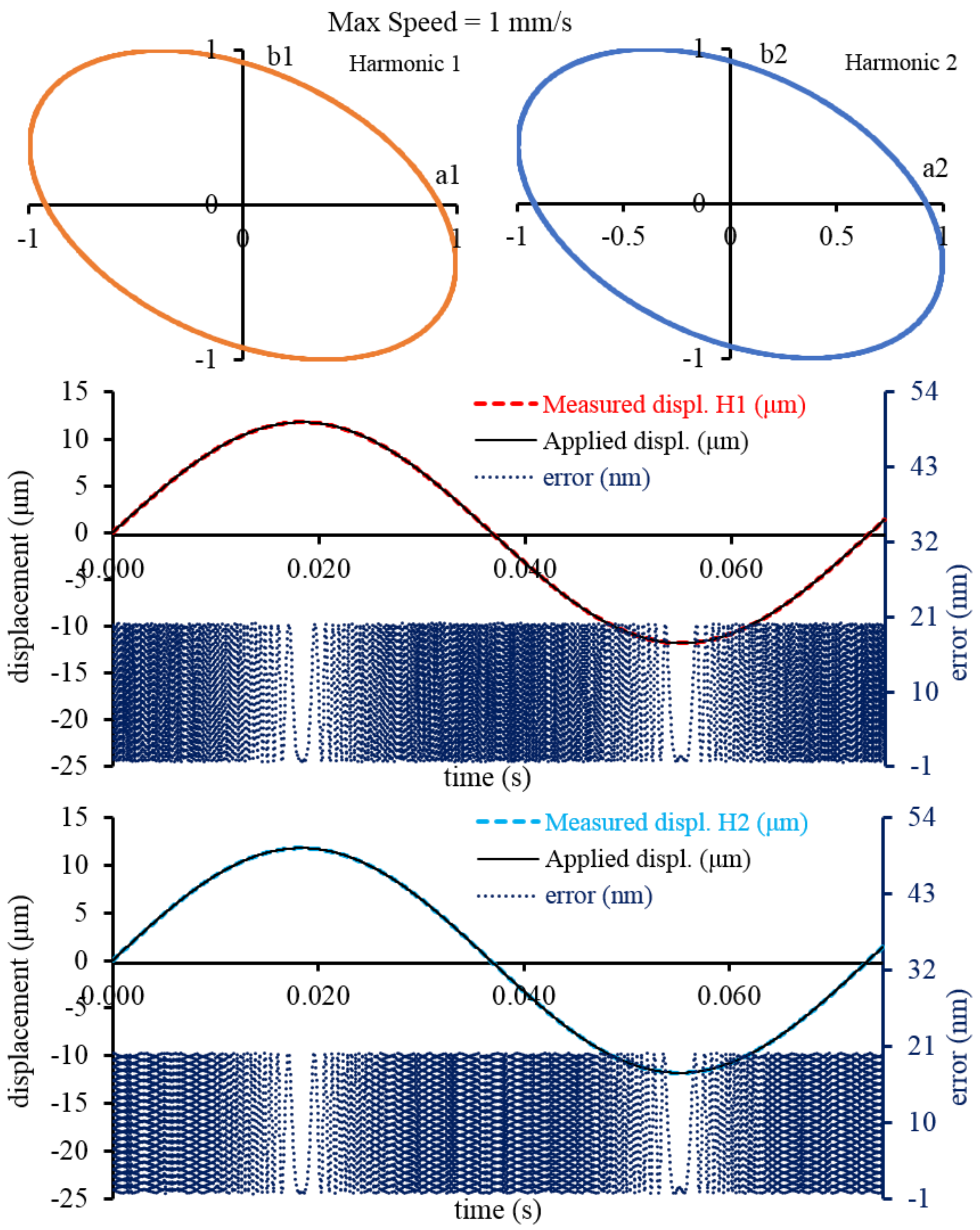


Figure 5-18- Lissajous of Fourier transform coefficients of first and second harmonics, and displacement calculated from these harmonics coefficients, compared with the applied displacement for maximum displacement speed of 1 mm·s^{-1}.

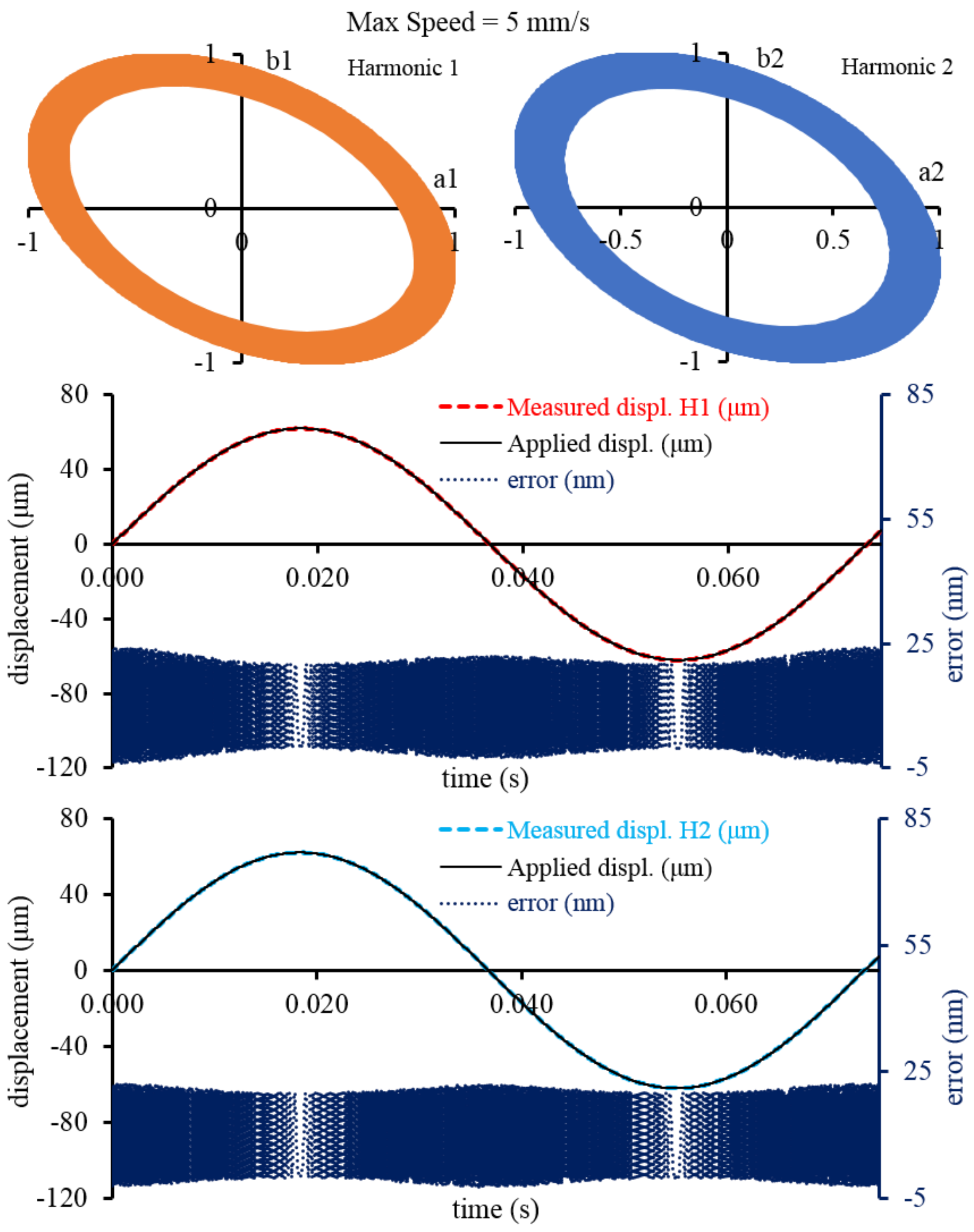


Figure 5-19- Lissajous of Fourier transform coefficients of first and second harmonics, and displacement calculated from these harmonics coefficients, compared with the applied displacement for maximum displacement speed of 5 mm·s^{-1}.

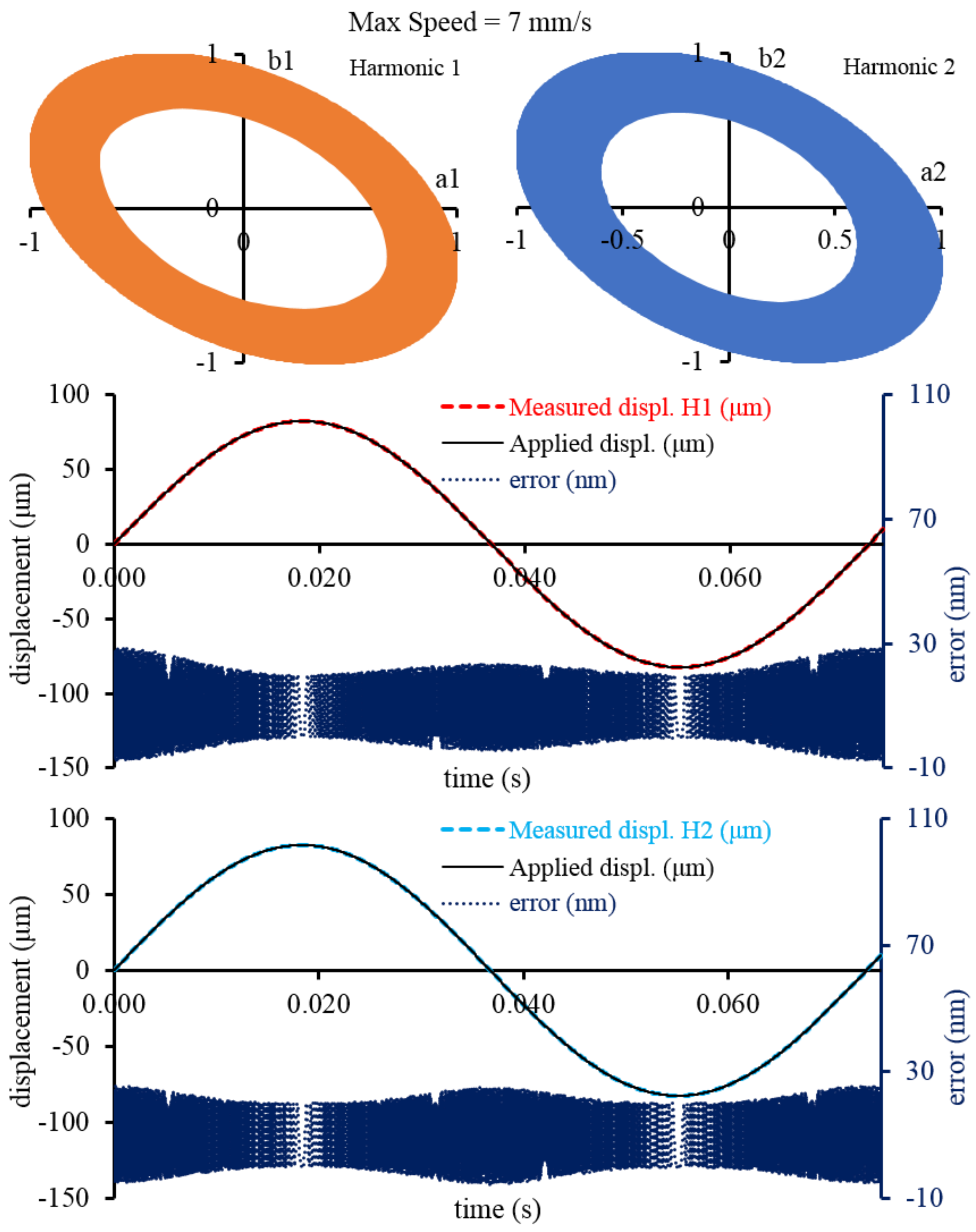


Figure 5-20- Lissajous of Fourier transform coefficients of first and second harmonics, and displacement calculated from these harmonics coefficients, compared with the applied displacement for maximum displacement speed of 7 mm·s^{-1}.

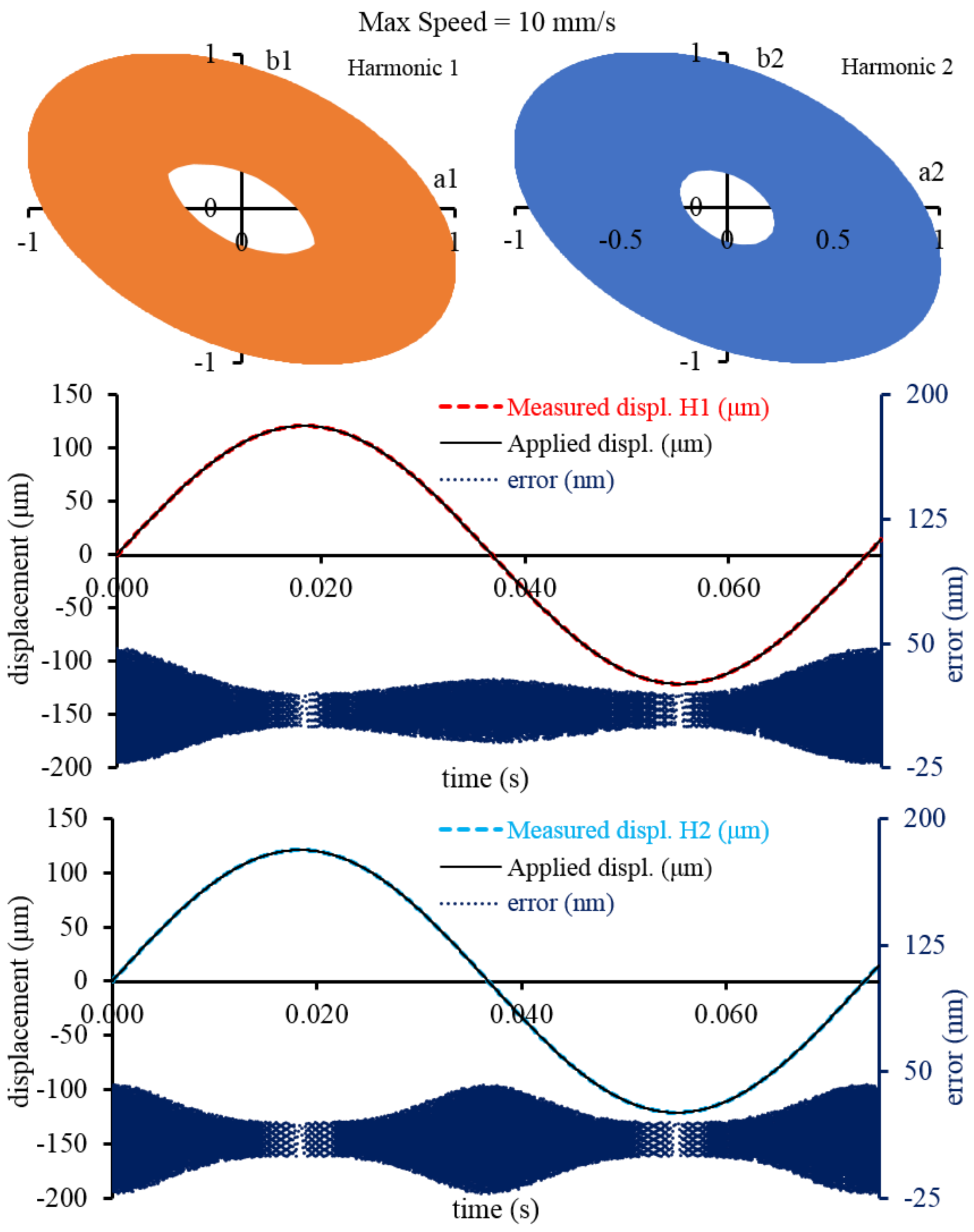


Figure 5-21- Lissajous of Fourier transform coefficients of first and second harmonics, and displacement calculated from these harmonics coefficients, compared with the applied displacement for maximum displacement speed of 10 mm·s^{-1}.

As seen in Figure 5-17 to Figure 5-21, the Lissajous curves transfer from an ellipse to an annulus by increasing the speed of displacement. Since the accuracy of the calculated phase from the Lissajous curves decreases by the growth of the annulus width, the calculated displacements show larger deviations from the applied displacement at higher speeds. These simulation reveal that the detection speed limit of the implemented Fourier transform to be around 7 mm·s^{-1} for 1 MHz ADC sampling rate. The speed of phase calculation by the Fourier transform can be boosted by either using lower numbers of samples per period of modulation frequency or by increasing the sampling rate of the ADC. The prior method results in a wider detection bandwidth and in its ultimate limit causes aliasing issues if the sampling frequency is less than 2 times the detection frequency.

In this research the second method was used to increase speed of measurement. To this end, the second photodetector was used to provide sampling rates up to 20 MHz. However, the digital signal processing unit that calculates square root of the amplitudes of the harmonics inside the FPGA microchip limits the phase calculation speed to 10 MHz. Therefore, to evaluate the speed of displacement measurement in the FPGA board a 10 MHz sampling rate Fourier transform algorithm with 32 samples per period followed by 32 points averaging was implemented and the on-chip simulations were conducted for this code as well. The range of error vs. the speed resulted from simulations by the 12-sample, 80 kHz bandwidth Fourier transform and 32-sample, 312.5 kHz bandwidth Fourier transform are shown in Figure 5-22. Also, the results of on-chip simulations of 32-sample Fourier transform for speeds of 20 mm·s^{-1} and 30 mm·s^{-1} are presented in Figure 5-23 and Figure 5-24, respectively. As seen in these three figures, the speed limit has increased up to 20 mm·s^{-1} with phase-unwrapping-related displacement errors of less than 20 nm. The 32-sample Fourier transform is used in real measurements. The main module of the VHDL code for 32-sample Fourier transform is presented in APPENDIX C. Also, the VHDL code that was developed in Vivado 2018.3, the LabVIEW code that was generated in LabVIEW 2017, and all the Arduino codes are attached to this document for future use.

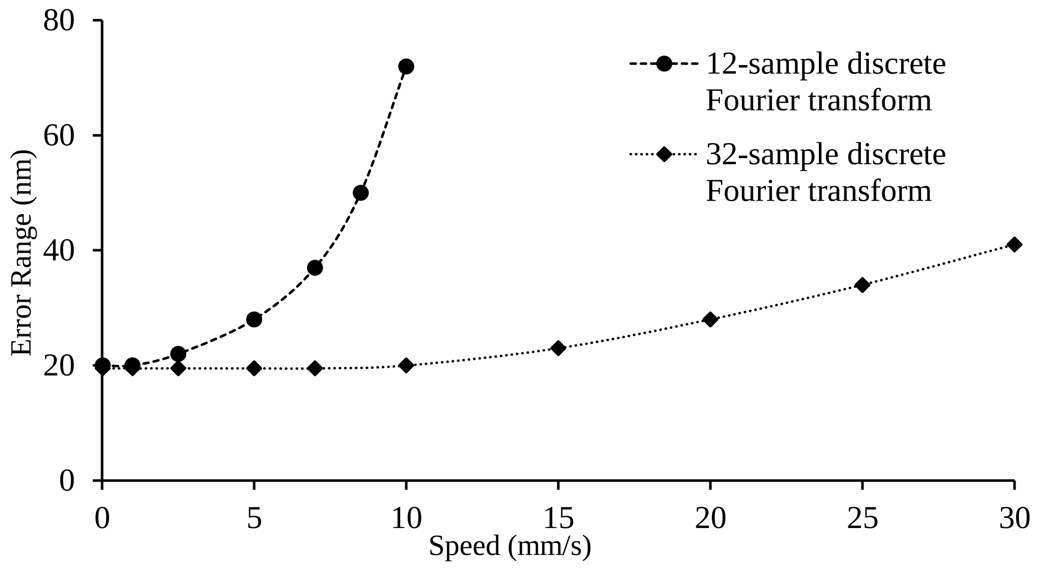


Figure 5-22- Error of 12-sample and 32-sample Fourier transforms vs. different speeds.

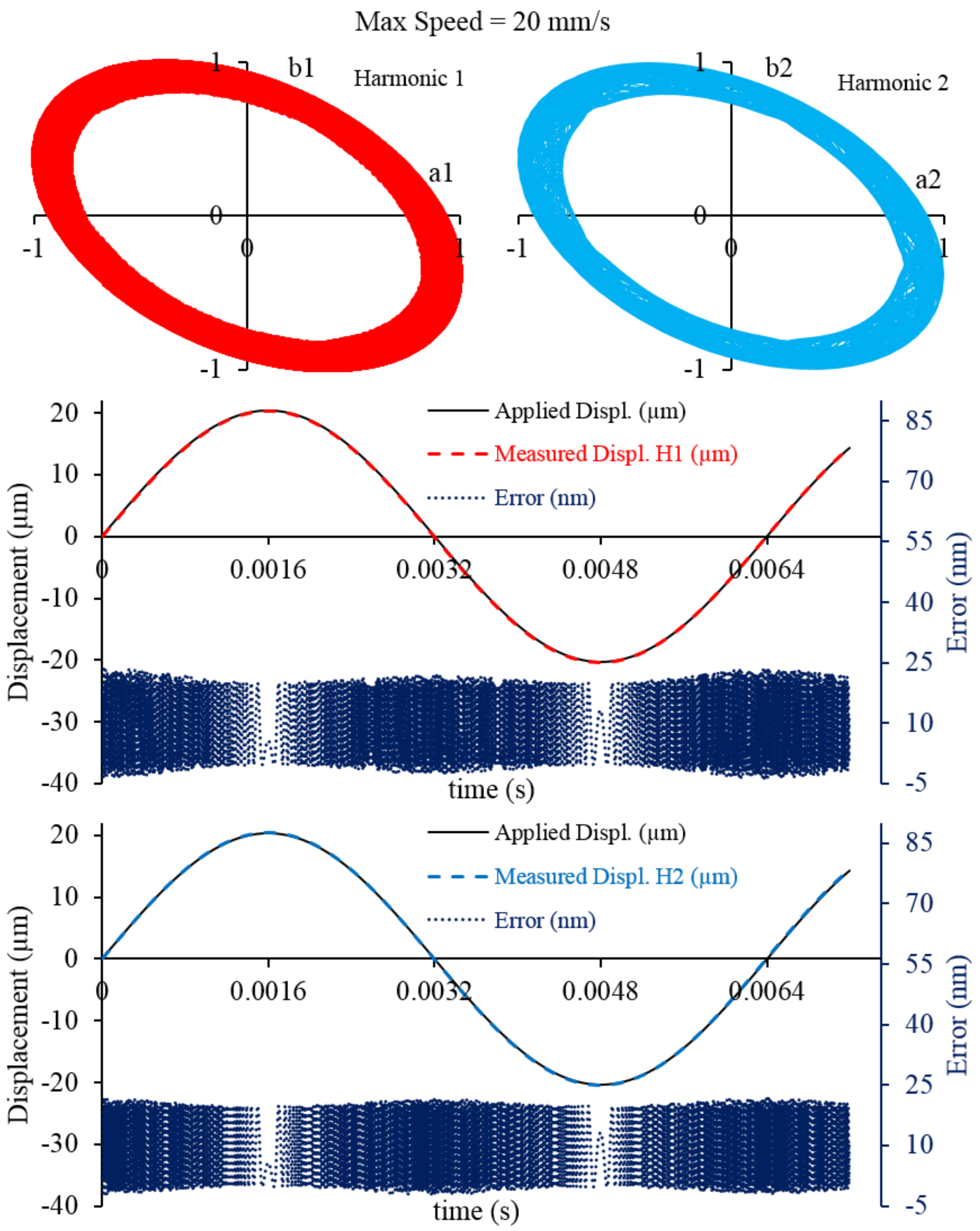


Figure 5-23- Lissajous of 32-sample, 32-averaged Fourier transform coefficients of first and second harmonics, and displacement calculated from these harmonics coefficients, compared with the applied displacement for maximum displacement speed of 20 mm·s^{-1}.

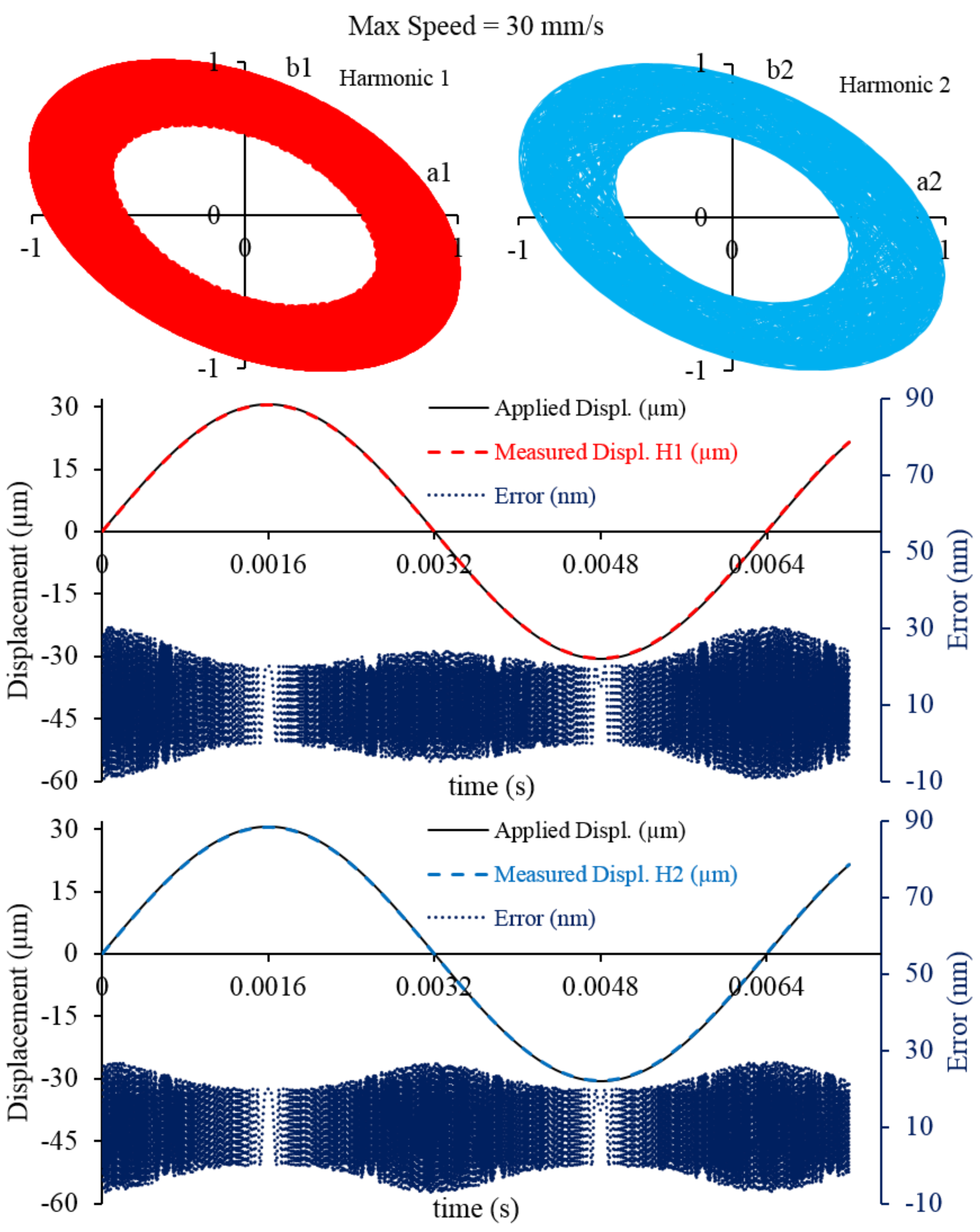


Figure 5-24- Lissajous of 32-sample, 32-averaged Fourier transform coefficients of first and second harmonics, and displacement calculated from these harmonics coefficients, compared with the applied displacement for maximum displacement speed of 30 mm·s^{-1}.

CHAPTER 6 DISPLACEMENT MEASUREMENT

This chapter consists of four sections that present the displacement measurements by standard laboratory equipment and analog demodulator circuits in section 6.1, displacement measurement results using FPGA-based discrete Fourier transform in section 6.2, synchronization issues between LabVIEW FPGA and the NEXYS4 DDR board in section 6.3, and suggestions for future works in section 6.4.

6.1 Measurements by analog lock-in amplifiers

The first experimental setup to evaluate displacement measurement by different pairs of modulation harmonics extracted by DSP lock-in amplifiers is shown in Figure 6-1. This setup comprises a piezoelectrically actuated translation stage with a moving mirror that is monitored by the polydyne interferometer. To evaluate the displacement measured by the polydyne interferometer, a commercial heterodyne laser interferometer (Hewlett Packard model 5529A [92]) is used to simultaneously measure the moving mirror as shown in Figure 6-1. The output of the photodetector shown in Figure 6-1 is connected to the signal input of the DSP lock-in amplifiers shown in Figure 5-9. Also, the modulation signal output from the FM board shown in Figure 5-13 is connected to the reference input of the lock-in amplifiers. Since the DSP lock-in amplifiers provide detection bandwidth up to 102 kHz, modulation frequencies of less than 10 kHz were used for this experiment so that up to 10 modulation harmonics can be measured. The lock-in amplifiers are set to different harmonics of the modulation frequency and the amplitude of the detected harmonics are collected using a LabView program. An example of amplitude variation of detected modulation signal harmonics while a sinusoidal displacement is injected to the moving arm of the interferometer is shown in Figure 6-2. Plotting these amplitudes versus each other results in a Lissajous curve, see Figure 6-3 for Lissajous loci of different harmonic pairs from different measurements. The spin of the Lissajous curve is dependent on the direction of the displacement.

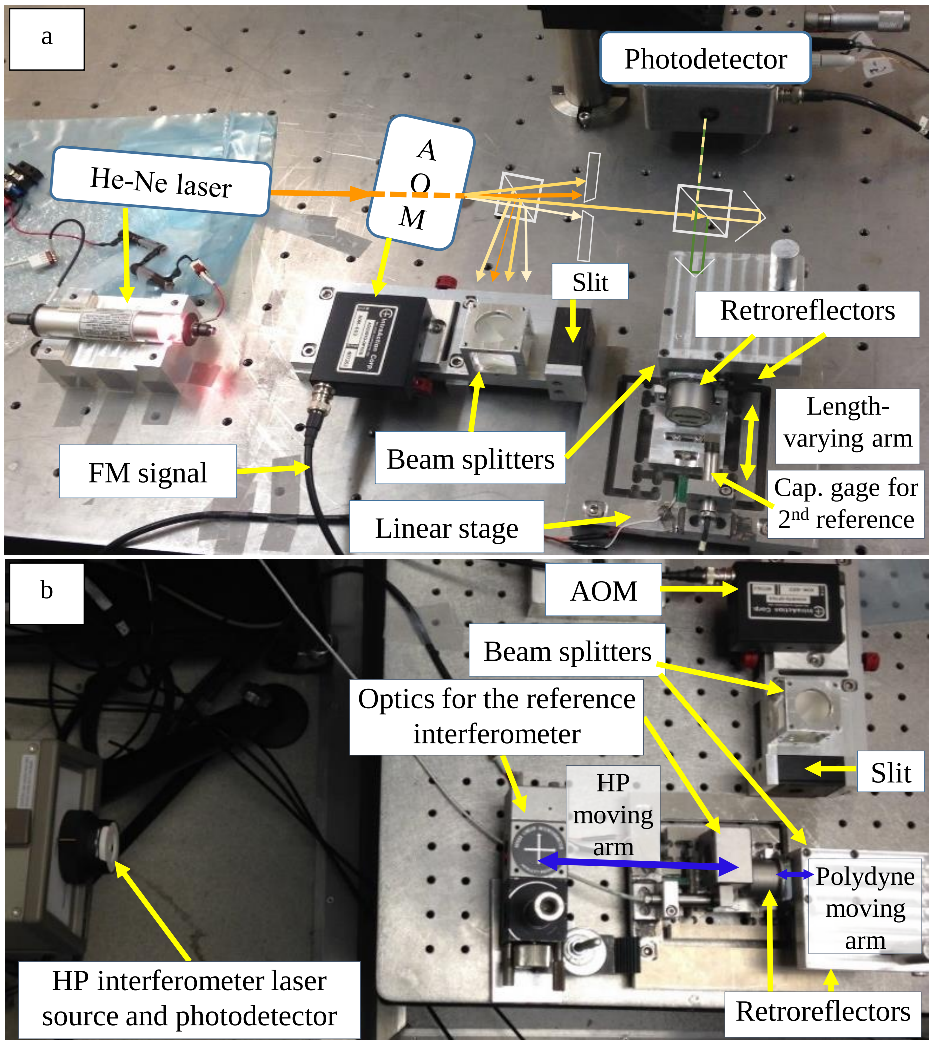


Figure 6-1- Apparatus for interferometric displacement measurement, a) polydyne interferometer setup, b) added optics for the reference interferometer.

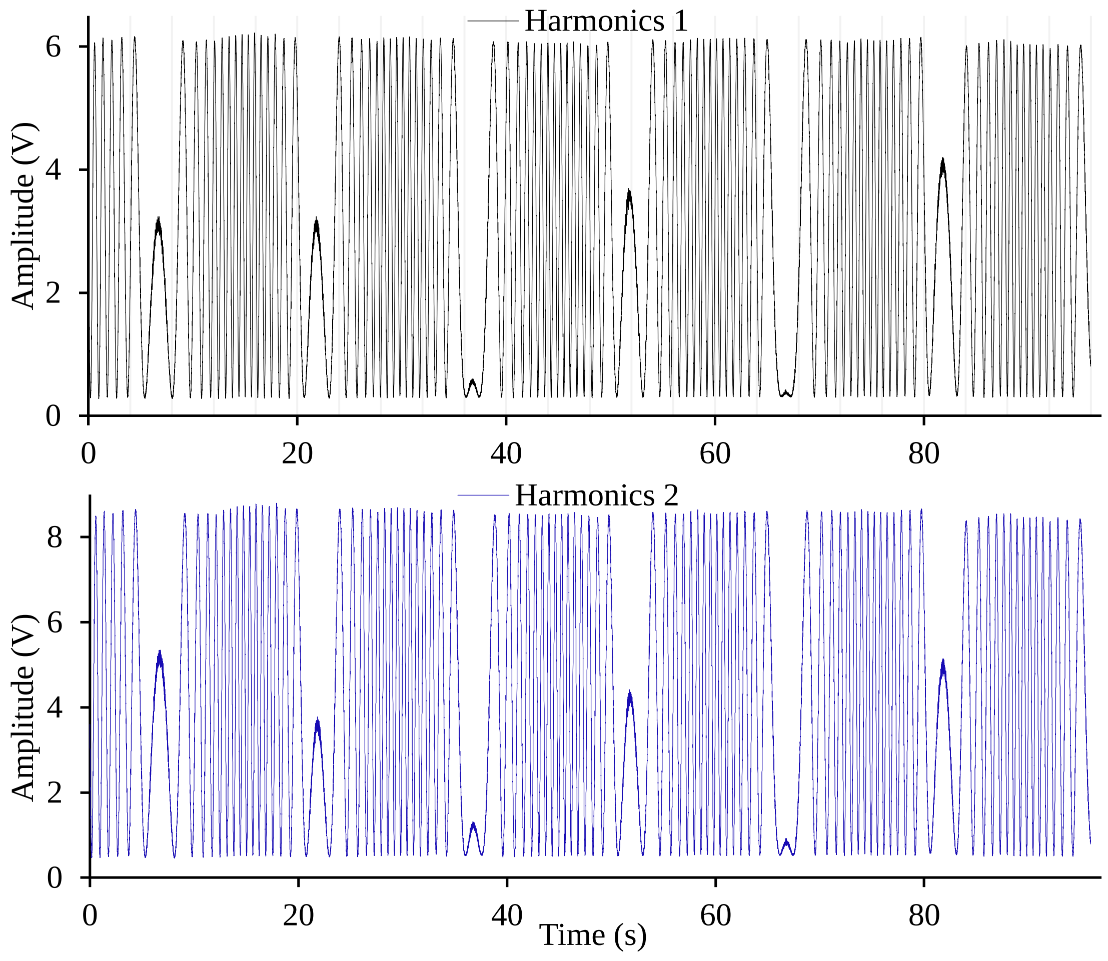


Figure 6-2- Modulation harmonics amplitude variations due to a sinusoidal displacement.

To extract the phase variations on the Lissajous curve, the elliptic curves is first normalized into a circle of unit radius, see Figure 6-4, then the phase is calculated using the quadrature method explained in CHAPTER 2. After unwrapping the measured phase, the displacement is extracted using equation (2-10). Among all the possible combinations of harmonic pairs from harmonics 1 to 5, the harmonics pair 2 and 5 (H2-H5 pair) were in-phase and did not lead to a smooth unwrapped phase. Therefore, the phase difference between them was not detectable for the unwrapping algorithm.

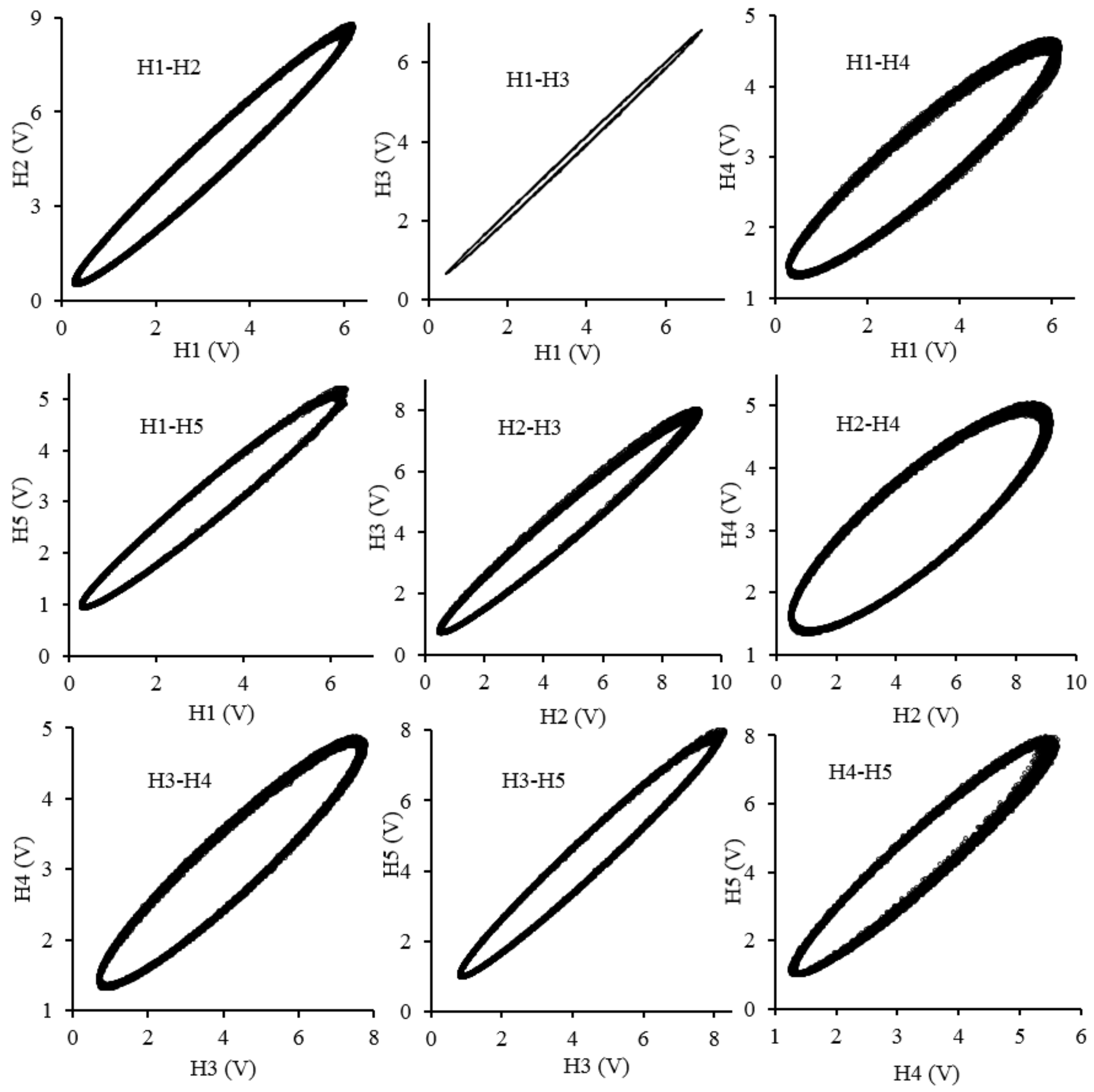


Figure 6-3- Lissajous curves from different harmonic pairs of $f_m = 10$ kHz.

To illustrate displacement measurement from the unwrapped phase, Figure 6-5 shows an example of the measured displacement with harmonic pairs (H1-H2) for FM signal of $f_c = 60$ MHz and $f_m = 5$ kHz. The displacement measured by the HP interferometer, and the difference between the HP interferometer and H1-H2 measurements (labelled deviation in this study) are also shown in this figure. The displacement measured by the reference interferometer has been artificially shifted +0.5

µm to make both the HP interferometer and the H1-H2 measurements visible in the plot. Similar data is obtained for all other harmonic pairs and the RMS of deviations for these measurements are presented in Table 6-1. These results show deviations within ±40 nm. The deviations, which contains both drift and noise, are most probably due to different metrology loops determined by the different locations of beam splitters and light sources used in the HP and polydyne interferometers. As seen in Figure 6-1, the moving arm of the HP interferometer is longer and consequently more exposed to the ambient variations than that of the polydyne interferometer. Also, the detector of the HP interferometer is located on the ground that is separated from the vibrational isolated basement of the optics table. This causes the HP interferometer to be more susceptible to environmental vibrations. All these differences between the HP and the polydyne optics setup as well as the resolution limit of the HP interferometer that is about 9 nm contribute in deviations between the measurements by these interferometers.

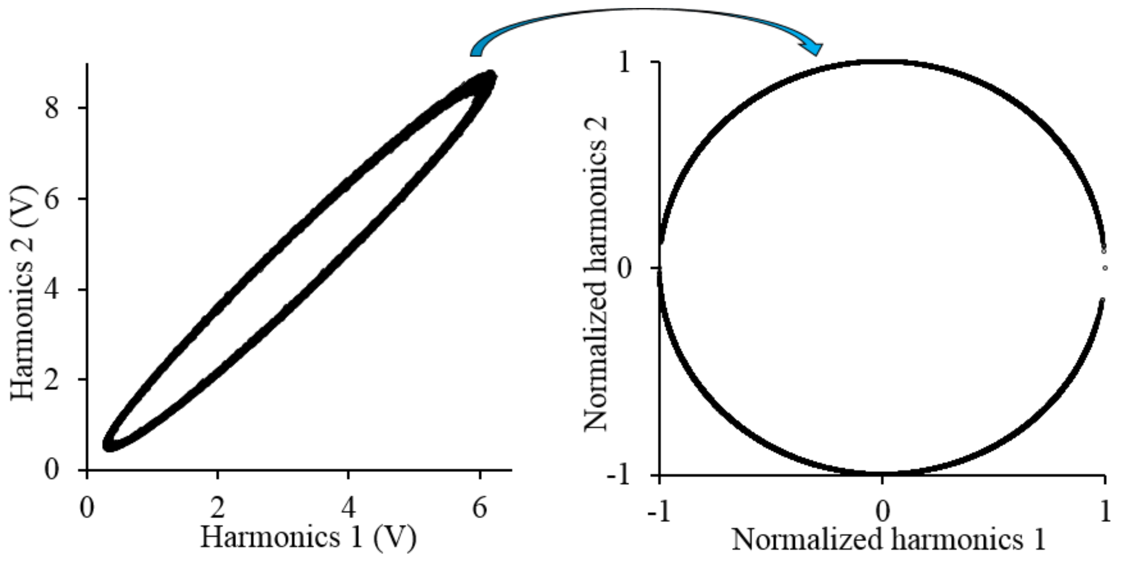


Figure 6-4- Normalized Lissajous curve.

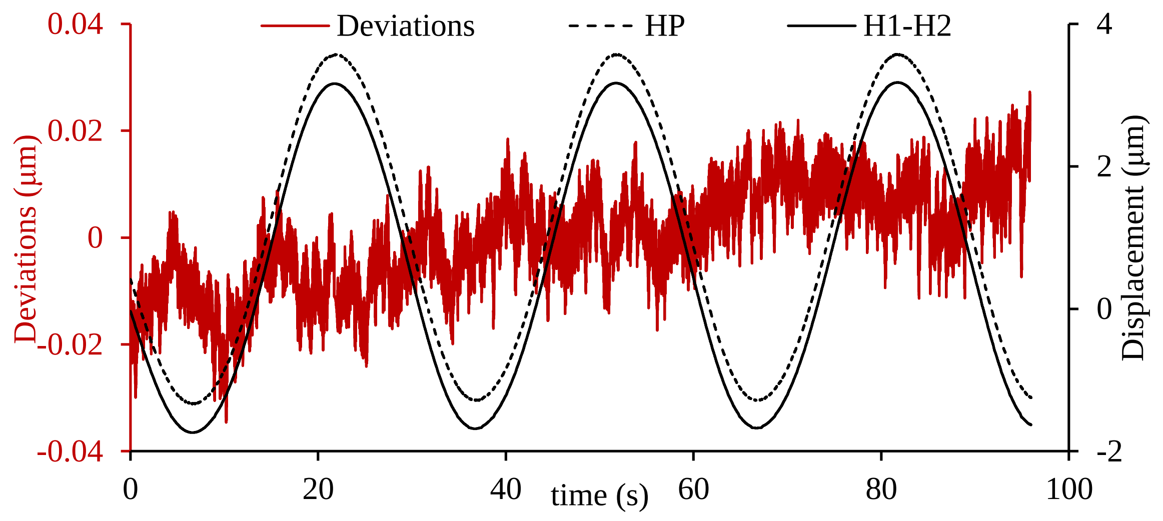


Figure 6-5- Displacement measured by reference interferometer and H1-H2.

Table 6-1- RMS of deviations between reference interferometer and harmonics pairs measurements.

Harmonic pairs	RMS of deviations (µm) for measurements by $f_m = 5$ kHz	RMS of deviations (µm) for measurements by $f_m = 10$ kHz
H1-H2	0.009	0.013
H1-H3	-	0.012
H1-H4	0.011	0.011
H1-H5	0.012	0.011
H2-H3	0.013	0.009
H2-H4	0.009	0.011
H2-H5	0.011	-
H3-H4	0.012	0.012
H3-H5	0.010	0.008
H4-H5	0.011	0.009

To evaluate the temporal stability of the system, a drift test was conducted by monitoring interferometer outputs while all components remained stationary. The optical path lengths for the reference and polydyne interferometers were 50 mm and 20 mm, respectively. In these tests the metrology loops were different and there was a small Abbe error of about 10 mm with misalignment between axes of two interferometers of up to 0.1 radians. However, for drifts of less than 1 µm these effects are likely to be insignificant. Temperature, pressure and humidity of the laboratory were monitored during the drift test. Optical path length changes due to variations of these parameters can be calculated using a modified Edlen equation [93]:

$$(L_c - L_m)/L_m = \underbrace{\left(K_T\left(T-20\right)\right)}_{A1} + \underbrace{\left(-K_P\left(P-760\right)\right)}_{A2} + \underbrace{\left(K_H\left(RH-50\right)\right)}_{A3} = A4, \qquad (6\text{-}1)$$

where, L_c is the corrected optical path length, L_m the measured optical path length, $K_T = 9.5\times10^{-7}$ (°C^{-1}), T the temperature in (°C), $K_P = 3.6\times10^{-7}$ (mmHg^{-1}), P the atmospheric pressure in (mmHg), $K_H = 5.0\times10^{-8}$ (RH^{-1}), and RH is the relative humidity. The correction factors A_1, A_2, A_3, and A_4 based on the measured temperature, pressure, and humidity are shown in Figure 6-6. Also, the drift test results are presented in Figure 6-7.

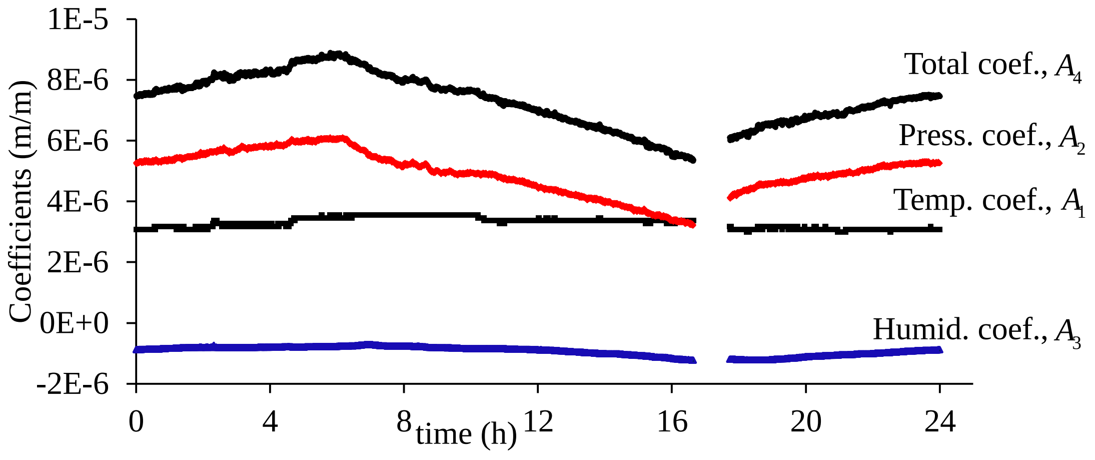


Figure 6-6- Length correction coefficients for temperature, A_1, pressure, A_2, humidity, A_3, and total, A_4.

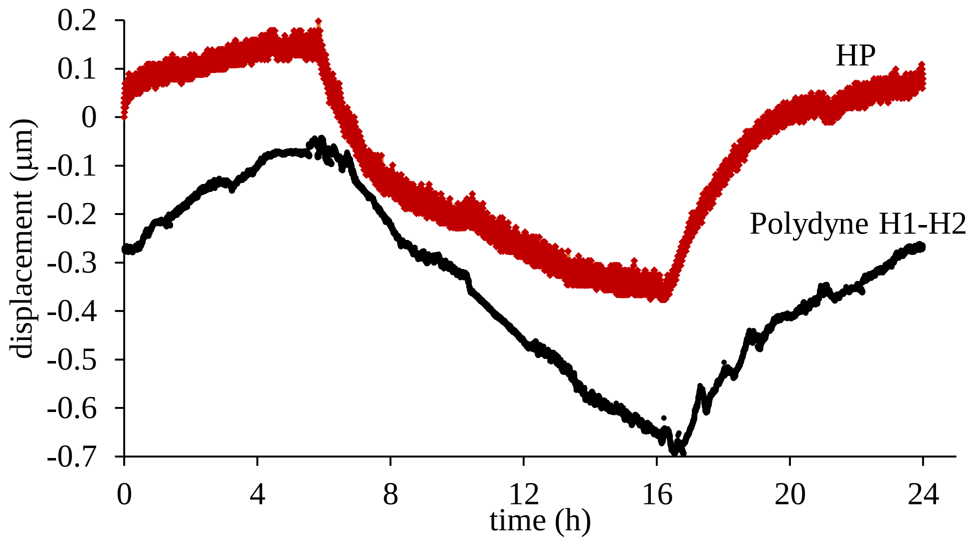


Figure 6-7- Drift test results for HP and polydyne interferometer.

From Figure 6-7, it can be seen that the displacement variations are comparable and of the order of 0.6 µm. A simple calculation for an optical path length of 20 mm and Edlen correction of 9×10^{-6} indicates a deviation of 0.18 µm. Assuming a thermal expansion of the setup comparable to aluminum (24×10^{-6} $^{o}C^{-1}$) indicates a deviation of 0.58 µm. These two effects predict 0.76 µm variation compared with an observed drift of 0.6 µm over a twenty four hour period. These calculations ignore the complexity of the translation stage and alignment mechanism that can also move during the drift test. Although the optical path length of the reference interferometer was longer, its measured drift was around 0.5 µm. During this drift experiment the laser, the AOM, and all other system components of the polydyne interferometer were mounted on an aluminum breadboard plate; this plate was, in turn, mounted onto a rigid table. For the reference interferometer, the beam splitter was mounted directly onto the rigid table, while only its moving mirror was connected onto the moving stage of the apparatus. Hence, the metrology loops for these two systems were considerably different. However, we believe that the majority of the observed deviations were due to refractive index and thermal expansion effects. Notwithstanding this, it is clear that the drift and temperature, pressure, and humidity effects are clearly correlated.

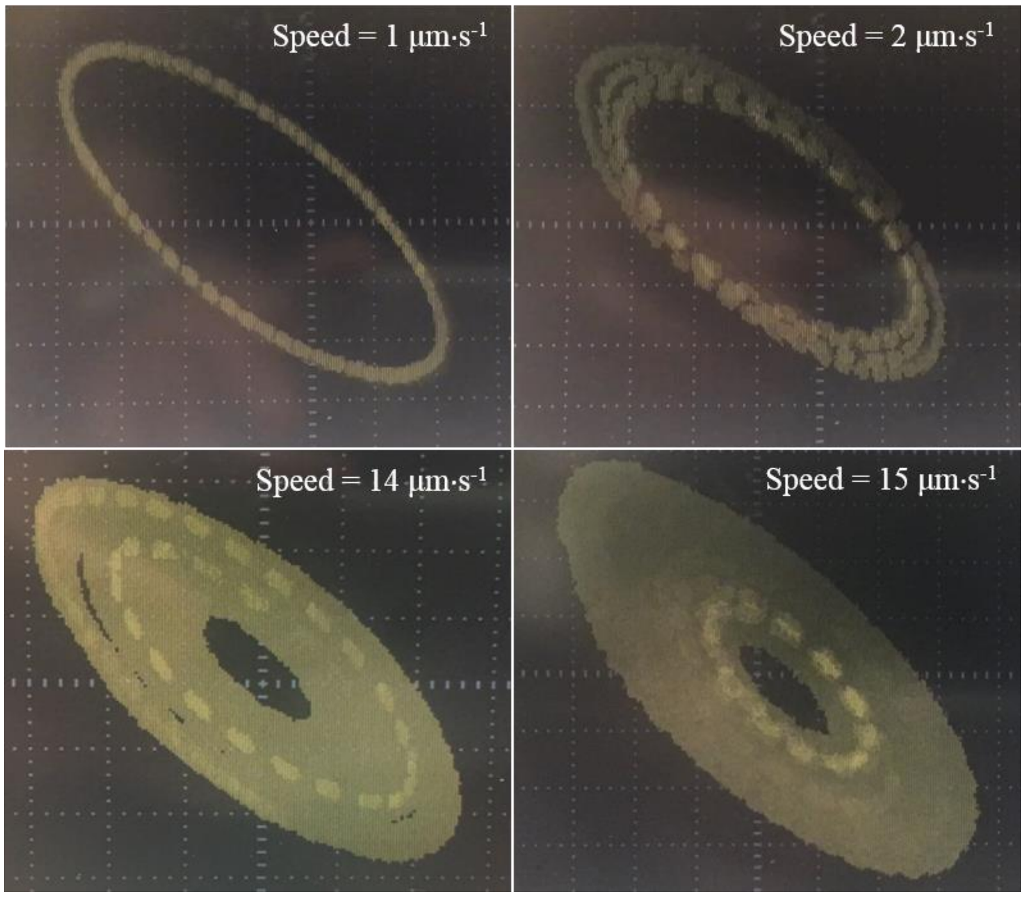


Figure 6-8- Lissajous measured by AD630 circuits at speeds of 1, 2, 14, and 15 µm·s^{-1}.

Displacement measurement from different modulation harmonics and the drift test results showed the feasibility of phase measurement by polydyning technique. However, the speed of measurement was limited to a few hundred nm.s^{-1} because of the low bandwidth of the DSP lock-in amplifiers. To increase this speed limit, an experiment similar to the one shown in Figure 6-1 was set up using AD630 circuits instead of DSP lock-ins. Results of this experiment in the form of Lissajous curves for different speeds are shown in Figure 6-8. As shown in Figure 6-8, increasing the displacement speed from 1 μm·s^{-1} to 15 μm·s^{-1} changes a reasonably precise elliptic Lissajous to a very wide annulus Lissajous. Increasing the speed further deteriorates the Lissajous shape and increases the phase unwrapping error drastically. Results of measurements by the analog circuits showed that the AD630 demodulators limit the speed of measurement to 15 μm·s^{-1}. Based on discussions with industry users, it was desirable that this limit be increased to 10 - 20 mm·s^{-1}. To address this issue the synchronous FPGA-based Fourier transform is used and the results are show in the following section.

6.2 Measurements by FPGA-based discrete Fourier transform

To test the performance of the discrete Fourier transform algorithm developed in a NEXYS 4 DDR board, an experiment was setup as shown in Figure 6-9. This experiment also uses the same HP interferometer to evaluate polydyne interferometer measurements. The phase unwrapping algorithm of this experiment is somewhat different from the previous measurements. Due to the limits in the number of digital signal processing and memory units of the FPGA microchip of the NEXYS 4 DDR board, we could not implement the discrete Fourier transform and phase unwrapping algorithm together inside the FPGA microchip. Therefore, the amplitude of two harmonics are extracted by the 32-sample Fourier transform in the FPGA and sent to a LabVIEW through the DAQ, see Figure 6-9. The digital data transfer from the FPGA to the LabVIEW uses a shared clock signal to time the data transformation. In the LabVIEW, an ellipse is fitted to the Lissajous that is developed by the amplitude of the detected harmonics through a calibration procedure. Then, the ellipse parameters are used to calculate the real-time phase of the Lissajous in a LabVIEW FPGA-code. The LabVIEW codes and the VHDL code are available in an external memory that is attached to the hard-copy of this document. All the codes are commented thoroughly to be self-explanatory for the users. Because the LabVIEW could not read 16 bit data at rising and falling edge of 20 MHz, the sampling rate of the ADC was reduced to 10 MHz that showed more stable Lissajous.

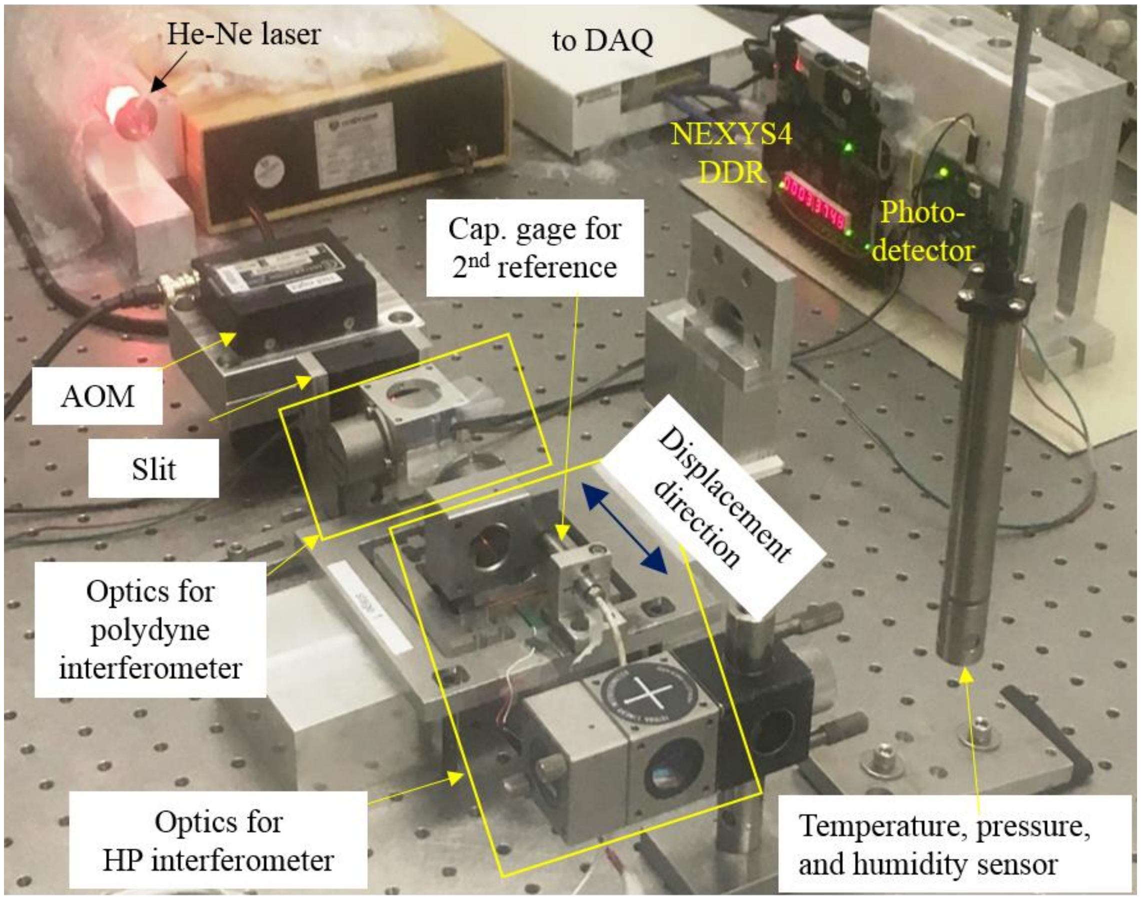

He-Ne laser
to DAQ
NEXYS4 DDR
Cap. gage for 2nd reference
Photo-detector
AOM
Slit
Displacement direction
Optics for polydyne interferometer
Optics for HP interferometer
Temperature, pressure, and humidity sensor

Figure 6-9- Experimental setup of polydyne interferometer with FPGA-based Fourier transform.

Displacements with different amplitudes and frequencies were injected to the moving arm of the interferometer and were measured by the setup shown in Figure 6-9. Examples of these Lissajous for speeds of 2 mm·s^{-1}, 4 mm·s^{-1}, 7 mm·s^{-1}, and 11 mm·s^{-1} are shown in Figure 6-10. As seen in this figure for 7 mm·s^{-1} and 11 mm·s^{-1} Lissajous curves, the averaging in the Fourier transform algorithm causes the detected harmonics amplitudes to construct two distinct Lissajous. The displacements measured at maximum speed of 2 mm·s^{-1} and the corresponding deviations between HP and polydyne measurements results are presented in Figure 6-11 and Figure 6-12, respectively.

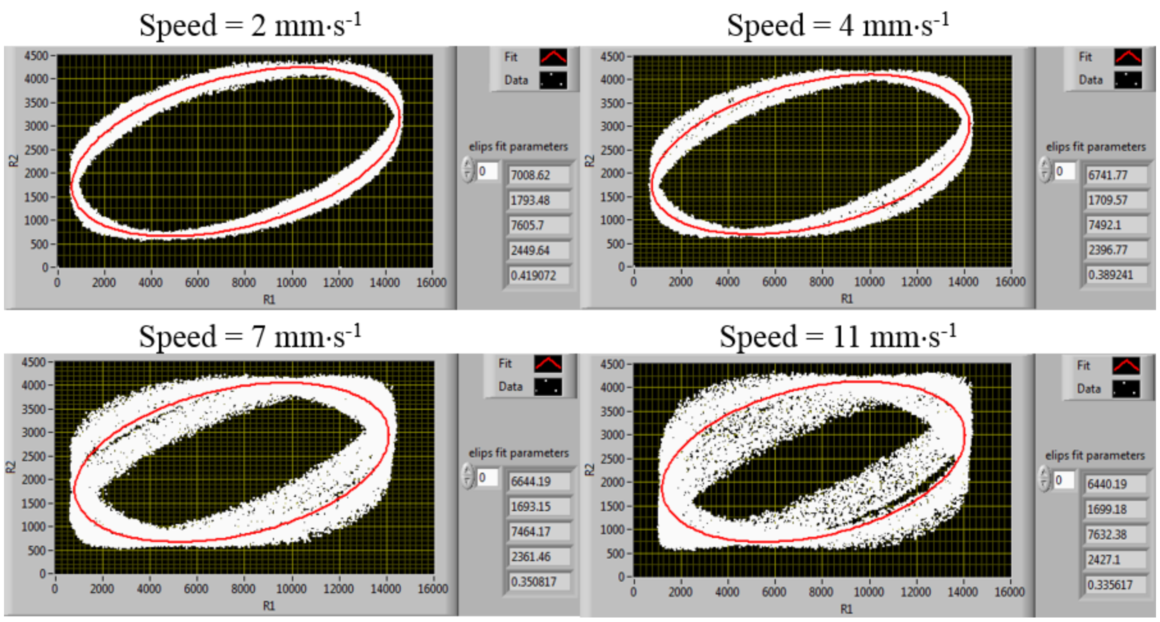


Figure 6-10- Lissajous and fitted ellipse to the results of measurements by 32-sample, 32-point-averaged Fourier transform with 10 MHz ADC sampling rate and different maximum displacement speeds of 2, 4, 7, and 11 mm·s^{-1}.

The deviations depicted in Figure 6-12 contains a variety of error components including an Abbe error due to different metrology loops and a systematic error due to non-synchronous measurements with the HP and polydyne interferometers. Notwithstanding all these error sources, the polydyne and HP interferometers provide a maximum point to point deviations with ±50 nm and RMS of 20 nm for displacement measurements up to 11 mm·s^{-1}, see Table 6-2.

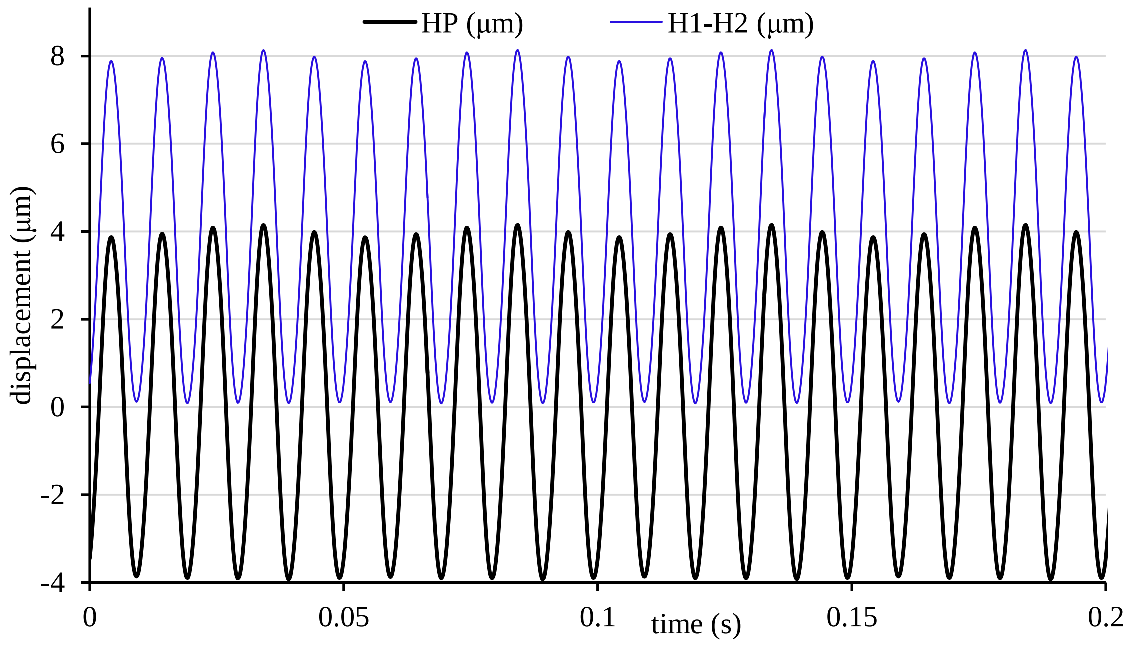


Figure 6-11- Sinusoidal displacement with maximum speed of 2 mm.s^{-1} measured by polydyne (H1-H2) and HP interferometers.

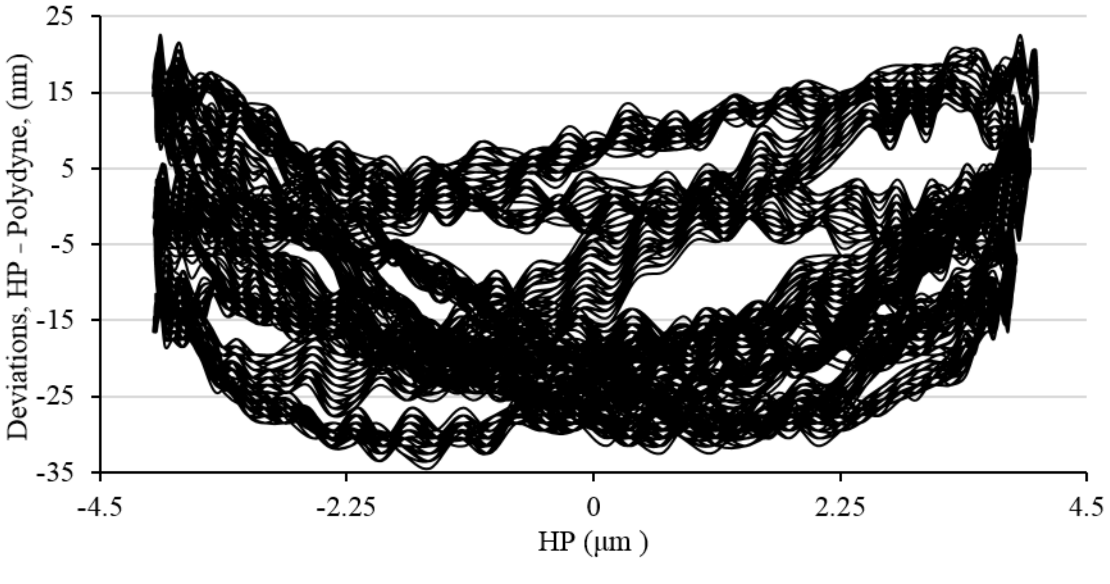


Figure 6-12- Deviations between HP and polydyne (H1-H2) displacement measurements at maximum speed of 2 mm·s^{-1}.

Table 6-2- Span and RMS of deviations between HP and polydyne interferometers measurements.

Maximum speed	Deviations span	RMS of deviations
2 mm·s^{-1}	-35 nm to 25 nm	15 nm
4 mm·s^{-1}	-35 nm to 25 nm	15 nm
7 mm·s^{-1}	-40 nm to 30 nm	18 nm
11 mm·s^{-1}	-45 nm to 40 nm	21 nm

6.3 Synchronization issues

Due to limits in the size and memory of the Artix 7 FPGA microchip that is used in NEXYS 4 DDR board, in the current experimental setup, the measured modulation harmonics amplitude is transferred from the NEXYS 4 DDR board into a National Instruments PXIe module, model NI7851R, to conduct real-time phase unwrapping on the phase of the measured harmonics amplitude. On one hand, the NI7851R does not provide external clocking option and its operation is not synchronous to the atomic clock. On the other hand, the data transfer speed between the NEXYS 4 DDR board and this NI module is very high, 320 Mbit·s^{-1}. This results in a high possibility of systematic error occurrence due to non-synchronous operation of the NI module and the NEXYS 4 DDR board, i.e. some of the transferred digital data from the NEXYS 4 DDR board are skipped by the NI module. Also, the LabVIEW program that is storing data in the PXIe module operates in a real time system that has a speed limit of 1 MHz. All these parameters can cause systematic errors in the measurements by the current setup. To demonstrate effects of these systematic errors in the measurements, a specific displacement measurement at the maximum speed of 11 mm·s^{-1} and the corresponding deviations between the HP and polydyne measurements in this experiment are shown in Figure 6-13 and Figure 6-14, respectively.

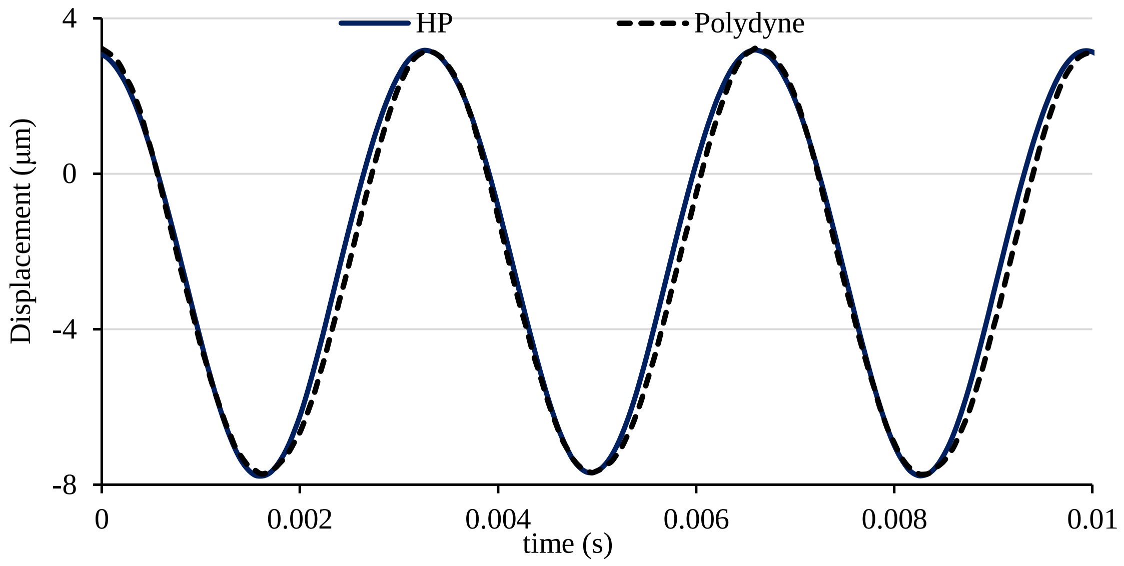


Figure 6-13- Displacement measurement at maximum speed of 11 mm·s^{-1}.

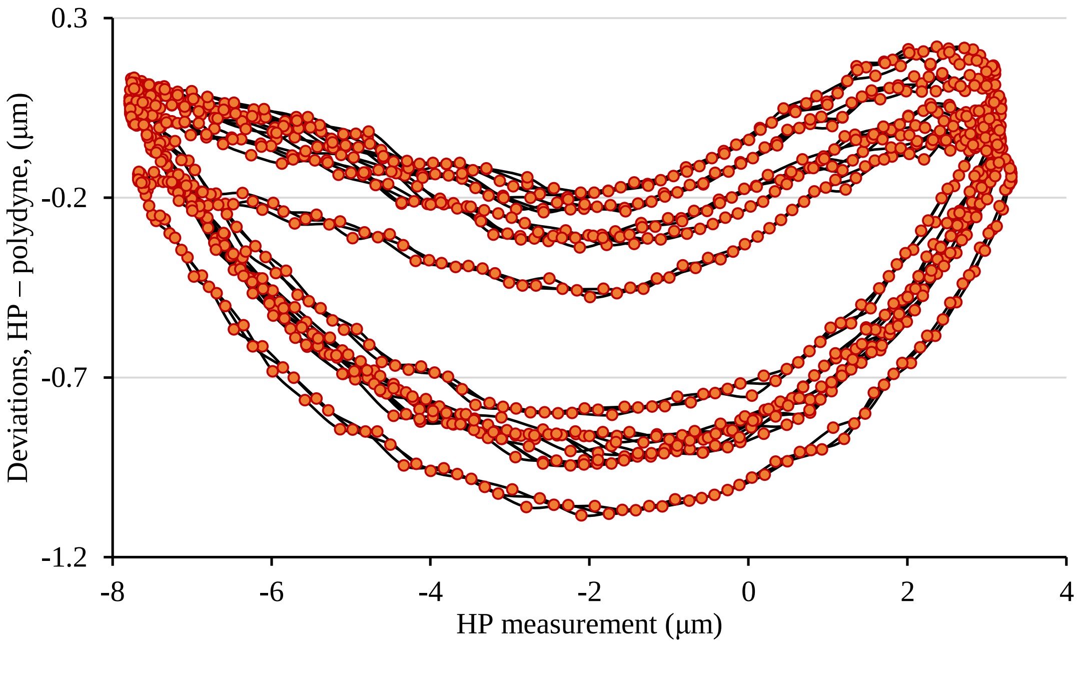


Figure 6-14- Deviations between HP and polydyne measurements shown in Figure 6-13.

Measurements presented in Figure 6-13 and Figure 6-14 show deviations of about -1.2 µm to 0.3 µm between HP and polydyne interferometers. These figures demonstrate not only that the polydyne setup is not synchronous to the HP measurement but also the time intervals that they capture the data do not have the same repeatability. To further demonstrate the existence of systematic error in this measurement, theoretical sine equations were fit to each measurement using Excel solver for minimizing the least square method. These fit equations were plotted versus each other, see Figure 6-15. As seen in this figure, there is a phase difference between these two fit sine waves. Since this phase difference is smaller than the resolution of the captured data, it results in a systematic error when comparing HP and polydyne measurements. Furthermore, the amplitudes of the fit sine waves has about 20 nm difference that is due to the measurement resolution in both HP and polydyne systems. For the results presented in Table 6-2, all these systematic errors have been compensated and the best measurements at highest stable conditions have been used.

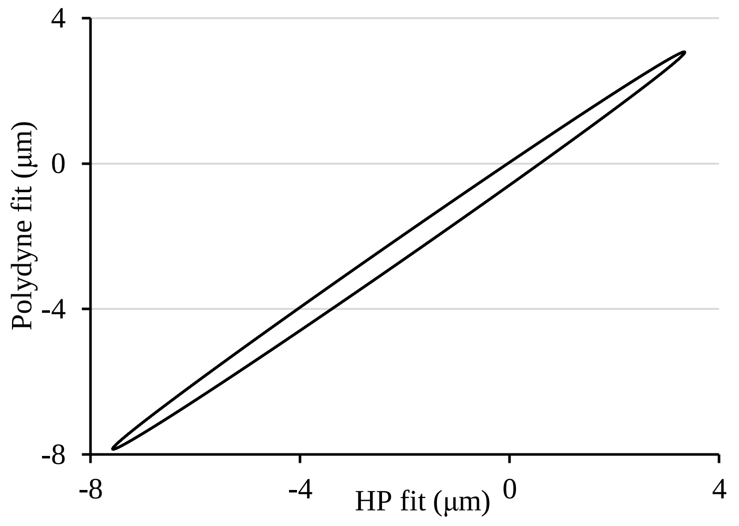


Figure 6-15- Lissajous generated by depicting fit sine waves to displacement measurements by HP and polydyne interferometers.

6.4 Future work

According to the issues explained in this section, here are a few suggestions for future work to both improve the repeatability and accuracy of the measurements by the polydyne interferometer, and to investigate different applications of the polydyning technique:

1. Replace the NEXYS 4 DDR board with another evaluation board that has larger FPGA microchip to implement the phase unwrapping in the same FPGA board. This eliminates the need for NI PXIe module resulting in elimination of synchronization issues between the NI module and the atomic clock as well as increasing the speed and bandwidth of the measurement if a loock-up table approach is adopted for the new phase extraction method. NEXYS Video evaluation board [94] looks promising for this application, since it uses an FPGA microchip that provide digital signal processing units and other resources as twice as the one that is used in NEXYS 4 DDR board.
2. To miniaturize the system into a portable device operating based on a battery power, the Acousto-Optic Modulator should be replaced with another modulation technique. Optical phase modulation techniques like changing the optical path length of a laser beam in its traverse direction by applying oscillations into a quartz crystal seems to be a reasonably feasible and promising approach to solve this issue. Furthermore, the He-Ne laser diode and the PRS 10 atomic clock can be respectively replaced by a VBG-stabilized diode laser and a miniature atomic clock.
3. Design and manufacture of a photodetector circuit board with 4 MHz band-width followed by an on-board FPGA unit will be the best approach to improve the

accuracy of the system along with reducing its size substantially. If this approach is adopted, the signal generation and processing units can be manufactured in a compact size to fit in a portable box.

4. Applied Fourier transform on the measured signal to extract the phase and amplitude of the harmonics showed that the optical phase can be extracted from the sine and cosine terms of each harmonics as well. This reveals the existence of small non-linear terms in the FM light beam. A small perturbation analysis can provide more detailed insight and more relevant equations for the result of interference of each of the diffraction side beams.
5. In the current interferometer set up, only one of the diffraction side beams is extracted via a mechanical slit for constructing polydyne interferometer. Since all the diffraction side beams contain the harmonics of modulation signal, the diffraction side beams that are currently being blocked by the slit can be used to implement a wide variety of interesting experiments just like multi-axis displacement measurement by a single AOM and multiple photodetectors. Also, multiple AOMs with different modulation frequencies can construct a multi-axis displacement measurement interferometer with a single photodetector.
6. Primary measurements for a two-axis displacement measurement polydyne interferometer with a single photodetector [95] proved the existence of modulation harmonics of both axes in the measured signals spectra. However, simultaneous displacement measurement of both axes did not result in accurate measurements. This was most probably due to the high amount of noise that was injected from the moving arm of each of these axes to the other one as well as the interaction of two FM signal

generators through long cables. Implementing a more elaborate interferometry stage that can guide the beams from two AOMs properly can provide a better insight to this experiment.

7. The polarization effects were neglected through all these experiments by using a linearly polarized He-Ne laser beam and non-polarization optics. However, the AOM and the beam splitters has a small polarization non-linearity that can affect the accuracy of the measurements. A comprehensive study on the polarization effects in the polydyne system can provide further insight to the extent of these effects and finally these nonlinearities can be compensated just like the study conducted on heterodyne interferometers in Ref. [96].

REFERENCES

[1] W. F. Magie, *A Source Book in Physics*, Harvard University Press, 309 (1935).

[2] A. A. Michelson, and E. W. Morley, *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 24, 5, 449 (1887), DOI: 10.1080/14786448708628130.

[3] D. W. Ball, *Field Guide to Spectroscopy,* SPIE Press, Bellingham, Washington, USA, 2006, ISBN: 9780819463524.

[4] J. D. Ellis, *Field Guide to Displacement Measuring Interferometry,* SPIE Press, Bellingham, Washington, USA, 2014, ISBN: 9780819497994.

[5] R. Jones, *Proc. Soc. Photo-Opt. Instrum. Eng.* 599, 309 (1985), DOI: 10.1117/12.952392.

[6] D. Chang, G. Cardell, A. San Martin, and G. Spiers, *NASA Tech. Briefs* 29, 9 (2005).

[7] C. Laskoskie, H. Hung, T. El-Wailly, and C. L. Chang, *J. Lightwave Technol.* **7**, 600 (1989), DOI: 10.1109/50.19085.

[8] D. Malacara, *Twyman–Green Interferometer, Optical Shop Testing, 3rd ed.*, John Wiley & Sons, Inc., 46 (2007).

[9] C. R. Steinmetz, *Performance evaluation of laser displacement interferometry on a precision coordinate measuring machine*, Industrial Metrology 1 (1990) 165-191.

[10] N. Bobroff, *Meas. Sci. Technol.* 4, 907 (1993), DOI: 10.1088/0957-0233/4/9/001.

[11] M. Keefer, R. Pinto, C. Dennison, and J. Turlo, *Handbook of thin film deposition processes and techniques, 2nd edition, chapter 6: the role of metrology and inspection in semiconductor processing*, 241 (2001).

[12] P. Spanò, Free-forms Optics into Astronomical Use: The Case of an All-mirror Anamorphic Collimator, *Proceedings of the SPI*, 7018 (2008), DOI: 10.1117/12.789183.

[13] P. Shore, C. Cunningham, D. DeBra, C. Evans, J. Hough, R. Gilmozzi, H. Kunzmann, P. Morantz, and X. Tonnellier, *CIRP Annals - Manufacturing Technology* 59, 694 (2010), DOI: 10.1016/j.cirp.2010.05.003.

[14] X. Liu, I. I. Iordachita, X. He, R. H. Taylor, and J. U. Kang, *Biomed. Opt. Express* 3, 1062 (2012), DOI: 10.1364/BOE.3.001062.

[15] L. J. Wuerz, R.C. Quenelle, *Precision Engineering* 5, 3, 111 (1983), DOI: 10.1016/0141-6359(83)90006-5.

[16] E. R. Peck and S.W. Obetz, *J. Opt. Soc. Am.* 43, 6, 505 (1953), DOI: 10.1364/JOSA.43.000505.

[17] J. N. Dukes and G. B. Gordon, *Hewlett Packard Journal* 21, 12, 2 (1970).

[18] C. Sternkopf, C. Diethold, U. Gerhardt, J.Wurmus, and E. Manske, *Meas. Sci. Technol.* 23, 074006 (2012), DOI: 10.1088/0957-0233/23/7/074006.

[19] S. J. Bennett and W. R. C. Rowley, *J. Phys. E: Sci. Instrum.* 6, 10, 963 (1973), DOI: 10.1088/0022-3735/6/10/002.

[20] M. A. Zumberge, J. Berger, M. A. Dzieciuch, and R. L. Parker, *Appl. Opt.* 43, 771 (2004), DOI: 10.1364/AO.43.000771.

[21] M. J. Downs and K. Raine, *Precision Engineering* 1, 85 (1979), DOI: 10.1016/0141-6359(79)90138-7.

[22] SIOS Meßtechnik GmbH, MI-Series Miniature Retroreflector Interferometer, https://www.piezosystem.com/fileadmin/datasheets/Interferometer-SIOS/MI-Miniature-Retroreflector-Interferometer-SIOS_Datasheet.pdf, visited on April 29, 2019.

[23] E. Leirset, H. E. Engan, and A. Aksnes, *Opt. Express* 21, 019900 (2013), DOI: 10.1364/OE.21.019900.

[24] K. Thurner, P. F. Braun, and K. Karrai, *Rev. Sci. Instrum.* 84, 095005 (2013), DOI: 10.1063/1.4821623.

[25] G. E. Sommagren, *US Patent* 4684828, 1987.

[26] B. K. Nowakowski, D. T. Smith, and S. T. Smith, *Rev. Sci. Instrum.* 87, 115102 (2016), DOI: 10.1063/1.4964622.

[27] B. K. Nowakowski, D. T. Smith, and S. T. Smith, in *Proceedings of the 29th ASPE Annual Meeting*, (2014).

[28] M. Teran, V. Mart´ın, L. Gesa, I. Mateos, F. Gibert, N. Karnesis, J. Ramos- Castro, T. Schwarze, O. Gerberding, G. Heinzel et al., *J. Phys.: Conf. Ser.* 610, 012042 (2015), DOI: 10.1088/1742-6596/610/1/012042.

[29] K. S. Isleif, O. Gerberding, T. S. Schwarze, M. Mehmet, G. Heinzel, and F. G. Cervantes, *Opt. Express* 24, 1676 (2016), DOI: 10.1364/OE.24.001676.

[30] O. Gerberding, *Opt. Express* 23, 14753 (2015), DOI: 10.1364/OE.23.014753.

[31] J. E. Greivenkamp and J. H. Bruning, Phase Shifting Interferometry, in *Optical Shop Testing*, D. Malacara, ed. (Whiley, New York, 1992), Chap. 14.

[32] P. Hariharan, *Optical Interferometry* (Academic, Orlando, Fla., 1985).

[33] P. de Groot, *Applied Optics* 34, 22, 4723 (1995), DOI: 10.1364/AO.34.004723.

[34] J. Watchi, S. Cooper, B. Ding, C. M. Mow-Lowry,and C. Collette1, *Rev. Sci. Instrum.* 89, 121501 (2018), DOI: 10.1063/1.5052042.

[35] I. Y. Poberezhskiy, B. Bortnik, J. Chou, B. Jalali, and H. R. Fetterman, *IEEE J. Quantum Electron* 41, 1533 (2005), DOI: 10.1109/JQE.2005.858467.

[36] A. Dandridge, and A. B. Tveton, *Electron. Lett.* 18, 302 (1982), DOI: 10.1049/el:19820206.

[37] S. Saito, Y. Yamamoto, and T. Kimura, *Electron. Lett.* 16, 826 (1980), DOI: 10.1049/el:19800587.

[38] C. T. Anger, *Neueste Schriften der Naturf. Ges. in Danzig* 5, 2 (1855).

[39] C. G. J. Jacobi, *Journal für die reineund angewandte Mathematik* 16, 344 (1836).

[40] J. Zheng, *Applied Optics* 43, 4189 (2004), DOI: 10.1364/AO.43.004189.

[41] G. Heinzel, F. Guzm´an Cervantes, A. F. Garc´ıa Marin, J. Kullmann, W. Feng, and K. Danzmann, *Opt. Express* 18, 19076 (2010), DOI: 10.1364/OE.18.019076.

[42] M. Abramowitz, I. A. Stegun et al., *Handbook of mathematical functions*, vol. 1 Dover New York, 1972.

[43] S. Haykin, *Communication systems*, 4th Ed., John Willey & Sons, INC., New York, 2011, Ch. 2, pp. 109–130.

[44] S. T. Cundiff, Y. Jun, and J. L. Hall, *Rev. Sci. Instrum.* 72, 3749 (2001), DOI: 10.1063/1.1400144.

[45] P. de Groot, H. A. Hill, *US patent* 5838485 A, Nov. 17, 1998.

[46] K. Thurner, P. F. Braun, and K. Karrai, *Rev. Sci. Instrum*. 84, 115002 (2013), DOI: 10.1063/1.4831800.

[47] I. Hatai, and I. Chakrabarti, *International Journal of Reconfigurable Computing* 342532, 10 pages (2011), DOI: 10.1155/2011/342532.

[48] AD9951 datasheet, Analog Devices, http://www.analog.com/media/en/technical-documentation/data-sheets/AD9951.pdf, visited on Dec 2, 2018.

[49] AD9912 datasheet, Analog Device, http://www.analog.com/static/imported-files/data_sheets/AD9912.pdf, visited on Dec 2, 2018.

[50] AD9834 datasheet, Analog Device, http://www.analog.com/media/en/technical-documentation/data-sheets/AD9834.pdf, visited on Dec 2, 2018.

[51] A. Gutierrez-Aitken, J. Matsui, E. N. Kaneshiro, B. K. Oyama, A. K. Oki, and D. C. Streit, *IEEE J. Solid-State Circuits* 37, 1115 (2002), DOI: 10.1109/JSSC.2002.801174.

[52] S. E. Turner, and D. E. Kotecki, *IEEE Microw. Wireless Compon. Lett.*16, 296 (2006), DOI: 10.1109/LMWC.2006.873490.

[53] S. E. Turner, and D. E. Kotecki, *IEEE J. Solid-State Circuits* 41, 2284 (2006), DOI: 10.1109/JSSC.2006.881552.

[54] S. E. Turner, R. T. Chan, and J. T. Feng, *IEEE Microw. Wireless Compon. Lett.*18, 566 (2008), DOI: 10.1109/LMWC.2008.2001025.

[55] X. Yu, F. F. Dai, J. D. Irwin, and R. C. Jaeger, *IEEE Trans. Microw. Theory Tech.* 56, 1257 (2008), DOI: 10.1109/TMTT.2008.921308.

[56] X. Yu, F. F. Dai, J. D. Irwin, and R. C. Jaeger, *IEEE J. Solid-State Circuits* 43, 1384 (2008), DOI: 10.1109/JSSC.2008.922739.

[57] S. Thuries, E. Tournier, A. Cathelin, S. Godet, and J. Graffeuil, *IEEE Microw. Wireless Compon. Lett.* 18, 1 (2008), DOI: 10.1109/LMWC.2007.911994.

[58] X. Geng, X. Yu, F. F. Dai, J. D. Irwin, and R. C. Jaeger, *IEEE Journal of Solid-State Circuits* 45, 300 (2010), DOI: 10.1109/JSSC.2009.2037542.

[59] X. Geng, F. F. Dai, J. D. Irwin, and R. C. Jaeger, *IEEE Journal of Solid-State Circuits* 45, 944 (2010), DOI: 10.1109/JSSC.2010.2041398.

[60] R. Thirunarayanan, D. Ruffieux, N. Scolari, and C. Enz, *European Solid-State Circuits Conference (ESSCIRC)*, 388 (2015), DOI: 10.1109/ESSCIRC.2015.7313909.

[61] PRS10 Rubidium Frequency Standard catalog and manual, Stanford Research Systems, https://www.thinksrs.com/downloads/pdfs/catalog/PRS10c.pdf, and https://www.thinksrs.com/downloads/pdfs/manuals/PRS10m.pdf, visited on Dec 2, 2018.

[62] R. E. Ziemer, and W. H. Tranter, *Principles of communications, systems, modulation, and noise*, 4th Ed., Wiley, 1995.

[63] LF353 Datasheet, Texas Instruments, http://www.ti.com/lit/ds/symlink/lf353-n.pdf, visited on Dec 2, 2018.

[64] LT1016 datasheet, Analog Devices, https://www.analog.com/media/en/technical-documentation/data-sheets/lt1016.pdf , visited on Dec 2, 2018.

[65] SN74LS123 Datasheet, Texas Instruments, https://datasheet.octopart.com/SN74LS123N-Texas-Instruments-datasheet-8442807.pdf, visited on Dec 2, 2018.

[66] AD5270 datasheet, Analog Devices, https://www.analog.com/media/en/technical-documentation/data-sheets/ad5270_5271.pdf, visited on Dec 2, 2018.

[67] MCP2518S voltage regulator datasheet, http://ww1.microchip.com/downloads/en/DeviceDoc/22056b.pdf, visited on Feb 24, 2019.

[68] J. D. Hagerty, An advanced direct-digital VFO, http://www.wa1ffl.com/Hagerty_QEX.pdf, 2008.

[69] American Radio Relay League, The Radio Amateur's Handbook (American Radio Relay League, USA, 1985), through 1994 editions, pp. 30-8–30-15.

[70] AD835 datasheet, Analog Devices, https://www.analog.com/media/en/technical-documentation/data-sheets/ad835.pdf, visited on Feb 24, 2019.

[71] L. Brillouin, *Annales de Physique* 17, 88 (1922), DOI: 10.1051/anphys/192209170088.

[72] P. Debye, and F. W. Sears, *PNAS* 18, 409 (1932), DOI: 10.1073/pnas.18.6.409.

[73] R. Lucas, and P. Biquard, *Journal de Physique* 71, 464 (1932).

[74] M. Born, and E. Wolf, *Principles of Optics, Electromagnetic theory of propagation interference and diffraction of light*, Sixth Edition, Cambridge University Press, Ch. 11, pp. 594, 1980.

[75] C. V. Raman, and N. S. N. Nath, *Parts I, II, and II in Proc. Ind. Acad. Sci.* 2, 406, 413 (1935), 3, 75 (1936), DOI: 10.1007/BF03035840, DOI: 10.1007/BF03035841, and DOI: 10.1007/BF03046238.

[76] E. David, *Phys. Z.* 38, 587 (1937).

[77] A. B. Bhatia, *Ultrasonic Absorption, An Introduction to the Theory of Sound Absorption and Dispersion in Gases, Liquids, and Solids*, Dover Publications Inc., New York, 1967.

[78] V. Vivek, Optical switches, https://wp.optics.arizona.edu/milster/wp-content/uploads/sites/48/2016/06/acousto-optics-modulator.pdf, visited on March 4, 2019.

[79] A. Yaris, P. Yeh, *Optical Waves in Crystals, Propagation and Control of Laser Radiation*, John Wiley & Sons, Inc., New York, Ch. 9 and 10, pp. 318-404, 1984.

[80] IntraAction AOM datasheet, https://intraaction.com/wp-content/themes/Divi/pdf/AOM-AF1_Series_04053.pdf, visited on March 5, 2019.

[81] Analog Devices, Circuit Note CN-0272, https://www.analog.com/en/design-center/reference-designs/hardware-reference-design/circuits-from-the-lab/cn0272.html#rd-overview, visited on Dec 2, 2018.

[82] G. Lochead, Photodiode amplifiers, application handbook, McGraw-Hill, second edition, 1996.

[83] Stanford Research Systems, Model SR810 DSP Lock-In Amplifier, https://www.thinksrs.com/downloads/pdfs/manuals/SR850m.pdf, visited on March 27, 2019.

[84] Linear Technologies, LT1793 datasheet, https://www.analog.com/media/en/technical-documentation/data-sheets/1793f.pdf, visited on March 27, 2019.

[85] Linear Technologies, LT1028 datasheet, https://www.analog.com/media/en/technical-documentation/data-sheets/1028fd.pdf, visited on March 27, 2019.

[86] OSRAM Opto Semiconductors, BPX61 datasheet, https://www.mouser.com/ds/2/311/BPX%2061,%20Lead%20(Pb)%20Free%20Product%20-%20RoHS%20Compliant-318809.pdf, visited on March 27, 2019.

[87] Analog Devices, AD630 Balanced Modulator/Demodulator, https://www.analog.com/media/en/technical-documentation/data-sheets/AD630.pdf, visited on March 27, 2019.

[88] R. W. Hamming, *Digital Filters*, Dover Publications, Inc., Mineola, New York, 3rd edition, Ch. 10, pp. 208-218, 1989.

[89] NEXYS4 DDR reference manual, Digilent National Instruments Company, https://reference.digilentinc.com/reference/programmable-logic/nexys-4-ddr/reference-manual, visited on Dec 2, 2018.

[90] D. E. Newland, *An Introduction To Random Vibrations, Spectral & Wavelet Analysis*, John Wiley & Sons, Inc., New York, 3rd Edition, Ch. 10, pp. 113-124, 1993.

[91] Xilinx Vivado software, https://www.xilinx.com/support/download.html, visited on March 31, 2019.

[92] Hewlett Packard model 5529A interferometer, https://www.keysight.com/en/pd-1000001491%3Aepsg%3Apro-pn-5529A/dynamic-calibration-system?nid=-33213.536880403&cc=TR&lc=eng, visited on March 31, 2019.

[93] P. E. Ciddor, Applied Optics 35, 1566 (1996), DOI: 10.1364/AO.35.001566.

[94] NEXYS Video reference manual, Digilent National Instruments Company, https://reference.digilentinc.com/_media/nexys-video:nexys_video_rm.pdf, visited on Dec 2, 2018.

[95] L. D. Perez, K. Arumugam, M. Arablu, and S. T. Smith, Combined frequency-modulated signals for two-axis displacement measurement, *euspen's 19th International Conference & Exhibition*, Bilbao, ES, June 2019.

[96] V. G. Badami, *Investigation and compensation of periodic nonlinearities in heterodyne interferometry*, PhD dissertation, The University of North Carolina at Charlotte, 1999.

APPENDIX A. ARDUINO CODE FOR FM BOARD

The Arduino code used to program the FM board is presented here. This code is the modified version of the simple separate codes used for each microchip. It is developed in a library format for Arduino. Users should copy the codes “FMGenerate.CPP”, “FMGenerate.h” and “keywords” in separate “.txt” formatted files and keep the title of the files as “FMGenerate”, “FMGenerate”, and “keywords”. Then, these files should be copied to a folder with title “FMGenerate” inside the library directory of the Arduino. Finally, the code “FMGenerate_Arduino” can be executed in an Arduino software to program the DDS chips and the rheostat of the FM board. The code contains comments to guide the users to choose proper Arduino digital pins to be connected to the digital pins of the board.

Appendix A.1 FMGenerate_Arduino

```
/*
This code was created on May 2017 for one FM board and expanded on Sept 2018 to program two FM
boards and a Saji board for multi-axis polydyne interferometry.

Created by Masoud Arablu.

The FMGenerate.h includes the library and the values should be sent via a function like
fmGenerate(long clk_,double PLLM_,long fm_1,long fc_1,double RPot_1,int ss3_1,int ss2_1,int IOU_1,int
OSK_1,int SDIO_1,int SCLK_1,int ss1_1,int IOSync_1,int RESET_1,int PDCtl_1,...
        long fm_2,long fc_2,double RPot_2,int ss3_2,int ss2_2,int IOU_2,int OSK_2,int SDIO_2,int
SCLK_2,int ss1_2,int IOSync_2,int RESET_2,int PDCtl_2,...
        long dfm,int SCLK_Saji,int DATA_Saji,int SS_Saji)

This is a function that programms two FM boards and a Saji board separately for a two-axis polydyne
interferometer.
Digital ground of Arduino should be connected to one of the GND digital pins on the FM boards.
```

```
The parameters ending to _1 are for FM board one, _2 for FM board 2, and _ are for both FM boards.
*/

#include <FMGenerate.h>

long clk_, fm_1, fm_2, fc_1, fc_2, dfm;
double PLLM_, RPot_1, RPot_2;
int ss2_1,ss3_1,IOU_1,OSK_1,SDIO_1,SCLK_1,ss1_1,IOSync_1,RESET_1,PDCtl_1;
int ss2_2,ss3_2,IOU_2,OSK_2,SDIO_2,SCLK_2,ss1_2,IOSync_2,RESET_2,PDCtl_2;
int SCLK_Saji,DATA_Saji,SS_Saji;

void setup()
  {
    clk_     = 10000000; // External Clock frequency. 10 MHz for the PRS10 atomic clock.
    PLLM_    = 20;       // An integer between 4 and 20 to be multiplied in the multiplexer of the PLL in
DDS2.
    // Set Values for the FM board 1
    RPot_1   = 20000;    // This is the resistance value of the rheostat of the FM board 1. Can vary from 0 to
20000.
    fm_1     = 312500;   // Modulation frequency of the FM board 1.
    fc_1     = 60000000; // Carier frequency of the FM board 1.
    PDCtl_1  = 12;       // Arduino digital pin number that is connected to the PwrDwnCtl pin of the FM
board 1.
    RESET_1  = 11;       // Arduino digital pin number that is connected to the Reset pin of the FM board 1.
    IOSync_1 = 10;       // Arduino digital pin number that is connected to the IOSync pin of the FM board 1.
    ss1_1    = 9;        // Arduino digital pin number that is connected to the SS1 pin of the FM board 1.
    SCLK_1   = 8;        // Arduino digital pin number that is connected to the SCLK pin of the FM board 1.
    SDIO_1   = 7;        // Arduino digital pin number that is connected to the SDIO pin of the FM board 1.
    OSK_1    = 6;        // Arduino digital pin number that is connected to the OSK pin of the FM board 1.
    IOU_1    = 5;        // Arduino digital pin number that is connected to the IOU pin of the FM board 1.
    ss2_1    = 4;        // Arduino digital pin number that is connected to the SS2 pin of the FM board 1.
    ss3_1    = 3;        // Arduino digital pin number that is connected to the SS3 pin of the FM board 1.

    // Set Values for the FM board 2
    RPot_2    = RPot_1;  // This is the resistance value of the rheostat of the FM board 2. Can vary from 0 to
20000.
    fm_2      = fm_1;    // Modulation frequency of the FM board 2.
    fc_2      = fc_1;    // Carier frequency of the FM board 2.
    PDCtl_2   = 25;      // Arduino digital pin number that is connected to the PwrDwnCtl pin of the FM
board 2.
    RESET_2   = 27;      // Arduino digital pin number that is connected to the Reset pin of the FM board 2.
    IOSync_2  = 29;      // Arduino digital pin number that is connected to the IOSync pin of the FM board 2.
    ss1_2     = 31;      // Arduino digital pin number that is connected to the SS1 pin of the FM board 2.
    SCLK_2    = 33;      // Arduino digital pin number that is connected to the SCLK pin of the FM board 2.
    SDIO_2    = 35;      // Arduino digital pin number that is connected to the SDIO pin of the FM board 2.
    OSK_2     = 37;      // Arduino digital pin number that is connected to the OSK pin of the FM board 2.
    IOU_2     = 39;      // Arduino digital pin number that is connected to the IOU pin of the FM board 2.
    ss2_2     = 41;      // Arduino digital pin number that is connected to the SS2 pin of the FM board 2.
    ss3_2     = 43;      // Arduino digital pin number that is connected to the SS3 pin of the FM board 2.

    // Set Values for Saji board
    dfm       = (fm_1-fm_2); // Frequency that should be generated by Saji board for beat detection.
    SCLK_Saji = 44;      // Arduino digital pin number that is connected to the CLK pin of the Saji board.
    DATA_Saji = 45;      // Arduino digital pin number that is connected to the DATA pin of the Saji board.
    SS_Saji   = 46;      // Arduino digital pin number that is connected to the SS pin of the Saji board.
  FMGenerate
fmGenerate(clk_,PLLM_,fm_1,fc_1,RPot_1,ss3_1,ss2_1,IOU_1,OSK_1,SDIO_1,SCLK_1,ss1_1,IOSync_
1,RESET_1,PDCtl_1,fm_2,fc_2,RPot_2,ss3_2,ss2_2,IOU_2,OSK_2,SDIO_2,SCLK_2,ss1_2,IOSync_2,R
ESET_2,PDCtl_2,dfm,SCLK_Saji,DATA_Saji,SS_Saji);
  }

void loop()
{
}
```

Appendix A.2 keywords

```
FMGenerate    KEYWORD1
Transfer   KEYWORD2
```

Appendix A.3 FM_Generate.h

```
// This is the Header file.

#ifndef FMGenerate_h
#define FMGenerate_h

class FMGenerate
{
 public:
    FMGenerate(long clk_,double PLLM_,long fm_1,long fc_1,double RPot_1,int ss3_1,int ss2_1,int
IOU_1,int OSK_1,int SDIO_1,int SCLK_1,int ss1_1,int IOSync_1,int RESET_1,int PDCtl_1,long
fm_2,long fc_2,double RPot_2,int ss3_2,int ss2_2,int IOU_2,int OSK_2,int SDIO_2,int SCLK_2,int
ss1_2,int IOSync_2,int RESET_2,int PDCtl_2,long dfm,int SCLK_Saji,int DATA_Saji,int SS_Saji);
    void Pot(int ssp,double RPot,int SCLK,int SDIO);
    void Transfer(int ControlWord, long FrequencyReg,int SCLK,int SDIO,int sst);
    void DDS1(long fm,int SCLK,int SDIO,int ss2,long clk);
    void DDS2(long fc,int SCLK,int SDIO,double PLLM,int ss1,int PDCtl,int RESET,int IOSync,int
OSK,int IOU,long clk);
 private:
    int i;
};

#endif
```

Appendix A.4 FM_Generate.CPP

```
// This is the main CPP file.

#include "Arduino.h"
#include "FMGenerate.h"
#include <math.h>

FMGenerate::FMGenerate(long clk_,double PLLM_,long fm_1,long fc_1,double RPot_1,int ss3_1,int
ss2_1,int IOU_1,int OSK_1,int SDIO_1,int SCLK_1,int ss1_1,int IOSync_1,int RESET_1,int
PDCtl_1,long fm_2,long fc_2,double RPot_2,int ss3_2,int ss2_2,int IOU_2,int OSK_2,int SDIO_2,int
SCLK_2,int ss1_2,int IOSync_2,int RESET_2,int PDCtl_2,long dfm,int SCLK_Saji,int DATA_Saji,int
SS_Saji){
   // For FM board 1:
   Pot(ss3_1,RPot_1,SCLK_1,SDIO_1);                    //calling Pot  (AD5270) function
      DDS1(fm_1,SCLK_1,SDIO_1,ss2_1,clk_);                    //calling DDS1 (AD9883) function for
cos(2.pi.fm1.t)
   DDS2(fc_1,SCLK_1,SDIO_1,PLLM_,ss1_1,PDCtl_1,RESET_1,IOSync_1,OSK_1,IOU_1,clk_);
   //calling DDS2 (AD9951) function for cos(2.pi.fc1.t)

   // For FM board 2:
   Pot(ss3_2,RPot_2,SCLK_2,SDIO_2);                      //calling Pot  (AD5270) function
      DDS1(fm_2,SCLK_2,SDIO_2,ss2_2,clk_);                    //calling DDS1 (AD9883) function for
cos(2.pi.fm2.t)
   DDS2(fc_2,SCLK_2,SDIO_2,PLLM_,ss1_2,PDCtl_2,RESET_2,IOSync_2,OSK_2,IOU_2,clk_);
   //calling DDS2 (AD9951) function for cos(2.pi.fc2.t)

   // For Saji board:
   DDS1(dfm,SCLK_Saji,DATA_Saji,SS_Saji,clk_);                //calling DDS1 (AD9883) function for
cos(2.pi.dfm.t)
} // end of FMGenerate configuration

// defining Pot function here
void FMGenerate::Pot(int ssp,double RPot,int SCLK,int SDIO){
   int DIN[16] = {0,0,0,1,1,1,0,0,0,0,0,0,0,0,1,0};
   unsigned long y1, z1;
   pinMode(ssp, OUTPUT);
   digitalWrite(ssp, 1);
   pinMode(SCLK, OUTPUT);
   digitalWrite(SCLK, 0);
   pinMode(SDIO, OUTPUT);
   digitalWrite(SDIO, 0);
   digitalWrite(ssp, 0);// Enable the chip by taking ssp to logic LOW
   delayMicroseconds(1);
   // WRITING DIN
   for (i = 0; i < 16; i++) {
      digitalWrite(SCLK, 1);
      digitalWrite(SDIO, DIN[i]);
      digitalWrite(SCLK, 0);
   }
   digitalWrite(ssp, 1); // Disabling the chip by taking ssp to logic HIGH.
   delayMicroseconds(1);
   DIN[3] = 0;
   DIN[4] = 0;
   //Calculating correct ratio of potentiometer and bit-reading it to DIN[15:6]
```

```
    z1 = RPot*1023/20000;
    y1 = (unsigned long)ceil(z1);
    for (i = 0; i < 10; i++) {
        DIN[15-i] = bitRead(y1, i);
    }
    digitalWrite(ssp, 0); // Enable the chip by taking ssp to logic LOW
    delayMicroseconds(1);
    // WRITING DIN
    for (i = 0; i < 16; i++) {
        digitalWrite(SCLK, 1);
        digitalWrite(SDIO, DIN[i]);
        digitalWrite(SCLK, 0);
    }
    digitalWrite(ssp, 1); // Disabling the chip by taking ssp to logic HIGH.
    delayMicroseconds(1);
} // end of Pot function

// defining DDS1 function here
void FMGenerate::DDS1(long fm,int SCLK,int SDIO,int ss2,long clk){
    int cw;
    char _type = 's';
    //fm = fm*(268435456/(10*clk)); // Value which goes to the Freq Register
        fm = long(fm*26.8435456);
    pinMode(SDIO,OUTPUT);
    pinMode(SCLK,OUTPUT);
    pinMode(ss2,OUTPUT);
    switch (_type){
        case 's': // s for Sine wave
            cw = 0;
        break;
        case 't': // t for Triangle
            cw = 16384;
        break;
        case 'p': // p for Sawtooth
            cw = 5120;
        break;
    } //end of switch
    Transfer(cw,fm,SCLK,SDIO,ss2);
    digitalWrite(ss2,1);
    Serial.println();
    delay(100);
} // End of DDS1 function

// defining Transfer function here
void FMGenerate::Transfer(int ControlWord, long FrequencyReg,int SCLK,int SDIO,int sst){
  // writing Control Word:
  digitalWrite(SCLK,0);
  digitalWrite(sst,1);
  digitalWrite(SCLK,1);
  delayMicroseconds(1);
  digitalWrite(sst,0);
  for (byte i=0; i<16; i++) {
    byte cwState = bitRead(ControlWord, i);
    digitalWrite(SDIO,cwState);
    delayMicroseconds(1);
    digitalWrite(SCLK,0);
    delayMicroseconds(1);
    digitalWrite(SCLK,1);
  }
  //writing Fine Tune:
  digitalWrite(SCLK,0);
  digitalWrite(sst,1);
  digitalWrite(SCLK,1);
  delayMicroseconds(1);
  digitalWrite(sst,0);
  // writing 01 to fm:
  digitalWrite(SDIO,0);
  delayMicroseconds(1);
  digitalWrite(SCLK,0);
  delayMicroseconds(1);
  digitalWrite(SCLK,1);
  digitalWrite(SDIO,1);
  delayMicroseconds(1);
  digitalWrite(SCLK,0);
  delayMicroseconds(1);
  digitalWrite(SCLK,1);
  // Fine Tune to fm:
  for (byte i=0; i<14; i++) {
    byte ftState = bitRead(FrequencyReg, 13-i);
    digitalWrite(SDIO,ftState);
    delayMicroseconds(1);
    digitalWrite(SCLK,0);
    delayMicroseconds(1);
    digitalWrite(SCLK,1);
  }
  // writing Control Word:
  digitalWrite(SCLK,0);
  digitalWrite(sst,1);
  digitalWrite(SCLK,1);
  delayMicroseconds(1);
  digitalWrite(sst,0);
  for (byte i=0; i<16; i++) {
    byte cwState2 = bitRead(ControlWord+8, i);
    digitalWrite(SDIO,cwState2);
    delayMicroseconds(1);
    digitalWrite(SCLK,0);
    delayMicroseconds(1);
    digitalWrite(SCLK,1);
  }
  // writing Coarse Tune:
  digitalWrite(SCLK,0);
  digitalWrite(sst,1);
  digitalWrite(SCLK,1);
  delayMicroseconds(1);
  digitalWrite(sst,0);
  // writing 01 to fm:
  digitalWrite(SDIO,0);
  delayMicroseconds(1);
  digitalWrite(SCLK,0);
  delayMicroseconds(1);
  digitalWrite(SCLK,1);
  digitalWrite(SDIO,1);
```

```
  delayMicroseconds(1);
  digitalWrite(SCLK,0);
  delayMicroseconds(1);
  digitalWrite(SCLK,1);
  // Coarse Tune to fm:
  for (byte i=0; i<14; i++) {
    byte state = bitRead(FrequencyReg, 27-i);
    Serial.print(state);
    digitalWrite(SDIO,state);
    delayMicroseconds(1);
    digitalWrite(SCLK,0);
    delayMicroseconds(1);
    digitalWrite(SCLK,1);
  }
}

// defining DDS2 function here
void FMGenerate::DDS2(long fc,int SCLK,int SDIO,double PLLM,int ss1,int PDCtl,int RESET,int
IOSync,int OSK,int IOU,long clk){
    unsigned long y;
    unsigned long z;
    double x;

        // CFR, ASF, ARR, FTW, and POW are the values that should be sent to the registers of the AD9951
to program it properly.
    // These values are available in AD9951 datasheet.

    int CFR1[40] = {0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0,  0,0,1,0,0,0,1,0,  0,0,0,0,0,0,1,0};
    int CFR2[32] = {0,0,0,0,0,0,0,1,  0,0,0,1,1,0,0,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,1,1};
    int ASF[24]  = {0,0,0,0,0,0,1,0,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0};
    int ARR[16]  = {0,0,0,0,0,0,1,1,  0,0,0,0,0,0,0,0};
    int FTW[40]  = {0,0,0,0,0,1,0,0};
    int POW[24]  = {0,0,0,0,0,1,0,1,  0,0,0,0,0,0,0,0,  0,0,0,0,0,0,0,0};

    pinMode(PDCtl, OUTPUT);  // PwrDwnCtrl Pin
    pinMode(RESET, OUTPUT);  // RESET Pin
    pinMode(IOSync, OUTPUT); // IOSync Pin
    pinMode(ss1, OUTPUT);    // Chip Select Pin
    pinMode(SCLK, OUTPUT);   // CLOCK Pin
    pinMode(SDIO, OUTPUT);   // Data Pin
    pinMode(OSK, OUTPUT);    // OSK Pin
    pinMode(IOU, OUTPUT);    // IOUPDATE Pin

    // Initialize the pins to inactive mode
    digitalWrite(PDCtl, 0);
    delayMicroseconds(1);
    digitalWrite(RESET, 0);
    delayMicroseconds(1);
    digitalWrite(IOSync, 0);
    delayMicroseconds(1);
    digitalWrite(ss1, 1);
    delayMicroseconds(1);
    digitalWrite(SCLK, 0);
    delayMicroseconds(1);
    digitalWrite(SDIO, 0);
    delayMicroseconds(1);
```

```
   digitalWrite(OSK, 0);
   delayMicroseconds(1);
   digitalWrite(IOU, 1);
   delayMicroseconds(1);

   // Bitreading PLLM to CFR2<7:3> which is equivalent to CFR2[24:28]
   z = (unsigned long)ceil(PLLM);
   for (i = 0; i < 5; i++){
      CFR2[28-i]= bitRead(z, i);
   }
   // Calculating the frequency tuning word according to the desired frequency and external clock in the
form of a float number x,
   // ceiling it and then bitReading it into FTW[i] for 8<= i <=39
   x = (4294967296.000/PLLM)*(fc/clk);
   if ( x > 2147483648.0 ) {
      x = 4294967296.000*(1-(fc/(PLLM*clk)));
   }
   y= (unsigned long)ceil(x);
      for (i = 0; i < 32; i++) {
   FTW[39-i]=bitRead(y,i);
   }
   //Toggle RESET:
   digitalWrite(RESET, 1);
   delayMicroseconds(1);
   digitalWrite(RESET, 0);
   delayMicroseconds(1);
   // Taking the IOU Low:
   digitalWrite(IOU, 0);
   delayMicroseconds(1);
   //Toggle IOSync:
   digitalWrite(IOSync, 1);
   delayMicroseconds(1);
   digitalWrite(IOSync, 0);
   delayMicroseconds(1);
   // Enable the chip by taking ss1 to logic LOW:
   digitalWrite(ss1, 0);
   delayMicroseconds(1);
   // WRITING CFR1:
   digitalWrite(SCLK, 0);
   for (i = 0; i < 40; i++) {
      delayMicroseconds(1);
      digitalWrite(SDIO, CFR1[i]);
      delayMicroseconds(1);
      digitalWrite(SCLK, 1);
      delayMicroseconds(1);
      digitalWrite(SCLK, 0);
   }
   // Toggle IOU:
   digitalWrite(IOU, 1);
   delayMicroseconds(1);
   digitalWrite(IOU, 0);
   delayMicroseconds(1);
   //Toggle IOSync:
   digitalWrite(IOSync, 1);
   delayMicroseconds(1);
   digitalWrite(IOSync, 0);
```

```
delayMicroseconds(1);
// WRITING CFR2:
for (i = 0; i < 32; i++) {
   delayMicroseconds(1);
   digitalWrite(SDIO, CFR2[i]);
   delayMicroseconds(1);
   digitalWrite(SCLK, 1);
   delayMicroseconds(1);
   digitalWrite(SCLK, 0);
}
// Toggle IOU:
digitalWrite(IOU, 1);
delayMicroseconds(1);
digitalWrite(IOU, 0);
delayMicroseconds(1);
//Toggle IOSync:
digitalWrite(IOSync, 1);
delayMicroseconds(1);
digitalWrite(IOSync, 0);
delayMicroseconds(1);
// WRITING ASF:
for (i = 0; i < 24; i++) {
   delayMicroseconds(1);
   digitalWrite(SDIO, ASF[i]);
   delayMicroseconds(1);
   digitalWrite(SCLK, 1);
   delayMicroseconds(1);
   digitalWrite(SCLK, 0);
}
// Toggle IOU:
digitalWrite(IOU, 1);
delayMicroseconds(1);
digitalWrite(IOU, 0);
delayMicroseconds(1);
//Toggle IOSync:
digitalWrite(IOSync, 1);
delayMicroseconds(1);
digitalWrite(IOSync, 0);
delayMicroseconds(1);
// WRITING ARR:
for (i = 0; i < 16; i++) {
   delayMicroseconds(1);
   digitalWrite(SDIO, ARR[i]);
   delayMicroseconds(1);
   digitalWrite(SCLK, 1);
   delayMicroseconds(1);
   digitalWrite(SCLK, 0);
}
// Toggle IOU:
digitalWrite(IOU, 1);
delayMicroseconds(1);
digitalWrite(IOU, 0);
delayMicroseconds(1);
//Toggle IOSync:
digitalWrite(IOSync, 1);
delayMicroseconds(1);
```

```
    digitalWrite(IOSync, 0);
    delayMicroseconds(1);
    // WRITING FTW:
    for (i = 0; i < 40; i++) {
        delayMicroseconds(1);
        digitalWrite(SDIO, FTW[i]);
        delayMicroseconds(1);
        digitalWrite(SCLK, 1);
        delayMicroseconds(1);
        digitalWrite(SCLK, 0);
    }
    // Toggle IOU:
    digitalWrite(IOU, 1);
    delayMicroseconds(1);
    digitalWrite(IOU, 0);
    delayMicroseconds(1);
    //Toggle IOSync:
    digitalWrite(IOSync, 1);
    delayMicroseconds(1);
    digitalWrite(IOSync, 0);
    delayMicroseconds(1);
    // WRITING POW:
    for (i = 0; i < 24; i++) {
        delayMicroseconds(1);
        digitalWrite(SDIO, POW[i]);
        delayMicroseconds(1);
        digitalWrite(SCLK, 1);
        delayMicroseconds(1);
        digitalWrite(SCLK, 0);
    }
    // Take IOU to logic HIGH:
    delayMicroseconds(1);
    digitalWrite(IOU, 1);
    // Disabling the Chip by taking ss1 HIGH.
    delayMicroseconds(1);
    digitalWrite(ss1, 1);
}// end of DDS2 function.
```

APPENDIX B. SPECTRA OF ELECTRICAL FM SIGNALS

Spectra of signals of $f_c = 10$, 30, 60, and 70 MHz with two different modulation frequency sets of (50, 100, and 150 kHz) and (200, 250, and 300 kHz) are presented:

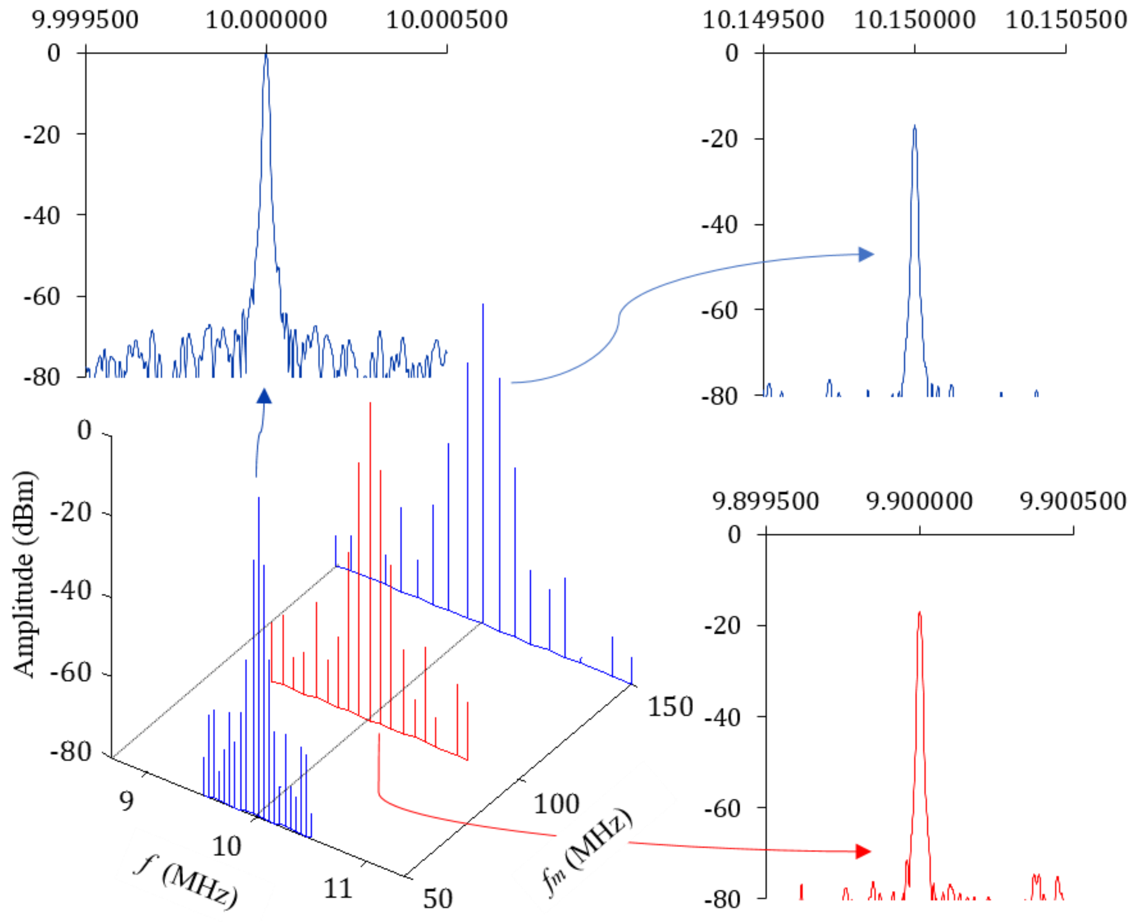


Figure Appendix B-1- Spectra of signals of $f_c = 10$ MHz and $f_m =$ 50, 100, and 150 kHz measured with frequency steps of 1 Hz.

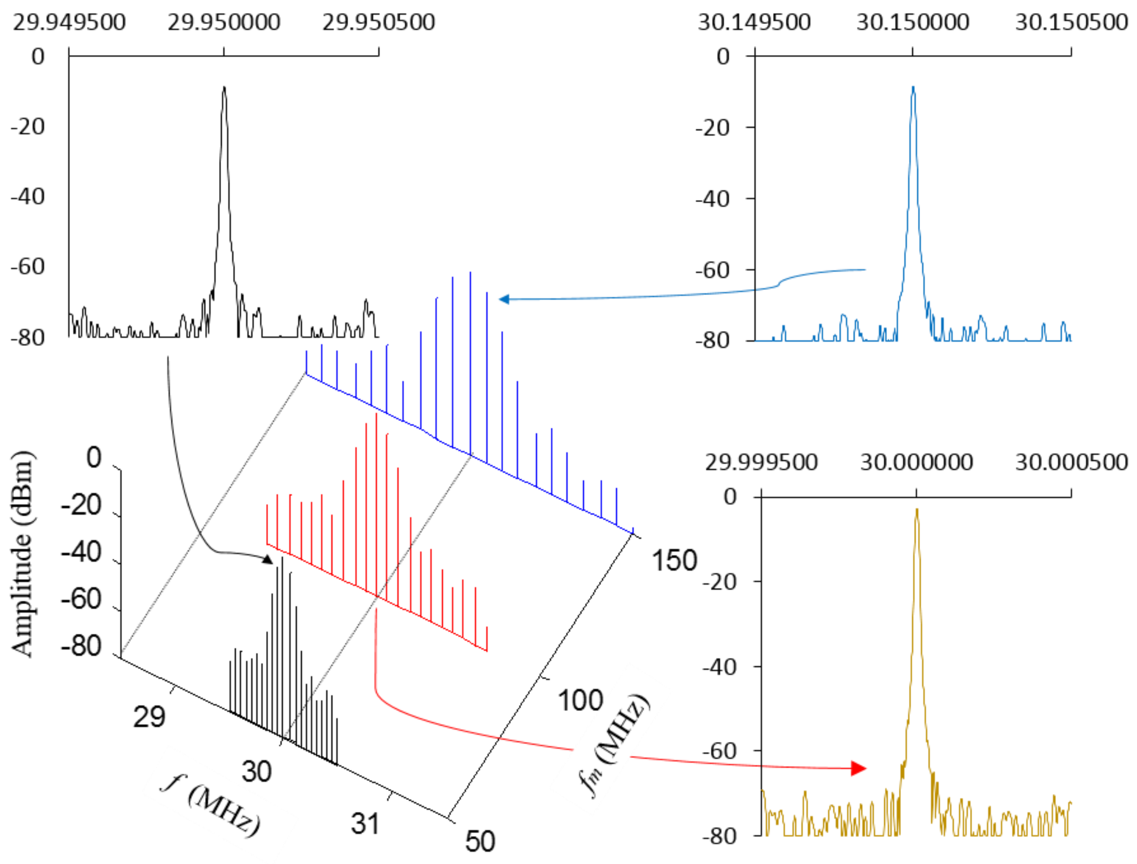


Figure Appendix B-2- Spectra of signals of $f_c = 30$ MHz and $f_m =$ 50, 100, and 150 kHz measured with frequency steps of 1 Hz.

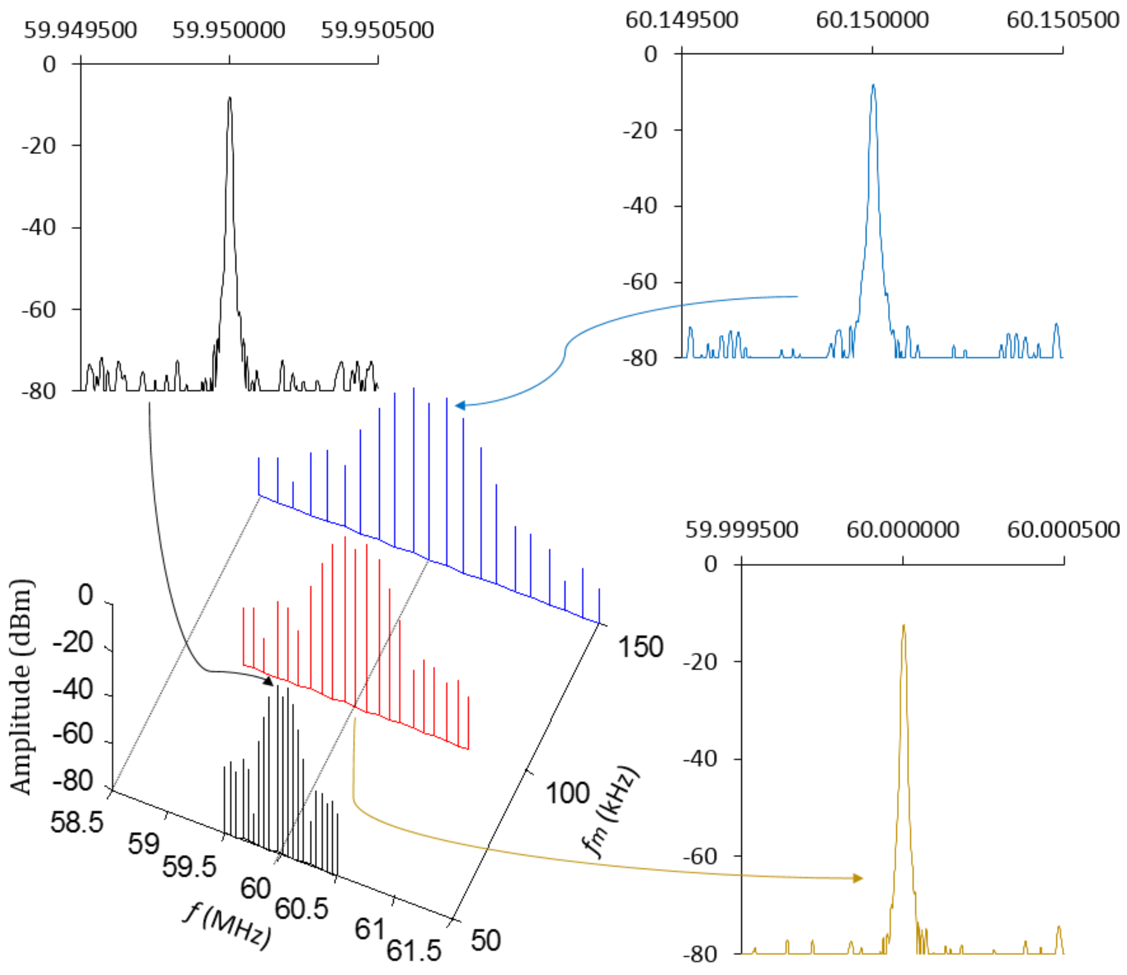


Figure Appendix B-3- Spectra of signals of $f_c = 60$ MHz and $f_m =$ 50, 100, and 150 kHz measured with frequency steps of 1 Hz.

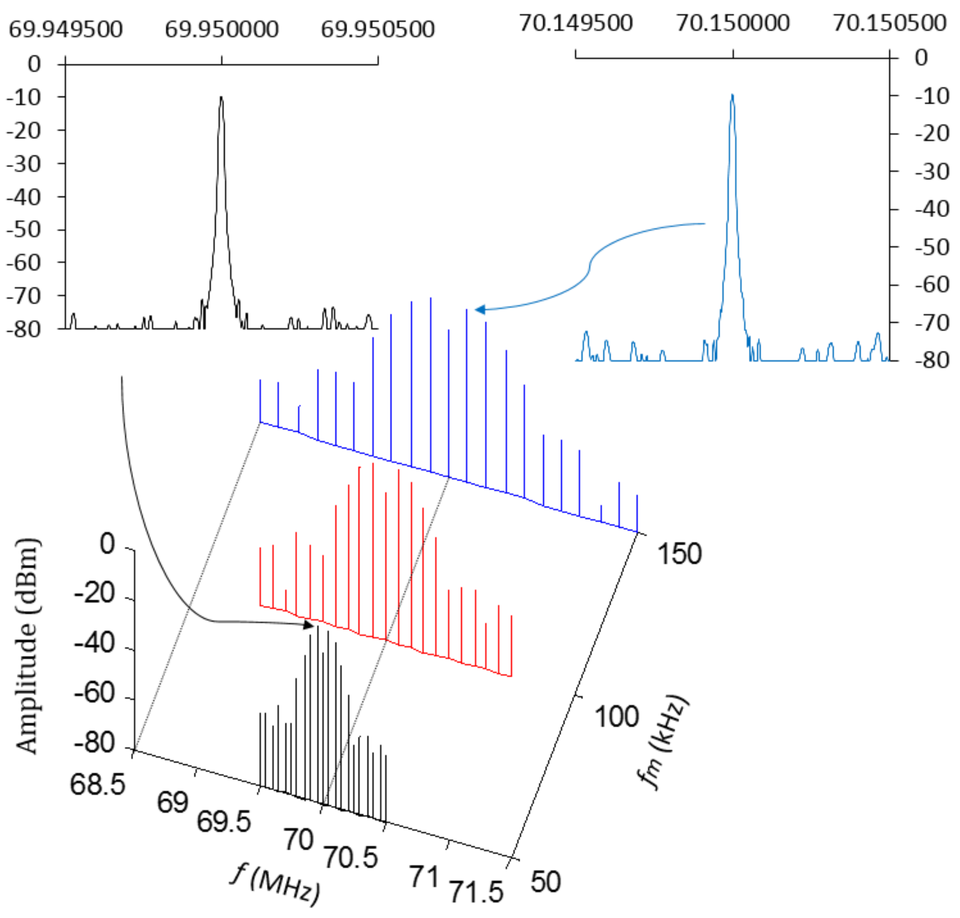


Figure Appendix B-4- Spectra of signals of $f_c = 70$ MHz and $f_m =$ 50, 100, and 150 kHz measured with frequency steps of 1 Hz.

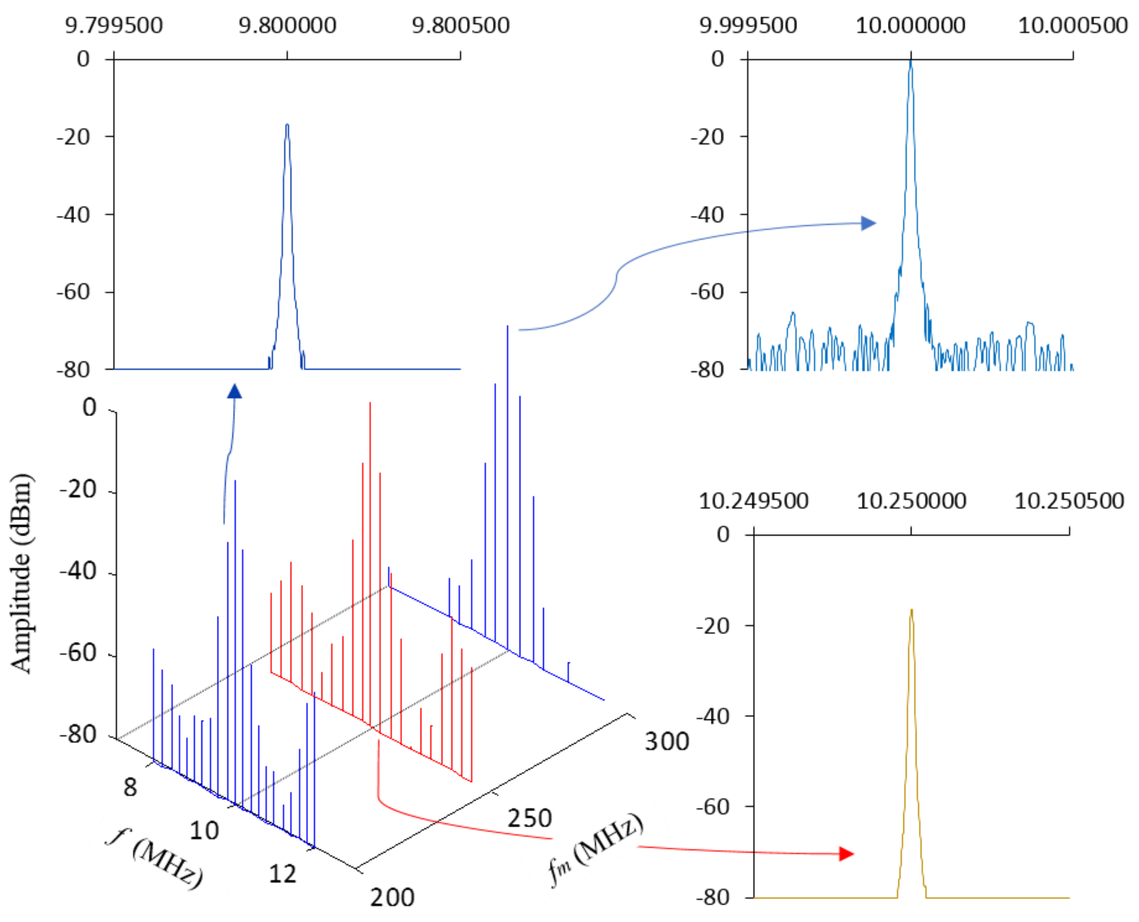


Figure Appendix B-5- Spectra of signals of $f_c = 10$ MHz and $f_m =$ 200, 250, and 300 kHz measured with frequency steps of 1 Hz.

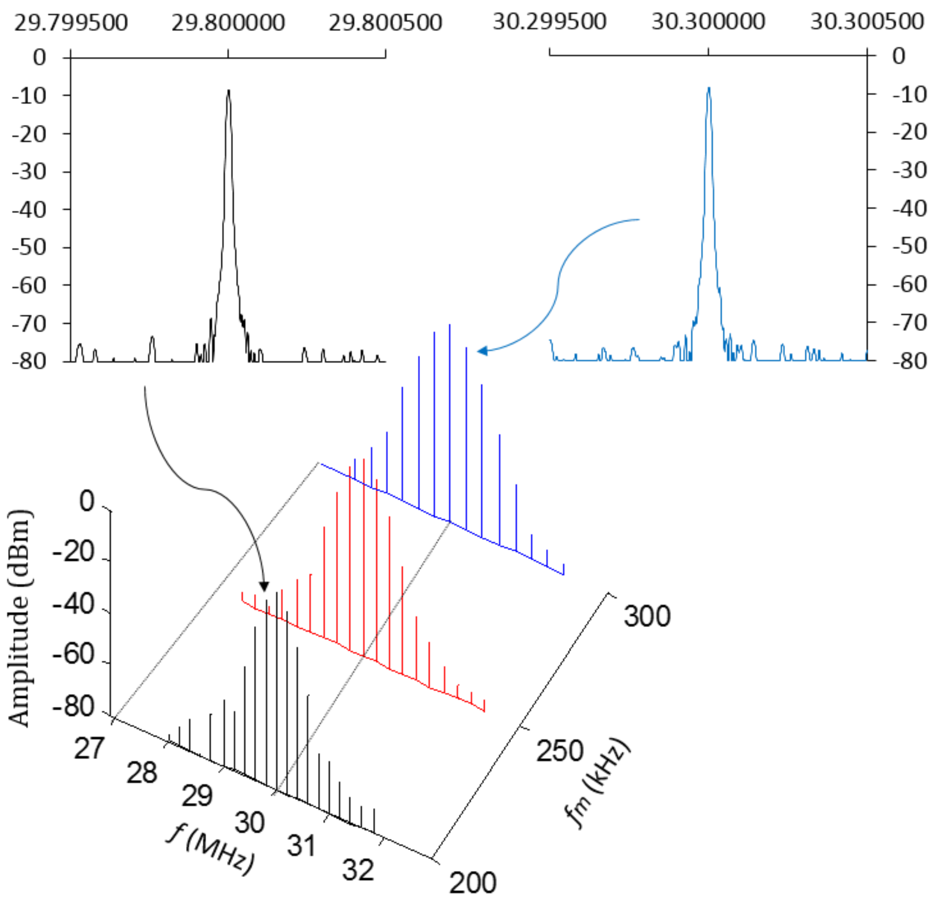


Figure Appendix B-6- Spectra of signals of $f_c = 30$ MHz and $f_m =$ 200, 250, and 300 kHz measured with frequency steps of 1 Hz.

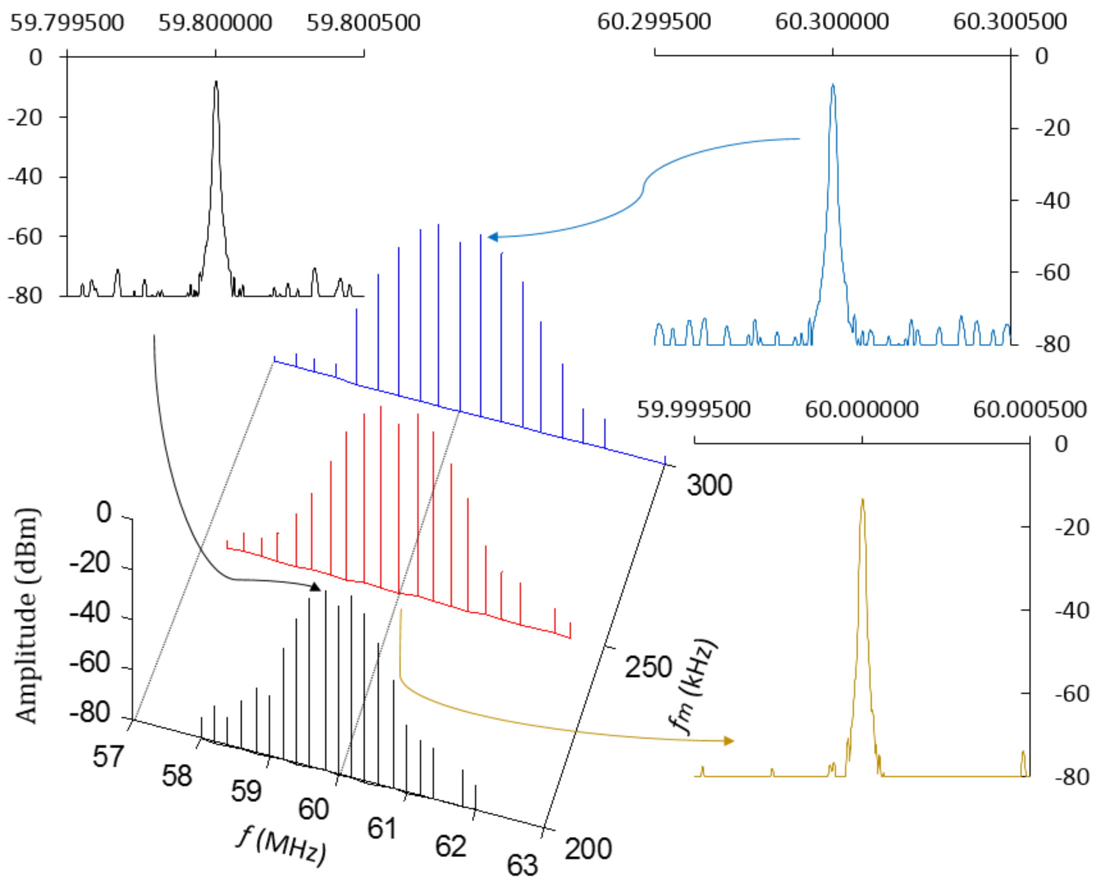


Figure Appendix B-7- Spectra of signals of $f_c = 60$ MHz and $f_m =$ 200, 250, and 300 kHz measured with frequency steps of 1 Hz.

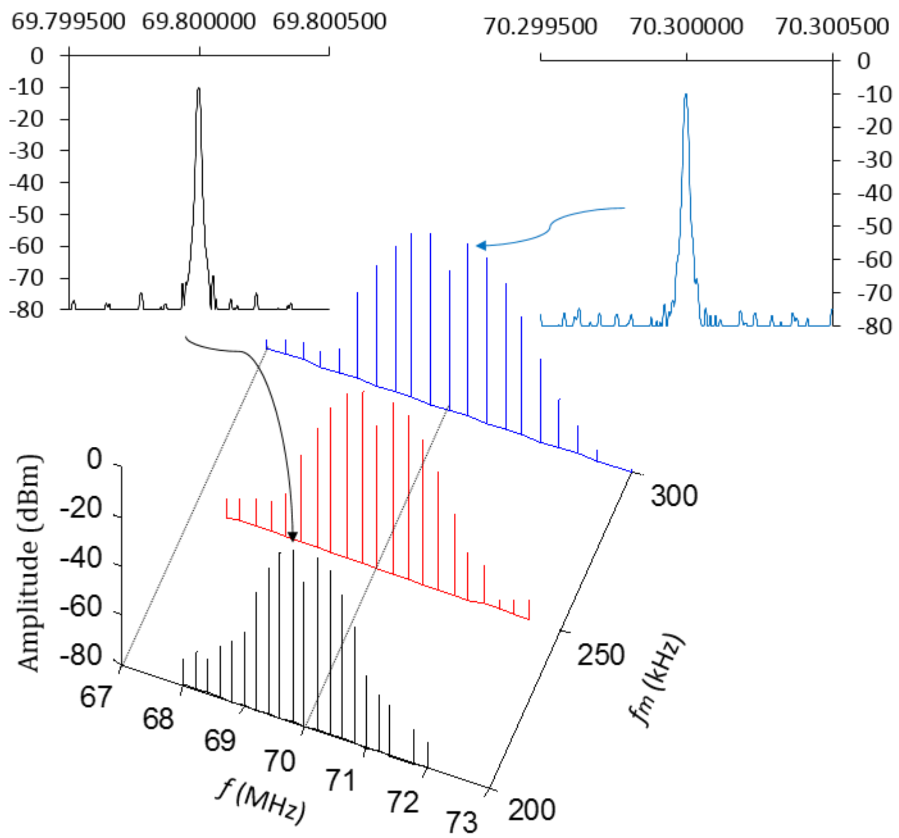


Figure Appendix B-8- Spectra of signals of $f_c = 70$ MHz and $f_m =$ 200, 250, and 300 kHz measured with frequency steps of 1 Hz.

APPENDIX C. VHDL code for 32-sample Fourier transform

The main module of the VHDL code for 32-sample Fourier transform is presented here. However, there are more than hundred submodules in this code that makes it infeasible to represent them in an appendix. Therefore, a complete version of the 32-sample Fourier transform VHDL code and the LabVIEW program that are used for phase extraction and unwrapping are attached to this document. Each of the codes are self-explanatory so that the readme file inside the codes navigate the users to operate the system easily.

```
----------------------------------------- Code Description -----------------------------
-- Company: UNC Charlotte
-- Student: Masoud Arablu
--
-- Create Date: 10/17/2018 08:06:14 AM
-- Design Name: Discrete Fourier Transform
-- Module Name: Top - Behavior
-- Project Name: Digital Methods for Phase Extraction in Interferometry
-- Target Devices: NEXYS 4 DDR
--
-- Description: This VHDL code is developed in Vivado 2018.3. It uses add/sub,
-- multiplier, Clock Wizard, CORDIC, and ILA IPs for implementation and debugging
-- DFT. Also, component bin2bcd converts binary to decimal and component DigitToSeg
-- monitors values in the 7-segment LEDs.
--
-- Revision 16 - Algorithm was expanded to two harmonics of 312.5 kHz.
-- Additional Comments: This algorithm reads 12 bit digital data in
-- parallel mode at 10 MHz speed and applies 32 sample digital DFT followed
-- by 32-points averaging.
--------------------------------------------- Libraries ---------------------------------1
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;
-------------------------------------------------------------------------------------end1
--------------------------------------------- Entity ------------------------------------2
entity Top is
```

```
    Port (
        CLK          : in  std_logic;
        CLK_Reset : in  std_logic;
        D              : in  std_logic_vector(11 downto 0);
        Photo_CLK : out std_logic := '0';
        Dp             : out std_logic;
        Seg            : out std_logic_vector(6 downto 0);
        an              : out std_logic_vector(7 downto 0);
        LED           : out std_logic_vector(15 downto 0);
        --sw           : in  std_logic_vector(1 downto 0):=( others=> '0');
        D_Out        : out std_logic_vector(15 downto 0) := ( others=> '0');
        Dec_CLK    : out std_logic := '0'
    );
end Top;
-----------------------------------------------------------------------------------------end2
------------------------------------------ Architecture ----------------------------------3
architecture Behavior of Top is
---------- Declaring variables and IPs ---------------------------------3.1-to-3.20
    signal count        : integer range 0 to 10000;
    signal count2      : integer range 0 to 10000;
    signal dig0          : std_logic_vector(3 downto 0);
    signal dig1          : std_logic_vector(3 downto 0);
    signal dig2          : std_logic_vector(3 downto 0);
    signal dig3          : std_logic_vector(3 downto 0);
    signal dig4          : std_logic_vector(3 downto 0);
    signal dig5          : std_logic_vector(3 downto 0);
    signal dig6          : std_logic_vector(3 downto 0);
    signal Bin_Data   : std_logic_vector(15 downto 0);
    -------------------------------------------------------
    signal MAIN_CLK        : std_logic;
    signal CLK_100MHz     : std_logic;
    signal reset_bcd          : std_logic;
    signal counter1            : integer range 0 to 511 := 0;
    signal counter2            : integer range 0 to 511 := 0;
    signal counter3            : integer range 0 to 511 := 0;
    signal probe_0              : std_logic_vector(11 downto 0) := ( others=> '0');
    signal probe_1              : std_logic_vector(0 downto 0)   := ( others=> '0');
    signal probe_2              : std_logic_vector(0 downto 0)   := ( others=> '0');
    signal probe_3              : std_logic_vector(0 downto 0)   := ( others=> '0');
    signal probe_4              : std_logic_vector(15 downto 0) := ( others=> '0');
    signal MMCM_Locked  : std_logic;
    signal D_In_Ready        : std_logic := '0';
    signal D_Out_Ready      : std_logic := '0';
    signal data                   : std_logic_vector(11 downto 0) := ( others=> '0');
    -------------------------------------------------------
    type mem32_12 is array (31 downto 0) of std_logic_vector (11 downto 0);
    signal data_1 : mem32_12 := (others => ( others=> '0'));
    -------------------------------------------------------
    type mem32_16 is array (31 downto 0) of std_logic_vector (15 downto 0);
    -- C1 is a 32-sample cosine LUT that contains 1 complete period, i.e. for first harmonic
    signal C1 : mem32_16 :=
("0111111111111111","0111110110001010","0111011001000010","0110101001101110",

    "0101101010000010","0100011100011101","0011000011111100","0001100011111001",

    "0000000000000000","1110011100000111","1100111100000100","1011100011100011",
    "1010010101111110","1001010110010010","1000100110111110","1000001001110110",

    "1000000000000000","1000001001110110","1000100110111110","1001010110010010",

    "1010010101111110","1011100011100011","1100111100000100","1110011100000111",

    "0000000000000000","0001100011111001","0011000011111100","0100011100011101",

    "0101101010000010","0110101001101110","0111011001000010","0111110110001010");
    -- S1 is a 32-sample sine LUT that contains 1 complete period, i.e. for first harmonic
    signal S1 : mem32_16 :=
   ("0000000000000000","0001100011111001","0011000011111100","0100011100011101",

    "0101101010000010","0110101001101110","0111011001000010","0111110110001010",

    "0111111111111111","0111110110001010","0111011001000010","0110101001101110",

    "0101101010000010","0100011100011101","0011000011111100","0001100011111001",

    "0000000000000000","1110011100000111","1100111100000100","1011100011100011",

    "1010010101111110","1001010110010010","1000100110111110","1000001001110110",

    "1000000000000000","1000001001110110","1000100110111110","1001010110010010",

    "1010010101111110","1011100011100011","1100111100000100","1110011100000111");
    -- C2 is a 32-sample cosine LUT that contains 2 complete periods, i.e. for second harmonic
    signal C2 : mem32_16 :=
   ("0111111111111111","0111011001000010","0101101010000010","0011000011111100",

    "0000000000000000","1100111100000100","1010010101111110","1000100110111110",

    "1000000000000000","1000100110111110","1010010101111110","1100111100000100",

    "0000000000000000","0011000011111100","0101101010000010","0111011001000010",

    "0111111111111111","0111011001000010","0101101010000010","0011000011111100",

    "0000000000000000","1100111100000100","1010010101111110","1000100110111110",

    "1000000000000000","1000100110111110","1010010101111110","1100111100000100",

    "0000000000000000","0011000011111100","0101101010000010","0111011001000010");
    -- S2 is a 32-sample sine LUT that contains 2 complete periods, i.e. for second harmonic
    signal S2 : mem32_16 :=
   ("0000000000000000","0011000011111100","0101101010000010","0111011001000010",

    "0111111111111111","0111011001000010","0101101010000010","0011000011111100",

    "0000000000000000","1100111100000100","1010010101111110","1000100110111110",

    "1000000000000000","1000100110111110","1010010101111110","1100111100000100",

    "0000000000000000","0011000011111100","0101101010000010","0111011001000010",
   "0111111111111111","0111011001000010","0101101010000010","0011000011111100",

   "0000000000000000","1100111100000100","1010010101111110","1000100110111110",

   "1000000000000000","1000100110111110","1010010101111110","1100111100000100");
   -------------------------------------------------------
   type mem32_28 is array (31 downto 0) of std_logic_vector (27 downto 0);
   signal X1_1  : mem32_28 := (others => ( others=> '0'));
   signal Y1_1  : mem32_28 := (others => ( others=> '0'));
   signal X2_1  : mem32_28 := (others => ( others=> '0'));
   signal Y2_1  : mem32_28 := (others => ( others=> '0'));
   -------------------------------------------------------
   type mem16_29 is array (15 downto 0) of std_logic_vector (28 downto 0);
   signal X1_2  : mem16_29 := (others => ( others=> '0'));
   signal Y1_2  : mem16_29 := (others => ( others=> '0'));
   signal X2_2  : mem16_29 := (others => ( others=> '0'));
   signal Y2_2  : mem16_29 := (others => ( others=> '0'));
   -------------------------------------------------------
   type mem8_30 is array (7 downto 0) of std_logic_vector (29 downto 0);
   signal X1_3  : mem8_30 := (others => ( others=> '0'));
   signal Y1_3  : mem8_30 := (others => ( others=> '0'));
   signal X2_3  : mem8_30 := (others => ( others=> '0'));
   signal Y2_3  : mem8_30 := (others => ( others=> '0'));
   -------------------------------------------------------
   type mem4_31 is array (3 downto 0) of std_logic_vector (30 downto 0);
   signal X1_4  : mem4_31 := (others => ( others=> '0'));
   signal Y1_4  : mem4_31 := (others => ( others=> '0'));
   signal X2_4  : mem4_31 := (others => ( others=> '0'));
   signal Y2_4  : mem4_31 := (others => ( others=> '0'));
   -------------------------------------------------------
   type mem2_32 is array (1 downto 0) of std_logic_vector (31 downto 0);
   signal X1_5  : mem2_32 := (others => ( others=> '0'));
   signal Y1_5  : mem2_32 := (others => ( others=> '0'));
   signal X2_5  : mem2_32 := (others => ( others=> '0'));
   signal Y2_5  : mem2_32 := (others => ( others=> '0'));
   -------------------------------------------------------
   type mem2_33 is array (1 downto 0) of std_logic_vector (32 downto 0);
   signal X1_6  : mem2_33 := (others => ( others=> '0'));
   signal Y1_6  : mem2_33 := (others => ( others=> '0'));
   signal X2_6  : mem2_33 := (others => ( others=> '0'));
   signal Y2_6  : mem2_33 := (others => ( others=> '0'));
   -------------------------------------------------------
   signal X1    : std_logic_vector(15 downto 0) := ( others=> '0');
   signal Y1    : std_logic_vector(15 downto 0) := ( others=> '0');
   signal X2    : std_logic_vector(15 downto 0) := ( others=> '0');
   signal Y2    : std_logic_vector(15 downto 0) := ( others=> '0');
   -------------------------------------------------------
   type mem32_33 is array (31 downto 0) of std_logic_vector (32 downto 0);
   signal Ave1_X1 : mem32_33 := (others => ( others=> '0'));
   signal Ave1_Y1 : mem32_33 := (others => ( others=> '0'));
   signal Ave1_X2 : mem32_33 := (others => ( others=> '0'));
   signal Ave1_Y2 : mem32_33 := (others => ( others=> '0'));
   -------------------------------------------------------
   type mem16_34 is array (15 downto 0) of std_logic_vector (33 downto 0);
   signal Ave2_X1 : mem16_34 := (others => ( others=> '0'));
   signal Ave2_Y1 : mem16_34 := (others => ( others=> '0'));
   signal Ave2_X2 : mem16_34 := (others => ( others=> '0'));
   signal Ave2_Y2 : mem16_34 := (others => ( others=> '0'));
   ---------------------------------------------------------
   type mem8_35 is array (7 downto 0) of std_logic_vector (34 downto 0);
   signal Ave3_X1 : mem8_35 := (others => ( others=> '0'));
   signal Ave3_Y1 : mem8_35 := (others => ( others=> '0'));
   signal Ave3_X2 : mem8_35 := (others => ( others=> '0'));
   signal Ave3_Y2 : mem8_35 := (others => ( others=> '0'));
   ---------------------------------------------------------
   type mem4_36 is array (3 downto 0) of std_logic_vector (35 downto 0);
   signal Ave4_X1 : mem4_36 := (others => ( others=> '0'));
   signal Ave4_Y1 : mem4_36 := (others => ( others=> '0'));
   signal Ave4_X2 : mem4_36 := (others => ( others=> '0'));
   signal Ave4_Y2 : mem4_36 := (others => ( others=> '0'));
   ---------------------------------------------------------
   type mem2_37 is array (1 downto 0) of std_logic_vector (36 downto 0);
   signal Ave5_X1 : mem2_37 := (others => ( others=> '0'));
   signal Ave5_Y1 : mem2_37 := (others => ( others=> '0'));
   signal Ave5_X2 : mem2_37 := (others => ( others=> '0'));
   signal Ave5_Y2 : mem2_37 := (others => ( others=> '0'));
   ---------------------------------------------------------
   type mem2_38 is array (1 downto 0) of std_logic_vector (37 downto 0);
   signal Ave6_X1 : mem2_38 := (others => ( others=> '0'));
   signal Ave6_Y1 : mem2_38 := (others => ( others=> '0'));
   signal Ave6_X2 : mem2_38 := (others => ( others=> '0'));
   signal Ave6_Y2 : mem2_38 := (others => ( others=> '0'));
   ---------------------------------------------------------
   signal Ave_X1        : std_logic_vector(15 downto 0) := ( others=> '0');
   signal Ave_Y1        : std_logic_vector(15 downto 0) := ( others=> '0');
   signal Ave_X11       : std_logic_vector(15 downto 0) := ( others=> '0');
   signal Ave_Y11       : std_logic_vector(15 downto 0) := ( others=> '0');
   signal Ave_XY1       : std_logic_vector(31 downto 0) := ( others=> '0');
   signal Ave_X2        : std_logic_vector(15 downto 0) := ( others=> '0');
   signal Ave_Y2        : std_logic_vector(15 downto 0) := ( others=> '0');
   signal Ave_XY2       : std_logic_vector(31 downto 0) := ( others=> '0');
   signal Ave_PHI1      : std_logic_vector(15 downto 0) := ( others=> '0');
   signal Ave_PHI2      : std_logic_vector(15 downto 0) := ( others=> '0');
   signal sqr_Ave_X1    : std_logic_vector(31 downto 0) := ( others=> '0');
   signal sqr_Ave_Y1    : std_logic_vector(31 downto 0) := ( others=> '0');
   signal sqr_Ave_X2    : std_logic_vector(31 downto 0) := ( others=> '0');
   signal sqr_Ave_Y2    : std_logic_vector(31 downto 0) := ( others=> '0');
   signal sqr_Ave_R1    : std_logic_vector(32 downto 0) := ( others=> '0');
   signal sqr_Ave_R2    : std_logic_vector(32 downto 0) := ( others=> '0');
   signal sqr_Ave_R1_2  : std_logic_vector(39 downto 0) := ( others=> '0');
   signal sqr_Ave_R2_2  : std_logic_vector(39 downto 0) := ( others=> '0');
   signal Ave_R1        : std_logic_vector(23 downto 0) := ( others=> '0');
   signal Ave_R2        : std_logic_vector(23 downto 0) := ( others=> '0');
   --------------------------------------------------------------------------------end3.1
   component mult_gen_0
   port (  A     : in    std_logic_vector(11 downto 0);
           B     : in    std_logic_vector(15 downto 0);
           CLK   : in    std_logic;
           P     : out   std_logic_vector(27 downto 0)          );
   end component mult_gen_0;
   --------------------------------------------------------------------------------end3.2
```

```
component mult_gen_1
port (  A       : in      std_logic_vector(15 downto 0);
        B       : in      std_logic_vector(15 downto 0);
        CLK     : in      std_logic;
        P       : out     std_logic_vector(31 downto 0)          );
end component mult_gen_1;
---------------------------------------------------------------------------------end3.3
component c_addsub_0
port (  A       : in      std_logic_vector(27 downto 0);
        B       : in      std_logic_vector(27 downto 0);
        CLK     : in      std_logic;
        S       : out     std_logic_vector(28 downto 0)          );
end component c_addsub_0;
---------------------------------------------------------------------------------end3.4
component c_addsub_1
port (  A       : in      std_logic_vector(28 downto 0);
        B       : in      std_logic_vector(28 downto 0);
        CLK     : in      std_logic;
        S       : out     std_logic_vector(29 downto 0)          );
end component c_addsub_1;
---------------------------------------------------------------------------------end3.5
component c_addsub_2
port (  A       : in      std_logic_vector(29 downto 0);
        B       : in      std_logic_vector(29 downto 0);
        CLK     : in      std_logic;
        S       : out     std_logic_vector(30 downto 0)          );
end component c_addsub_2;
---------------------------------------------------------------------------------end3.6
component c_addsub_3
port (  A       : in      std_logic_vector(30 downto 0);
        B       : in      std_logic_vector(30 downto 0);
        CLK     : in      std_logic;
        S       : out     std_logic_vector(31 downto 0)          );
end component c_addsub_3;
---------------------------------------------------------------------------------end3.7
component c_addsub_4
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(32 downto 0)          );
end component c_addsub_4;
---------------------------------------------------------------------------------end3.8
component c_addsub_5
port (  A       : in      std_logic_vector(32 downto 0);
        B       : in      std_logic_vector(32 downto 0);
        CLK     : in      std_logic;
        S       : out     std_logic_vector(33 downto 0)          );
end component c_addsub_5;
---------------------------------------------------------------------------------end3.9
component c_addsub_6
port (  A       : in      std_logic_vector(33 downto 0);
        B       : in      std_logic_vector(33 downto 0);
        CLK     : in      std_logic;
        S       : out     std_logic_vector(34 downto 0)          );
end component c_addsub_6;
---------------------------------------------------------------------------------end3.10
```

```
component c_addsub_7
port (   A       : in     std_logic_vector(34 downto 0);
         B       : in     std_logic_vector(34 downto 0);
         CLK     : in     std_logic;
         S       : out    std_logic_vector(35 downto 0)          );
end component c_addsub_7;
------------------------------------------------------------------------------------end3.11
component c_addsub_8
port (   A       : in     std_logic_vector(35 downto 0);
         B       : in     std_logic_vector(35 downto 0);
         CLK     : in     std_logic;
         S       : out    std_logic_vector(36 downto 0)          );
end component c_addsub_8;
------------------------------------------------------------------------------------end3.12
component c_addsub_9
port (   A       : in     std_logic_vector(36 downto 0);
         B       : in     std_logic_vector(36 downto 0);
         CLK     : in     std_logic;
         S       : out    std_logic_vector(37 downto 0)          );
end component c_addsub_9;
------------------------------------------------------------------------------------end3.13
component c_addsub_10
port (   A       : in     std_logic_vector(15 downto 0);
         CLK     : in     std_logic;
         S       : out    std_logic_vector (15 downto 0)         );
end component c_addsub_10;
------------------------------------------------------------------------------------end3.14
component ila_0
port (   clk       : in   std_logic;
         probe0    : in   std_logic_vector(11 downto 0);
         probe1    : in   std_logic_vector(0 downto 0);
         probe2    : in   std_logic_vector(0 downto 0);
         probe3    : in   std_logic_vector(0 downto 0);
         probe4    : in   std_logic_vector(15 downto 0)          );
end component ila_0;
------------------------------------------------------------------------------------end3.15
component cordic_0
port (   aclk                      : in     std_logic;
         s_axis_cartesian_tvalid   : in     std_logic;
         s_axis_cartesian_tdata    : in     std_logic_vector(39 downto 0);
         m_axis_dout_tvalid        : out    std_logic;
         m_axis_dout_tdata         : out    std_logic_vector(23 downto 0)     );
end component;
------------------------------------------------------------------------------------end3.16
component cordic_1
port (   aclk                      : in     std_logic;
         s_axis_cartesian_tvalid   : in     std_logic;
         s_axis_cartesian_tdata    : in     std_logic_vector(31 downto 0);
         m_axis_dout_tvalid        : out    std_logic;
         m_axis_dout_tdata         : out    std_logic_vector(15 downto 0)     );
end component;
------------------------------------------------------------------------------------end3.17
component clk_wiz_0
port (   clk_out1  : out    std_logic;
         clk_out2  : out    std_logic;
         clk_out3  : out    std_logic;
```

```
            reset       : in     std_logic;
            locked      : out    std_logic;
            clk_in1     : in     std_logic     );
   end component;
   --------------------------------------------------------------------------------end3.18
   component DigitToSeg
   port (   in1         : in     std_logic_vector(3 downto 0);
            in2         : in     std_logic_vector(3 downto 0);
            in3         : in     std_logic_vector(3 downto 0);
            in4         : in     std_logic_vector(3 downto 0);
            in5         : in     std_logic_vector(3 downto 0);
            in6         : in     std_logic_vector(3 downto 0);
            in7         : in     std_logic_vector(3 downto 0);
            in8         : in     std_logic_vector(3 downto 0);
            mclk        : in     std_logic;
            an          : out    std_logic_vector(7 downto 0);
            dp          : out    std_logic;
            seg         : out    std_logic_vector(6 downto 0)     );
   end component DigitToSeg;
   --------------------------------------------------------------------------------end3.19
   component bin2bcd
   port (   mclk, reset : in std_logic;
            binary_in   : in std_logic_vector(15 downto 0);
            bcd0, bcd1, bcd2, bcd3, bcd4, bcd5, bcd6 : out std_logic_vector(3 downto 0)  );
   end component bin2bcd;
   --------------------------------------------------------------------------------end3.20
-------------------------------- start of DFT processes ----------------------------------
begin
   -- multiplying sine and cosine arrays to the signal, the outputs are 28 bit -- 3.21
   multiplier0: for i in 0 to 31 generate -- generates 28 bit data
      gives_X1_1 : component mult_gen_0
      port map (   CLK    => MAIN_CLK,
                   A      => data_1(i),
                   B      => C1(i),
                   P      => X1_1(i)          );
      ---------------------------
      gives_Y1_1 : component mult_gen_0
      port map (   CLK    => MAIN_CLK,
                   A      => data_1(i),
                   B      => S1(i),
                   P      => Y1_1(i)          );
      ---------------------------
      gives_X2_1 : component mult_gen_0
      port map (   CLK    => MAIN_CLK,
                   A      => data_1(i),
                   B      => C2(i),
                   P      => X2_1(i)          );
      ---------------------------
      gives_Y2_1 : component mult_gen_0
      port map (   CLK    => MAIN_CLK,
                   A      => data_1(i),
                   B      => S2(i),
                   P      => Y2_1(i)          );
   end generate multiplier0;
   --------------------------------------------------------------------------------end3.21
   ----------------- summing 28 bit data results in 29 bit data ------------------ 3.22
```

```
adder0: for i in 0 to 15 generate -- generates 29 bit data
   gives_X1_2 : component c_addsub_0
   port map (   CLK    => MAIN_CLK,
                A      => X1_1(i),
                B      => X1_1(i+16),
                S      => X1_2(i)          );
   ----------------------------
   gives_Y1_2 : component c_addsub_0
   port map (   CLK    => MAIN_CLK,
                A      => Y1_1(i),
                B      => Y1_1(i+16),
                S      => Y1_2(i)          );
   ----------------------------
   gives_X2_2 : component c_addsub_0
   port map (   CLK    => MAIN_CLK,
                A      => X2_1(i),
                B      => X2_1(i+16),
                S      => X2_2(i)          );
   ----------------------------
   gives_Y2_2 : component c_addsub_0
   port map (   CLK    => MAIN_CLK,
                A      => Y2_1(i),
                B      => Y2_1(i+16),
                S      => Y2_2(i)          );
end generate adder0;
-------------------------------------------------------------------------------------end3.22
------------------ summing 29 bit data results in 30 bit data ------------------- 3.23
adder1: for i in 0 to 7 generate -- generates 30 bit data
   gives_X1_3 : component c_addsub_1
   port map (   CLK    => MAIN_CLK,
                A      => X1_2(i),
                B      => X1_2(i+8),
                S      => X1_3(i)          );
   ----------------------------
   gives_Y1_3 : component c_addsub_1
   port map (   CLK    => MAIN_CLK,
                A      => Y1_2(i),
                B      => Y1_2(i+8),
                S      => Y1_3(i)          );
   ----------------------------
   gives_X2_3 : component c_addsub_1
   port map (   CLK    => MAIN_CLK,
                A      => X2_2(i),
                B      => X2_2(i+8),
                S      => X2_3(i)          );
   ----------------------------
   gives_Y2_3 : component c_addsub_1
   port map (   CLK    => MAIN_CLK,
                A      => Y2_2(i),
                B      => Y2_2(i+8),
                S      => Y2_3(i)          );
end generate adder1;
-------------------------------------------------------------------------------------end3.23
------------------ summing 30 bit data results in 31 bit data ------------------- 3.24

adder2: for i in 0 to 3 generate -- generates 31 bit data
```

```
    gives_X1_4 : component c_addsub_2
    port map (   CLK     => MAIN_CLK,
                 A       => X1_3(i),
                 B       => X1_3(i+4),
                 S       => X1_4(i)           );
    ----------------------------
    gives_Y1_4 : component c_addsub_2
    port map (   CLK     => MAIN_CLK,
                 A       => Y1_3(i),
                 B       => Y1_3(i+4),
                 S       => Y1_4(i)           );
    ----------------------------
    gives_X2_4 : component c_addsub_2
    port map (   CLK     => MAIN_CLK,
                 A       => X2_3(i),
                 B       => X2_3(i+4),
                 S       => X2_4(i)           );
    ----------------------------
    gives_Y2_4 : component c_addsub_2
    port map (   CLK     => MAIN_CLK,
                 A       => Y2_3(i),
                 B       => Y2_3(i+4),
                 S       => Y2_4(i)           );
end generate adder2;
-------------------------------------------------------------------------------------end3.24
------------------ summing 31 bit data results in 32 bit data ------------------- 3.25
adder3: for i in 0 to 1 generate -- generates 32 bit data
    gives_X1_5 : component c_addsub_3
    port map (   CLK     => MAIN_CLK,
                 A       => X1_4(i),
                 B       => X1_4(i+2),
                 S       => X1_5(i)           );
    ----------------------------
    gives_Y1_5 : component c_addsub_3
    port map (   CLK     => MAIN_CLK,
                 A       => Y1_4(i),
                 B       => Y1_4(i+2),
                 S       => Y1_5(i)           );
    ----------------------------
    gives_X2_5 : component c_addsub_3
    port map (   CLK     => MAIN_CLK,
                 A       => X2_4(i),
                 B       => X2_4(i+2),
                 S       => X2_5(i)           );
    ----------------------------
    gives_Y2_5 : component c_addsub_3
    port map (   CLK     => MAIN_CLK,
                 A       => Y2_4(i),
                 B       => Y2_4(i+2),
                 S       => Y2_5(i)           );
end generate adder3;
-------------------------------------------------------------------------------------end3.25
------------------ summing 32 bit data results in 33 bit data ------------------- 3.26
gives_X1_6_0 : component c_addsub_4 -- generates 33 bit data
port map (       CLK     => MAIN_CLK,
                 A       => X1_5(0),
                        B         => X1_5(1),
                        S         => X1_6(0)            );
----------------------------
gives_Y1_6_0 : component c_addsub_4 -- generates 33 bit data
port map (       CLK      => MAIN_CLK,
                        A         => Y1_5(0),
                        B         => Y1_5(1),
                        S         => Y1_6(0)            );
------------------------------------------------------------------------------------end3.26
--------------- Divide to ((half the # of samples)*(LUT amp.)) --------------- 3.27
-- X1_6(0) is divided to (16*2^15) and saved into 16-bit X1. Divide less to amplify it.
X1_6(1)   <= std_logic_vector(shift_right(signed(X1_6(0)), 12));
X1          <= X1_6(0)(32) & X1_6(1)(14 downto 0);
-- Y1_6(0) is divided to (16*2^15) and saved into 16-bit Y1. Divide less to amplify it.
Y1_6(1)   <= std_logic_vector(shift_right(signed(Y1_6(0)), 12));
Y1          <= Y1_6(0)(32) & Y1_6(1)(14 downto 0);
----------------------------
gives_X2_6_0 : component c_addsub_4 -- generates 33 bit data
port map (   CLK      => MAIN_CLK,
                    A         => X2_5(0),
                    B         => X2_5(1),
                    S         => X2_6(0)               );
----------------------------
gives_Y2_6_0 : component c_addsub_4 -- generates 33 bit data
port map (   CLK      => MAIN_CLK,
                    A         => Y2_5(0),
                    B         => Y2_5(1),
                    S         => Y2_6(0)               );
----------------------------
-- X2_6(0) is divided to (16*2^15) and saved into 16-bit X2. Divide less to amplify it.
X2_6(1)   <= std_logic_vector(shift_right(signed(X2_6(0)), 12));
X2          <= X2_6(0)(32) & X2_6(1)(14 downto 0);
-- Y2_6(0) is divided to (16*2^15) and saved into 16-bit Y2. Divide less to amplify it.
Y2_6(1)   <= std_logic_vector(shift_right(signed(Y2_6(0)), 12));
Y2          <= Y2_6(0)(32) & Y2_6(1)(14 downto 0);
------------------------------------------------------------------------------------end3.27
----------------------------- Average 32 values and report as Ave_Q values -------
adder5: for i in 0 to 15 generate -- generates 34 bit data
   gives_Ave2_X1 : component c_addsub_5
   port map (   CLK      => MAIN_CLK,
                        A         => Ave1_X1(i),
                        B         => Ave1_X1(i+16),
                        S         => Ave2_X1(i)       );
   ----------------------------
   gives_Ave2_Y1 : component c_addsub_5
   port map (   CLK      => MAIN_CLK,
                        A         => Ave1_Y1(i),
                        B         => Ave1_Y1(i+16),
                        S         => Ave2_Y1(i)       );
   ----------------------------
   gives_Ave2_X2 : component c_addsub_5
   port map (   CLK      => MAIN_CLK,
                        A         => Ave1_X2(i),
                        B         => Ave1_X2(i+16),
                        S         => Ave2_X2(i)       );
   ----------------------------
   gives_Ave2_Y2 : component c_addsub_5
   port map (   CLK     => MAIN_CLK,
                A       => Ave1_Y2(i),
                B       => Ave1_Y2(i+16),
                S       => Ave2_Y2(i)       );
end generate adder5;
--------------------------------------------------------------------------------end3.28
adder6: for i in 0 to 7 generate -- generates 35 bit data
   gives_Ave3_X1 : component c_addsub_6
   port map (   CLK     => MAIN_CLK,
                A       => Ave2_X1(i),
                B       => Ave2_X1(i+8),
                S       => Ave3_X1(i)       );
   ----------------------------
   gives_Ave3_Y1 : component c_addsub_6
   port map (   CLK     => MAIN_CLK,
                A       => Ave2_Y1(i),
                B       => Ave2_Y1(i+8),
                S       => Ave3_Y1(i)       );
   ----------------------------
   gives_Ave3_X2 : component c_addsub_6
   port map (   CLK     => MAIN_CLK,
                A       => Ave2_X2(i),
                B       => Ave2_X2(i+8),
                S       => Ave3_X2(i)       );
   ----------------------------
   gives_Ave3_Y2 : component c_addsub_6
   port map (   CLK     => MAIN_CLK,
                A       => Ave2_Y2(i),
                B       => Ave2_Y2(i+8),
                S       => Ave3_Y2(i)       );
end generate adder6;
--------------------------------------------------------------------------------end3.29
adder7: for i in 0 to 3 generate -- generates 36 bit data
   gives_Ave4_X1 : component c_addsub_7
   port map (   CLK     => MAIN_CLK,
                A       => Ave3_X1(i),
                B       => Ave3_X1(i+4),
                S       => Ave4_X1(i)       );
   ----------------------------
   gives_Ave4_Y1 : component c_addsub_7
   port map (   CLK     => MAIN_CLK,
                A       => Ave3_Y1(i),
                B       => Ave3_Y1(i+4),
                S       => Ave4_Y1(i)       );
   ----------------------------
   gives_Ave4_X2 : component c_addsub_7
   port map (   CLK     => MAIN_CLK,
                A       => Ave3_X2(i),
                B       => Ave3_X2(i+4),
                S       => Ave4_X2(i)       );
   ----------------------------
   gives_Ave4_Y2 : component c_addsub_7
   port map (   CLK     => MAIN_CLK,
                A       => Ave3_Y2(i),
                B       => Ave3_Y2(i+4),
                    S        => Ave4_Y2(i)      );
end generate adder7;
--------------------------------------------------------------------------------end3.30
adder8: for i in 0 to 1 generate -- generates 37 bit data
   gives_Ave5_X1 : component c_addsub_8
   port map (   CLK    => MAIN_CLK,
                A        => Ave4_X1(i),
                B        => Ave4_X1(i+2),
                S        => Ave5_X1(i)      );
   ----------------------------
   gives_Ave5_Y1 : component c_addsub_8
   port map (   CLK    => MAIN_CLK,
                A        => Ave4_Y1(i),
                B        => Ave4_Y1(i+2),
                S        => Ave5_Y1(i)      );
   ----------------------------
   gives_Ave5_X2 : component c_addsub_8
   port map (   CLK    => MAIN_CLK,
                A        => Ave4_X2(i),
                B        => Ave4_X2(i+2),
                S        => Ave5_X2(i)      );
   ----------------------------
   gives_Ave5_Y2 : component c_addsub_8
   port map (   CLK    => MAIN_CLK,
                A        => Ave4_Y2(i),
                B        => Ave4_Y2(i+2),
                S        => Ave5_Y2(i)      );
end generate adder8;
--------------------------------------------------------------------------------end3.31
gives_Ave6_X1_0 : component c_addsub_9 -- generates 38 bit data
port map (   CLK    => MAIN_CLK,
             A        => Ave5_X1(0),
             B        => Ave5_X1(1),
             S        => Ave6_X1(0)         );
----------------------------
gives_Ave6_Y1_0 : component c_addsub_9 -- generates 38 bit data
port map (   CLK    => MAIN_CLK,
             A        => Ave5_Y1(0),
             B        => Ave5_Y1(1),
             S        => Ave6_Y1(0)         );
----------------------------
gives_Ave6_X2_0 : component c_addsub_9 -- generates 38 bit data
port map (   CLK    => MAIN_CLK,
             A        => Ave5_X2(0),
             B        => Ave5_X2(1),
             S        => Ave6_X2(0)         );
----------------------------
gives_Ave6_Y2_0 : component c_addsub_9 -- generates 38 bit data
port map (   CLK    => MAIN_CLK,
             A        => Ave5_Y2(0),
             B        => Ave5_Y2(1),
             S        => Ave6_Y2(0)         );
--------------------------------------------------------------------------------end3.32
---- Divide to ((half the # of samples)*(# of averaging)*(LUT amp.)) ----- 3.33
-- Ave6_X1(0) is divided to 2^24 and saved into Ave_X1. Divide less to amplify it.
Ave6_X1(1) <= std_logic_vector(shift_right(signed(Ave6_X1(0)), 17));
```

```
Ave_X11     <= Ave6_X1(0)(37) & Ave6_X1(1)(14 downto 0);
givesAve_X1: component c_addsub_10
port map(    A        => Ave_X11,
             CLK      => MAIN_CLK,
             S        => Ave_X1            );
-- Ave6_Y1(0) is divided to 2^24 and saved into Ave_Y1. Divide less to amplify it.
Ave6_Y1(1) <= std_logic_vector(shift_right(signed(Ave6_Y1(0)), 17));
Ave_Y11     <= Ave6_Y1(0)(37) & Ave6_Y1(1)(14 downto 0);
givesAve_Y1: component c_addsub_10
port map(    A        => Ave_Y11,
             CLK      => MAIN_CLK,
             S        => Ave_Y1            );
Ave_XY1     <= Ave_X1(15 downto 0) & Ave_Y1(15 downto 0);
----------------------------
-- Ave6_X2(0) is divided to 2^24 and saved into Ave_X2. Divide less to amplify it.
Ave6_X2(1) <= std_logic_vector(shift_right(signed(Ave6_X2(0)), 17));
Ave_X2      <= Ave6_X2(0)(37) & Ave6_X2(1)(14 downto 0);
-- Ave6_Y2(0) is divided to 2^24 and saved into Ave_Y2. Divide less to amplify it.
Ave6_Y2(1) <= std_logic_vector(shift_right(signed(Ave6_Y2(0)), 17));
Ave_Y2      <= Ave6_Y2(0)(37) & Ave6_Y2(1)(14 downto 0);
Ave_XY2     <= Ave_X2(15 downto 0) & Ave_Y2(15 downto 0);
------------------------------------------------------------------------------------end3.33
gives_Ave_PHI1 :component cordic_1
port map (   aclk                      => MAIN_CLK,
             s_axis_cartesian_tvalid   => D_In_Ready,
             s_axis_cartesian_tdata    => Ave_XY1,
             m_axis_dout_tvalid        => open,
             m_axis_dout_tdata         => Ave_PHI1       );
----------------------------
gives_Ave_PHI2 :component cordic_1
port map (   aclk                      => MAIN_CLK,
             s_axis_cartesian_tvalid   => D_In_Ready,
             s_axis_cartesian_tdata    => Ave_XY2,
             m_axis_dout_tvalid        => open,
             m_axis_dout_tdata         => Ave_PHI2       );
------------------------------------------------------------------------------------end3.34
gives_sqr_Ave_X1 : component mult_gen_1
port map (   CLK      => MAIN_CLK,
             A        => Ave_X1,
             B        => Ave_X1,
             P        => sqr_Ave_X1                      );
----------------------------
gives_sqr_Ave_Y1 : component mult_gen_1
port map (   CLK      => MAIN_CLK,
             A        => Ave_Y1,
             B        => Ave_Y1,
             P        => sqr_Ave_Y1                      );
----------------------------
gives_sqr_Ave_X2 : component mult_gen_1
port map (   CLK      => MAIN_CLK,
             A        => Ave_X2,
             B        => Ave_X2,
             P        => sqr_Ave_X2                      );
----------------------------
gives_sqr_Ave_Y2 : component mult_gen_1
port map (   CLK      => MAIN_CLK,
                A       => Ave_Y2,
                B       => Ave_Y2,
                P       => sqr_Ave_Y2                        );
-------------------------------------------------------------------------------end3.35
-- calculate sqr_Ave_R1= sqr_Ave_X1+sqr_Ave_Y1
gives_sqr_Ave_R1 : component c_addsub_4
port map (      CLK     => MAIN_CLK,
                A       => sqr_Ave_X1,
                B       => sqr_Ave_Y1,
                S       => sqr_Ave_R1                        );
sqr_Ave_R1_2 <= "0000000" & sqr_Ave_R1(32 downto 0);
---------------------------------------------------
-- calculate sqr_Ave_R2= sqr_Ave_X2+sqr_Ave_Y2
gives_sqr_Ave_R2 : component c_addsub_4
port map (      CLK     => MAIN_CLK,
                A       => sqr_Ave_X2,
                B       => sqr_Ave_Y2,
                S       => sqr_Ave_R2                        );
sqr_Ave_R2_2 <= "0000000" & sqr_Ave_R2(32 downto 0);
-------------------------------------------------------------------------------end3.36
-- calculate Ave_R1= sqrt(sqr_Ave_R1)
gives_Ave_R1 :component cordic_0
port map (      aclk                        => MAIN_CLK,
                s_axis_cartesian_tvalid     => D_In_Ready,
                s_axis_cartesian_tdata      => sqr_Ave_R1_2,
                m_axis_dout_tvalid          => D_out_Ready,
                m_axis_dout_tdata           => Ave_R1        );
---------------------------------------------------
-- calculate Ave_R2= sqrt(sqr_Ave_R2)
gives_Ave_R2 :component cordic_0
port map (      aclk                        => MAIN_CLK,
                s_axis_cartesian_tvalid     => D_In_Ready,
                s_axis_cartesian_tdata      => sqr_Ave_R2_2,
                m_axis_dout_tvalid          => open,
                m_axis_dout_tdata           => Ave_R2        );
-------------------------------------------------------------------------------end3.37
-- take 10 MHz external clock and generate sync 100, 200, and 10 (MHz)
Clock : clk_wiz_0
port map (      clk_out1        => CLK_100MHz,
                clk_out2        => MAIN_CLK,
                clk_out3        => Photo_CLK,
                reset           => CLK_Reset,
                locked          => MMCM_Locked,
                clk_in1         => CLK                       );
-------------------------------------------------------------------------------end3.38
-- show results in 7 segment LEDs
segment1: DigitToSeg
port map(       in1     => dig0,
                in2     => dig1,
                in3     => dig2,
                in4     => dig3,
                in5     => dig4,
                in6     => dig5,
                in7     => dig6,
                in8     => "0000",
                mclk    => MAIN_CLK,
          an       => an,
          dp       => open,
          seg      => seg                        );
dp <= '1';
------------------------------------------------------------------------------------end3.39
-- Read digital inputs
process(MAIN_CLK, MMCM_Locked)
begin
    if rising_edge(MAIN_CLK) then
      if (CLK_Reset = '1') then
        counter1  <= 0;
        counter2  <= 0;
        counter3  <= 0;
      end if;
      counter1    <= counter1 + 1;
      counter2    <= counter2 + 1;
      counter3    <= counter3 + 1;

      -- generate 10 MHz clock from 200 MHz MAIN_CLK
      if (counter1 = 82) then
        counter1                <= 63;
        probe_0(11 downto 0)    <= D(11 downto 0);
        data_1(0)(11 downto 0)  <= D(11 downto 0);
        Ave1_X1(0)              <= X1_6(0);
        Ave1_Y1(0)              <= Y1_6(0);
        Ave1_X2(0)              <= X2_6(0);
        Ave1_Y2(0)              <= Y2_6(0);
        C1(0)                   <= C1(31);
        S1(0)                   <= S1(31);
        C2(0)                   <= C2(31);
        S2(0)                   <= S2(31);
        My_loop0: for i in 31 downto 1 loop
          data_1(i)      <= data_1(i-1);
          C1(i)          <= C1(i-1);
          S1(i)          <= S1(i-1);
          Ave1_X1(i)     <= Ave1_X1(i-1);
          Ave1_Y1(i)     <= Ave1_Y1(i-1);
          C2(i)          <= C2(i-1);
          S2(i)          <= S2(i-1);
          Ave1_X2(i)     <= Ave1_X2(i-1);
          Ave1_Y2(i)     <= Ave1_Y2(i-1);
        end loop My_loop0;
      end if;

      if (counter2 = 113) then
        counter2      <= 94;
        D_In_Ready    <= '1';
        probe_1       <= "1";
      else
        D_In_Ready    <= '0';
        probe_1       <= "0";
      end if;

      if (D_out_Ready = '1') then
        probe_2       <= "1";
      else
```

```
            probe_2          <= "0";
        end if;

        -- Send data to LabVIEW @ 5 MHz speed based on a 200 MHz clock
        if (counter3 > 127 and counter3 < 138) then
            Dec_CLK    <= '1';
            D_Out      <= Ave_R1 (15 downto 0);
            probe_4    <= Ave_R1 (15 downto 0);
            probe_3    <= "1";
        elsif (counter3 > 137 and counter2 < 148) then
            Dec_CLK    <= '0';
            D_Out      <= Ave_R2 (15 downto 0);
            probe_4    <= Ave_R2 (15 downto 0);
            probe_3    <= "0";
            if (counter3 = 147) then
                counter3  <= 128;
            end if;
        end if;
    end if;
end process;
-----------------------------------------------------------------------------------end3.40
-- binary to decimal conversion
reset_bcd <= NOT MMCM_Locked;
gives_digis : component bin2bcd
port map (  mclk         => MAIN_CLK,
            reset        => reset_bcd,
            binary_in    => Bin_Data,
            bcd0         => dig0    ,
            bcd1         => dig1    ,
            bcd2         => dig2    ,
            bcd3         => dig3    ,
            bcd4         => dig4    ,
            bcd5         => dig5    ,
            bcd6         => dig6            );
-----------------
process(MAIN_CLK)
begin
    if rising_edge(MAIN_CLK) then --Show data_1(0) in LEDs and 7 Segments
        count <= count + 1;
        if (count = 999) then
            count <= 0;
            count2 <= count2+1;
                if(count2 = 4899)then
                    count2       <= 0;
                    Bin_Data     <= data_1(0)(11) & "0000" & data_1(0)(10 downto 0);
                    LED          <= data_1(0)(11) & "0000" & data_1(0)(10 downto 0);
                end if;
        end if;
    end if;
end process;
-----------------------------------------------------------------------------------end3-42
-- ILA core for probing the processes
ILA: ila_0
port map (  clk        => MAIN_CLK,
            probe0     => probe_0,
            probe1     => probe_1,
                  probe2     => probe_2,
                  probe3     => probe_3,
                  probe4     => probe_4          );
    ----------------------------------------------------------------------------------------end3-43
end Behavior;
```

---end3