HARDWARE/SOFTWARE DESIGN AND IMPLEMENTATION OF A LASER SCANNING CONFOCAL MICROSCOPE CONTROLLER USING OPEN DESIGN APPROACH

BARAN YALÇIN

KOC UNIVERSITY
SEPTEMBER 2015

Hardware/Software Design and Implementation of a Laser Scanning Confocal Microscope Controller Using Open Design Approach

by

Baran Yalçın

A Thesis Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science
in
Optoelectronics and Photonics Engineering

Koc University

September 2015

Koc University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Baran Yalçın

and have found that it is complete and satisfactory in all respects,

and that any and all revisions required by the final

examining committee have been made.

Committee Members:

____

Alper Kiraz, Ph. D. (Advisor)

____

Halil Bayraktar, Ph. D.

____

Alexandr Jonas, Ph. D.

Date: ____________________

In dedication to my mother

for making me who I am,

and İpek for supporting

me all the way!

ABSTRACT

Laser scanning confocal microscope (LSCM) is a powerful electro-optic instrument in biological imaging and material science, compared to traditional wide-field microscopy methods. This stems from the fact that confocal microscopy enables optical imaging with a better spatial resolution. However, high cost and complexity of commercially available confocal systems hinder their wider usage.

In this thesis, a new LSCM hardware/software is studied and developed using an open design approach. This device is made more convenient by adapting low cost design techniques, widely adapted components and well supported open source software. For this reason, a CDAQ (Control and Data Acquisition) unit is designed and implemented to perform essential electrical input/output (I/O) operations. This unit consists of a 2-channel 16-bit ultralow glitch Digital to Analog Converter (DAC) for driving the galvanometer scanning mirrors, and interference reduction circuitry for a 3-channel on-board 12-bit Analog to Digital Converter (ADC) which is used for sampling photomultiplier tube and motor position feedback signals. These I/O peripherals are controlled by an ARM Cortex-M based microcontroller which runs a multithreaded firmware to accomplish given tasks. Components of the CDAQ are all pluggable I/O interfaces and they are open to modification for the desired application. Along with these operations, CDAQ provides power supply and gain control units.

CDAQ unit is controlled as a device from the user interface named “Konfokal” which is developed using Python programming language along with PyQt framework. “Konfokal” is an open source software which aims at increasing the productivity of researchers by providing environment where LSCM image acquisition and modification can be done under one single program.

Also, the opportunity to tailor or modify the source for the application is possible. Capabilities of this complete working device are shown with acquired images of test samples and it is offered as a more convenient and modern device for use of researchers.

"Konfokal" program was also extended in order to incorporate a Digital Micromirror Device (DMD) as a scanning unit in place of the galvanometer scanning mirrors.

ÖZET

Laser Taramalı Konfokal Mikroskobu (LTKM) biyolojik görüntüleme, malzeme bilimlerinde geniş açılı mikroskobu metodlarının yanı sıra kullanılan güçlü bir elektro-optik alettir. Konfokal mikroskobinin tercih edilme sebebi sağladığı daha iyi çözünürlükten kaynaklanmaktadır. Ticari olarak halihazırda bulunan konfokal sistemler karmaşık ve yüksek maliyetlidir. Ayrıca kullanım alanlarının genişletilmesi sınırlandırılmıştır.

Bu tezde yeni bir LTKM donanımı/yazılımı üzerinde çalışılmış ve açık tasarım ilkesi güdülerek geliştirilmiştir. Düşük maliyetli tasarım teknikleri, geniş kitleler tarafından kullanılan malzemeler ve kullanıcılar tarafından desteklenen yazılımlar sayesinde aletin kullanılabilirliği arttırılmıştır. Bu sebeple elektriksel girdi çıktı (I/O) işlemlerini gerçekleştirecek bir kontrol ünitesi (CDAQ) geliştirilmiştir. Bu ünite galvanometrik taramalı aynalarının sürmek için 2-kanallı 16-bit düşük gürültülü DAClerden, karışma (interference) engelleyici devreden ve motor/PMT bilgisini örneklemek için 3-kanallı 12-bit ADClerden oluşmaktadır. Bu girdi çıktı çevre aygıtları üzerinde çoklu kullanımlı bir bellenim koşan ARM Cortex-M tabanlı bir mikrodenetleyici tarafından kontrol edilmektedir. CDAQ'nun bileşenleri takılıp çıkartılabilir ve böylece istenilen uygulama için uyarlanmaya müsaittir. Bunların yanı sıra CDAQ güç kaynağı ve kazanç kontrol ünitelerine sahiptir.

CDAQ ünitesi Python programlama dili ve PyQt kütüphanesi kullanılarak geliştirilmiş bir kullanıcı arayüzü olan Konfokal programından kontrol edilebilmektedir. Konfokal açık kaynak kodlu bir yazılımdır. Konfokal, araştırmacıların LTKM görüntü aktarımını ve görüntü dönüşümlerini aynı yerde gerçekleştirmesine olanak sunarak üretkenliklerini arttırmayı amaçlamaktadır. Programın ihtiyaç duyulan yerlerinde değişiklikler yapılarak, diğer uygulamalara adapte edilebilir. Bu aletlerin yapabilecekleri, elde edilmiş olan görüntüleme sonuçlarıyla birlikte bu tezde belirtilmiş ve araştırmacılara daha kullanışlı bir platform olarak önerilmiştir.

Konfokal programı aynı zamanda Sayısal Mikroayna Aygıtı (DMD) ile birlikte kullanılabilecek şekilde genişletilmiştir.

ACKNOWLEDGEMENTS

I thank Prof. Dr. Alper Kiraz for his assistance and guidance during my thesis. His broad and deep expertise has always been very educational and instructive on me. Chance of working with him and his group was a life changing experience.

I also thank Adnan Kurt for his mentorship and help on the development and implementation of the hardware/software. He was always very helpful and provided assistance when needed. I have always admired his problem solving skills.

I thank to Professors Alexandr Jonas and Halil Bayraktar for joining my thesis committee.

Selçuk Çakmak has always provided different and interesting insights about hardware and software. I thank Mustafa Eryürek, İsmail Yorulmaz and Abdullah Muti, Gökalp Rençber for their help on almost about anything, they are great mentors and friends.

Last but not least, I thank to TÜBİTAK for funding 113F172 "OptoSayısal Konfokal Mikroskop Geliştirilmesi" project.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOMENCLATURE

LSCM	Laser scanning confocal microscopy
NA	Numerical aperture
r_{xy}	Lateral resolution
f_{eff}	Effective focal length
n	Refractive index
PMT	Photomultiplier tube
CCD	Charge coupled device
CMOS	Complementary metal oxide semiconductor
ADC	Analog to digital converter
DAC	Digital to analog converter
PWM	Pulse width modulation
CDAQ	Control and data acquisition unit
τ_{PWM}	PWM period
f_{PWM}	PWM frequency
τ	Time constant
MCU	Microcontroller unit
MSPS	Megasample per second
USB	Universal serial bus
UART	Universal asynchronous receiver/transmitter

Chapter 1

INTRODUCTION

1.1 Fundamentals of Microscopy

Human eye has limits. It is not capable of visualizing every object identically. Limits of vision exist as a consequence of conditions that are extrinsic and intrinsic to the human eye. Some of such extrinsic conditions are related to dimensions of the object, object's distance to the retina, color composition of the object, illumination from the object and also some of such intrinsic ones are related to resolution ability of the eye, power gathering capability and sensitivity of eye to light color [1]. Therefore, a correction is necessary for having a better vision and this correction is carried by optical instruments.

Microscopes are optical instruments that make small objects visible. Assuming an accepted proximity (d = 25 cm) between the object and the eye, the term 'small' can be defined by the angular resolution limit of the eye. This limit is 4 arcminutes, or in another unit, 0.07 degrees [2]. Calculation of this limit corresponds to ~ 1.75 cm (25 cm * 0.07). Beyond this limit, assuming other criteria are being satisfied, human eye is not capable of performing a proper visualization. A cell is an example for objects which human eye is not capable of visualizing.

Microscope notion usually connotes the light microscope, for that has been the oldest and the most common one. However, today's microscopes come in different varieties, named by the techniques and physical principles that are being employed for imaging. Other than fluorescence and light microscopes, electron microscopes and scanning probe microscopes are among the most significant ones. Due to the nature of this thesis, only fluorescence and light microscopy will be investigated throughout this section.

Fluorescence and light microscopes are used for producing magnified images of small objects with the help of light source, lenses and optionally with other optical elements such as mirrors, filters etc. Basically, two convex lenses and a light source are enough for magnifying the desired object. Figure 1.1 depicts the configuration and ray diagram of a simple microscope setup.

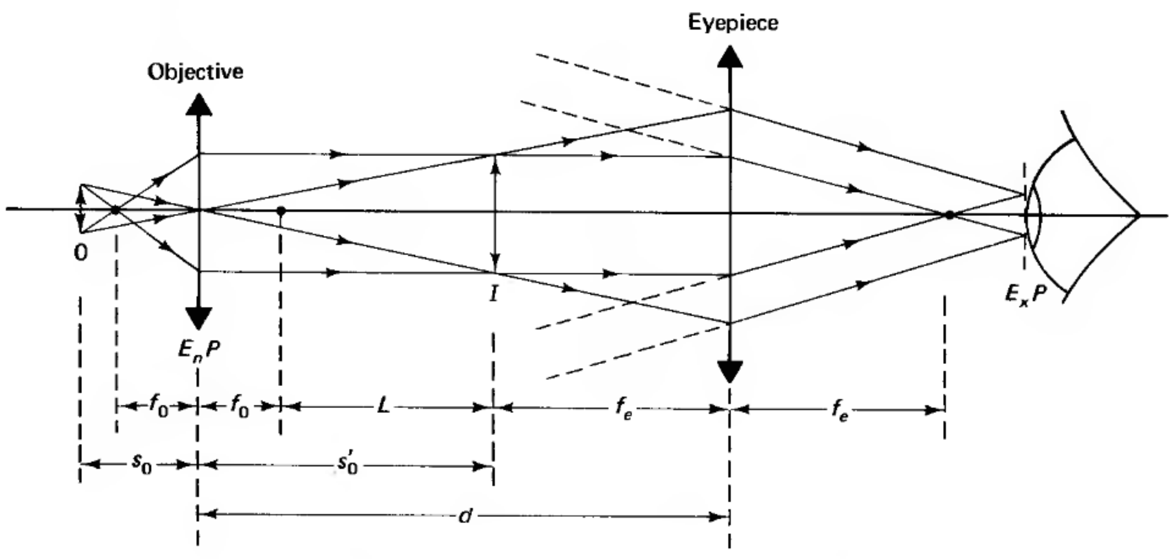


Figure 1.1 A rudimentary microscope setup [3]

The compound microscope in Figure 1.1 consists of two lenses. These lenses are enclosed in a tube. The one which is close to the object O is named as 'objective', and similarly the one which is close to the eye is named as 'eyepiece'.

Although not shown in Figure 1.1, a light source should be present to illuminate the object. The light source is usually a bulb located on left of the object (O) in Figure 1.1. This illumination configuration, which is widely adapted also for opaque samples, is known as Köhler illumination. Important thing to note here is this: Köhler illumination can be used for transparent and opaque samples where the light is condensed with the help of a dedicated condenser lens and auxiliary lenses for creating optimal distribution of light on the sample plane.

Here, the main role of the objective is to form a real intermediate image of sample (I) which is located outside of f_o at $s`_o$ of Figure 1.1. Real image is usually located somewhere between the eyepiece and $f_e$ (at d - $s`_o$) of Figure 1.1 and it is observed by the eyepiece. By looking at this real intermediate image – not the sample itself – eyepiece forms an inverted virtual image and this is the image that is perceived by the eye itself.

Similar to magnifiers, magnification of a microscope is expressed via (all lengths given in cm),

$$M = \frac{25}{f_{eff}}$$

where M is the total angular magnification and f_{eff} is the effective focal length of the system of lenses, which are separated by the distance d, as seen in Figure 1.1 [3]. Here, f_{eff} is given by

$$\frac{1}{f_{eff}} = \frac{1}{f_o} + \frac{1}{f_e} - \frac{d}{f_o f_e}$$

After further elaborations and applying Newton's thin lens magnification formula, the resulting master equation for this rudimentary microscope becomes

$$M = -\left(\frac{25}{f_e}\right)\left(\frac{L}{f_o}\right)$$

An important thing to note here is the L. L is standardized as 16 cm in many microscopes [3].

However, this is true for older microscopes. Modern microscopes operate with infinite tube distance. In most of the cases, magnification is determined by the focal lengths of objective and the eyepiece.

Another very important concept about the objective is the numerical aperture (NA). The numerical aperture is defined as $NA = n\ sin\ \alpha$, where n is the refractive index of the medium and α is the entrance half-angle. Since the objective defines the entrance pupil of the system, its ability to collect rays from different angles (NA) – as much as possible – is very critical. It is critical, because this parameter is inversely proportional to lateral resolution (by NA), whereas inversely proportional to axial resolution or depth of field (by NA^2) [3].

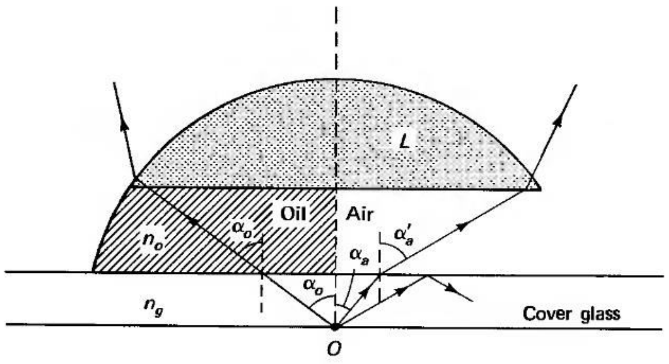


Figure 1.2 Oil-immersion objective [3]

Thus, this makes NA a very important parameter when selecting an objective lens. Increasing the NA is possible by using oil as shown in Figure 1.2. This type of objectives are known as ‘oil-immersion objectives’.

Considering these, objective becomes the most critical component in a microscope for it is directly related with resolution and depth of field. However, without taking other components and parameters into account, relying only on the quality of objective for better results is an oversimplification.

Microscope should be taken as a whole with the fact that: any enhancements made in one component may lead to performance decrease in overall system unless everything is calculated as a whole. Therefore microscope designer's approach should be to design everything in a perfect harmony while considering system as a whole. Every parameter (such as aberrations) must be calculated and considered from the initial to final step.

1.2 Fundamentals of Confocal Microscopy

Basics of optical microscopy that were mentioned above also hold their validities in confocal microscopy. Here, rather than illuminating the whole sample, sample is scanned by a beam, one point at a time. Light source used in this configuration is usually a coherent laser source. Since it was invented in 1957 by Marvin Minsky, it has evolved into four different techniques that are commercially available on the market [4]. These are known as: *Laser Scanning Confocal Microscopes, Spinning Disk (or Nipkow Disk) Confocal Microscopes, Dual Spinning Disk Confocal Microscopes* and lastly, *Programmable Array Microscopes*. In this section, only the *Laser Scanning Confocal Microscopes* and *Programmable Array Microscopes* will be investigated, for others are beyond the scope of this thesis.

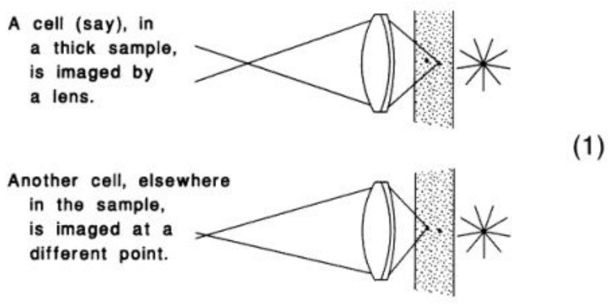

A cell (say), in a thick sample, is imaged by a lens.
Another cell, elsewhere in the sample, is imaged at a different point.
(1)

Figure 1.3 Wide field microscopy [5]

From a simplistic point of view, Figure 1.3 illustrates the fundamentals of wide field microscopy. Whole sample is uniformly illuminated with an extended light source such as a halogen lamp.

After passing through the sample, rays travel to the lens and get imaged by the two-dimensional detector (for example, a CCD camera). Since extended light source illuminates the whole sample, beams from all sample points reach to the detector and distort the quality of the image. To overcome this issue, a pinhole can be added to the system (see Figure 1.4). With the pinhole, undesired points within the sample do not contribute to the image; therefore, a higher image quality is obtained.

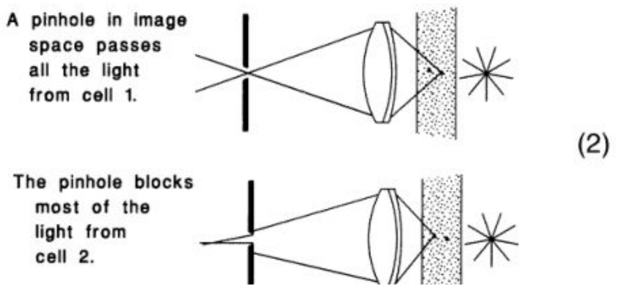

A pinhole in image space passes all the light from cell 1.
The pinhole blocks most of the light from cell 2.
(2)

Figure 1.4 Effect of a pinhole [5]

Point source and pinhole satisfy the condition for confocal microscopy. With a point source, only the selected point is illuminated in the sample and interferences with other points are blocked. In addition, with the help of pinhole, only light from the observed point is detected by a point detector. This is illustrated in Figure 1.5. The illuminated point is now optically conjugate to the pinhole, and since the word confocal is actually a combination of 'conjugate' and 'focal', the condition for confocal imaging becomes truly satisfied.

'Confocal' paradigm has advantages over wide field microscopy in terms of resolution and contrast. These advantages are due to point-by-point (or one at a time) imaging approach of this technique.

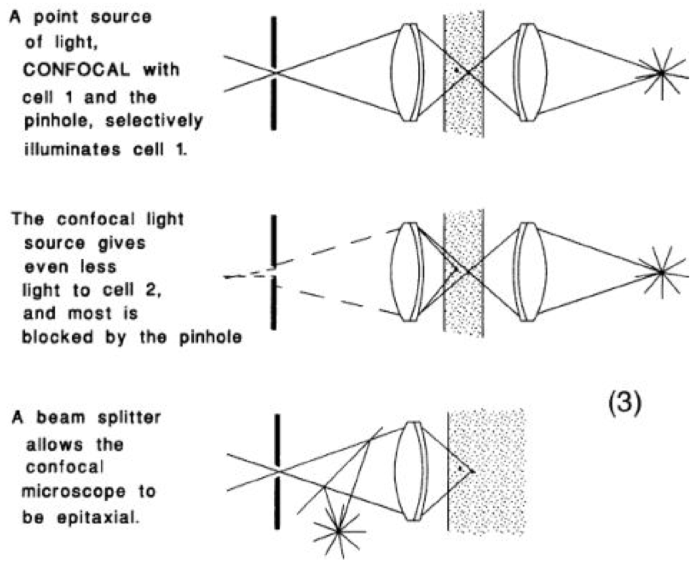

A point source of light, CONFOCAL with cell 1 and the pinhole, selectively illuminates cell 1.
The confocal light source gives even less light to cell 2, and most is blocked by the pinhole
A beam splitter allows the confocal microscope to be epitaxial.
(3)

Figure 1.5 Confocal microscopy [5]

Since only a single point is illuminated at a time, the interference by other points (in X, Y and Z axes) is minimal. Consequently, the lateral resolution, axial resolution and thus the contrast of the image increase significantly. This also makes confocal microscopy a 3D technique by enabling a scan through the volume of interest.

Lateral and axial resolution concepts are illustrated in Figure 1.6. These parameters are defined with respect to the optical axis of the microscope. Lateral resolution is the minimum distance between two distinguishable points that lie within the surface of the specimen –a plane perpendicular to optical axis- , and axial resolution is the minimum distance between two distinguishable points that lie within the depth of the specimen.

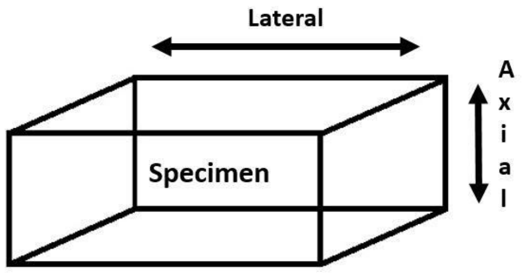


Figure 1.6 Lateral and axial resolution

For confocal microscopy lateral resolution is given by the following equation

$$r_{xy} = \frac{0.61\lambda}{\sqrt{2}NA}$$

Here r_{xy} is the lateral resolution, λ is the wavelength and NA is the numerical aperture of the objective lens [6]. Similarly the scanning angle is given by the following equation

$$\theta = \pm\tan^{-1}\frac{\frac{l}{2}}{f_{ep}}$$

Where θ is the scanning angle, l is the scanning length and f_{ep} is the focal length of the eyepiece [6].

By tilting and changing the angle of mirror, desired area is scanned. *Laser Scanning Confocal Microscopes* and *Programmable Array Microscopes* both base on this principle, the movement of mirrors. To scan in Z axis, the stage that holds the sample is incrementally moved in Z direction for every new XY scan.

Since only a single point is being imaged at a time, whole specimen should be scanned with the point source. Compared to wide field microscopy this results in longer acquisition time and longer exposure time. The tradeoff for 3D imaging is area of view.

Detection and acquisition of the light information in confocal microscopy is performed by various photodetectors, usually with Photo Multiplier Tube (PMT), Charge Coupled Device (CCD) Camera, Complementary Metal Oxide Semiconductor (CMOS) Camera, or photodiode [6]. While mirrors are performing a raster scan, light information should be detected by a light sensor and transferred to the unit where image formation and further processing is performed. This brings real time constraints into the definition of problem and as one can easily guess, trouble is doubled for a real time video streaming application. However with modern processors and peripherals such as Analog to Digital Converters (ADC) and Universal Serial Bus (USB) this data can easily be converted and transferred into a computer for further processing to be performed.

Confocal microscopy requires electrical hardware, firmware and software that runs on a high performance machine such as a personal computer. From hardware perspective, electronics modules for controlling mirrors and detecting the incoming light are needed. This should also be supported with a microcontroller which runs the firmware that performs scanning, recording and data transfers. Recorded data is transferred into a PC and received by the master program which is controlled by the researcher. Since techniques used for performing these operations are vast, only the *Laser Scanning Confocal Microscopes* will be detailed in the next section.

1.3 Current Methodologies in Confocal Microscopy

1.3.1 Laser Scanning Confocal Microscope

Laser Scanning Confocal Microscope (LSCM) was born owing to breakthroughs in laser technology. First adapters of this technique are Thomas and Christoph Cremer, where the specimen was scanned with a laser beam in a point-by-point approach [7]. A widely-adapted and known LSCM setup is given in Figure 1.7. LSCM is an electro-optic system and it contains three building blocks. As depicted in the Figure 1.7, these building blocks are:

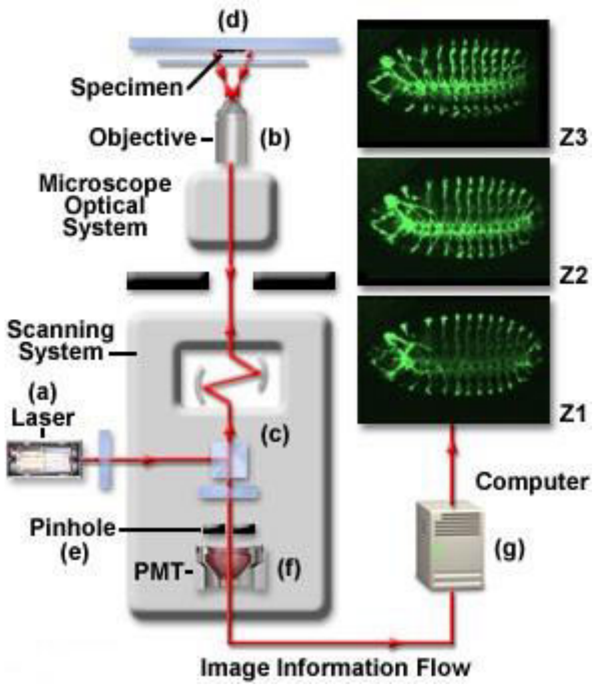


Figure 1.7 LCSM setup [8]

- Optical system: Lenses, mirrors, filters, beam splitters, laser
- Control and data acquisition unit: Galvanometer scanning mirrors, PMT, electronics, microcontroller
- PC: Software for controlling the LSCM and forming images

  Combined together, these subsystems form the integrated LSCM device. Elements of the optical system establish the optical pathways between light source, specimen and the sensor.

Though it is quite similar to the system adopted for conventional wide-field fluorescence microscopy, additional elements such as scan lens (also known as f-Theta lens) and beam splitter exist. Scan lens is an essential element for building LSCM, because it corrects the spot size of the beam at different scan angles so that the spot size in the plane of the specimen almost always stays constant. Dichroic beam splitter, which is another element used in LSCM, splits the incoming laser beam into two, hence the beam travels to specimen and into the PMT.

Control and data acquisition unit (CDAQ) is a crucial element which makes the LSCM possible. This unit is responsible for commanding galvanometer scanning mirrors and collecting the light information from the PMT. Along with these extremely important tasks, it also provides the necessary power and input/output lines of interest to other parts of the LSCM such as the power rails of galvanometer scanning mirrors controller, PMT gain voltage, microscope lamp and USB interface for connecting to the PC. CDAQ actually serves as a central bridge between these delicate instruments and the image. Therefore to grasp the vital functions of this unit one must have more knowledge about the PMT and galvanometer scanning mirrors first.

PMT is an instrument used for detecting light signal and converting it into electrical signal. Its operating principles are based on two physical effects which are known as photoelectric effect and secondary emission. PMT is composed of two stages and these two physical effects play role in these stages. In the first stage the light signal is translated into electrical signal, or in other sense, the information is translated from photon to electron. At this stage photoelectric effect takes place. In the second stage based on the secondary emission effect, this electrical signal (electrons) is amplified. These stages and the structure of the PMT are detailed in Figure 1.8.

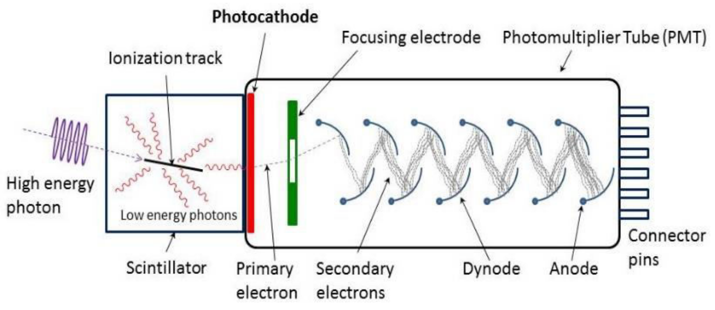


Figure 1.8 Structure of PMT [9]

PMT is an extremely sensitive device and therefore its components reside inside a sealed vacuum tube. First, the incoming light hits the photocathode, which actually is a thin conducting layer. This triggers the photoelectric effect and causes release of electrons from the other side. With the help of the focusing electrode, electron travels and hits to the first dynode. Dynodes have certain voltage applied to them which accelerates the electrons, thus increasing their kinetic energy. When the accelerated electrons hit the surface of the next dynode, secondary emission effect takes place and results in more electron release.

Every bounce on dynodes increases the electron count. After bouncing back and forth inside the dynode chain, electrons finally reach their final destination: anode. More dynodes mean more current or higher amplification. The amplification is actually a controllable parameter for PMT devices. To illustrate better, Figure 1.9 is provided below.

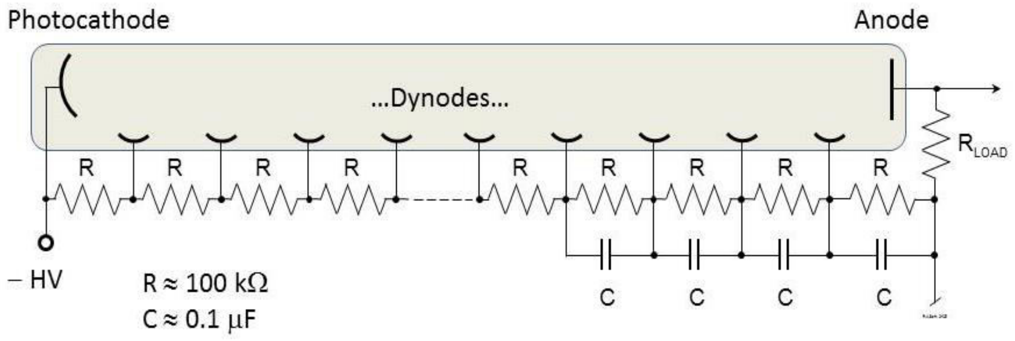


Figure 1.9 PMT gain circuit [10]

A negative voltage is applied to the photocathode and the last dynode is grounded after its terminating resistor. This negative voltage applied to the photocathode is usually called as “High Voltage Tube” or HV for short. Applied voltage is then divided into voltage steps, throughout the dynodes. Therefore every dynode has a higher potential than the previous dynode. Since bouncing of electrons between two dynodes lead to multiplication of electrons, this multiplication coefficient is actually determined by the potential applied to the photocathode. Adjusting HV increases or decreases the potential of dynodes accordingly and this causes increase or decrease in the current amplification process. PMT devices usually multiply on the order of thousands and actually they can reach up to 100 million times multiplication. This of course is determined by the parameters of the PMT device such as number of dynodes and applied HV.

Another essential element in LSCM is the galvanometer scanning mirror. Galvanometer scanning mirror is used for steering the beam. This mirror and its properties are very critical for performing a successful scan and also for obtaining a good quality image. Speed, stability, cost, reliability, size, precision, quality and performance are some of the criteria which determine the most suitable scanning mirror for the application. To meet these physical specifications, mirror is mounted on a DC motor which is controlled by a PD controller. A scan head is shown in Figure 1.10. Scan head can come with a built-in PD controller or an additional controller may be provided or built for the need. These controllers aim to translate applied voltage into tilt angle of the mirror. Combined with suitable waveform and resonant frequency, scanning mirror controllers offer best beam steering solution for LSCM.

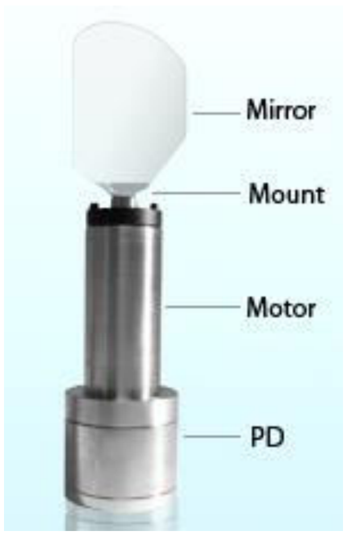


Figure 1.10 Scan head [11]

PMT and galvanometer scanning mirrors form the essential instruments while performing LSCM. Quality of these devices effect the overall quality of the system and image. Along with these instruments other optical elements and devices are required for an LSCM too. These devices are mentioned and described in the section where the optical setup is detailed.

1.4 Laser Scanning Confocal Microscope Setup

In this thesis an LSCM has been built. To build the microscope, optical and electrical parts have been combined. Although the electronics and software are the main focus for this thesis, optics will be investigated briefly. Figure 1.11 shows the setup's view from above.

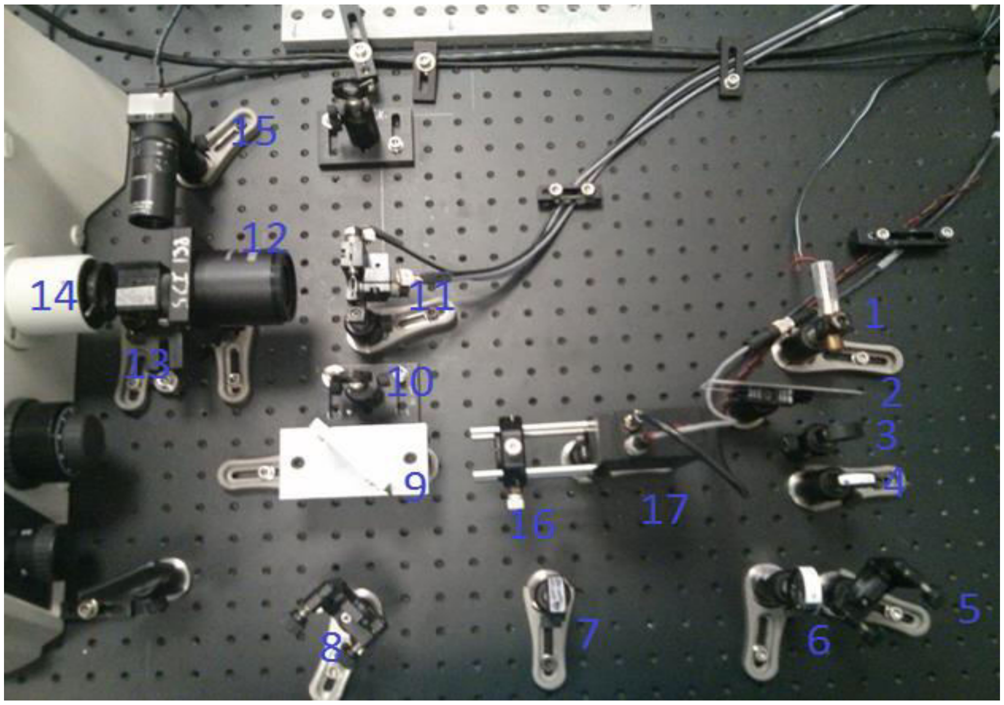


Figure 1.11 Optical components used in setup

Optical components in Figure 1.11 are numbered and annotated briefly below:

1- *Laser source:* 532 nm green laser light source. Its power capability is 5 mW and the beam quality is sufficient.
2- *Variable neutral density filter:* Adjusts the intensity of the laser beam.
3- *Intensity filter:* Reduces the intensity of the beam.
4- *Plano-convex lens:* Beam passes through this lens. This lens has f = 200 mm to provide better point illumination.
5- *Mirror:* Beam is reflected to next lens.
6- *Plano-convex lens:* Lens with f = 50 mm.
7- *Plano-convex lens:* Lens with f = 100 mm. Combination of lenses 6 and 7 forms a telescope and increases beam diameter two times.
8- *Mirror:* Beam is reflected from this mirror.
9- *Dichroic beam splitter:* Excitation laser beam passes through this beam splitter and then reaches to galvanometer scanning mirrors. Detection light coming from the microscope is reflected from this beam splitter towards the PMT.
10- *Diaphragm:* Regulates the amount of light that passes through the galvanometer scanning mirrors.
11- *Galvanometer scanning mirrors:* Used for steering the excitation beam in X and Y axes. Fast motion mirror provides motion of beam in X axis and slow motion mirror provides motion of beam in Y axis.
12- *Scan lens:* Special Optics telecentric f-Theta laser scan lens increases image area as much as 100 times of a conventional microscope. It is optimized for confocal microscopy, therefore it focuses incoming laser beam to a spot on the microscope slide. Its back focal length is 70 mm.
13- *Beam splitter:* Laser beam passes through the beam splitter and reaches into the microscope.

14- *Tube lens:* Resides inside Nikon Inverted Eclipse Ti-U microscope. Its focal length is 200 mm and it is used because of the infinity corrected objective. Then, beam enters the objective which is an infinity corrected Nikon 40X objective with working distance of 2.1 mm and NA=0.55. Image is formed by putting the specimen at the focal plane of the objective.

15- *Camera:* Paraxial rays passing from the tube lens hit the beam splitter again and reach into the USB camera. This is useful for viewing the sample on PC.

16- *PMT lens:* Light coming from galvanometer scanning mirrors goes through this lens before reaching to PMT. Rays are focused to be detected by the PMT.

17- *PMT:* Detects the light and translates light signal into electrical signal.

Chapter 2

SYSTEM CONSIDERATIONS

2.1 Overview

LSCM is an electro-optic instrument and it needs certain hardware and software in order to function. Developing and maintaining an LSCM system requires one to meet the criteria listed below. Although these are vaguely defined for now, they will become clear at the end of this section. To summarize:

- Delivering necessary power and signal to components
- Controlling galvanometer mirrors
- Performing PMT readout
- Automating for everyday use

are the major requirements for an LSCM system which is aimed to be used by a researcher for everyday purposes.

In this section work completed for meeting these requirements is described. To understand these requirements, components and their specifications are given. Also, prototypes developed for these purposes are detailed. At the end of this section, the need for certain system parameters will become very easy to understand.

Hence, system requirements and specifications, which are derived from these prototypes, will become solid and valid. Therefore with these final results, we will be ready to make the final design.

2.2 Electro-optic Elements

To construct the LSCM setup, two major elements are needed: PMT and galvanometer scanning mirrors. Electrical hardware should form a bridge between these two pieces and work as a station. In order to produce such a system, details and working conditions of the PMT and galvanometer scanning mirrors are needed. Therefore their specifications must be considered during the design process.

2.2.1 PMT

PMT used in the setup is "PMM02 Amplified Photomultiplier" and it is manufactured by ThorLabs [12]. Figure 2.1 shows this product with its interconnection cables and power supply. General and electrical specifications of this PMT are detailed in Table 2.1.

Figure 2.1 PMM02 with accessories

Photocathode Type	Multialkali (S20)
Photocathode Geometry	Head-On
Dynode Chain Orientation	Circular
Photocathode Active Diameter	22 mm
Wavelength Range	280 – 850 nm
Gain (Max)	3.1×10^6
Peak Responsivity (Max)	67 mA/W
Quantum Efficiency at Peak (Typ)	21%
Transimpedance Gain	Hi-Z: 1×10^6 V/A 50 Ω: 5×10^5 V/A
Dark Current (@ 20 °C)	0.5 – 5 nA
Dark Count Rate (@ 20 °C)	3000 s^{-1}
Bandwidth (6dB)	0-20 kHz
Amplifier Noise (Typ)	2 mV RMS
Amplifier Offset (Typ)	1 mV
Output Rise and Fall Times	15 μs
Output Impedance	50 Ω
Output Signal	0-10 V (unterminated) 0-5 V (term. into 50 Ω)
Power Input	+12 V (12 to 15): 40 mA -12 V (-12 to -15): 10 mA
Anode Current (Max)	100 μA
Tube Voltage	0 to -1800 V
Tube Voltage Control	0 to 1.8 V
HV Control Connector	2.5 mm Mono Jack

HV Control Sensitivity	-1000 V/V
Warm Up Time	<10 s
Output Connector	SMA

Table 2.1 PMM02 Specifications

Design of the CDAQ unit must obey given PMT specifications. These are:

- Output signal should be carried by a 50 Ω coaxial cable so that the impedance is matched.
- Output signal varies between 0 – 10 V. Because it is unterminated to ground and only carried via cable.
- For tube voltage control (gain), an adjustable 0 – 1.25 V signal should be provided.
- Since PMT comes with its own power supply, another power supply unit is not needed.
- Figure 2.2 shows the spectral response with respect to wavelength. PMM02 has a good sensitivity for 532 nm wavelength. Therefore, a monochromatic 532 nm green laser was chosen as the laser source for the LSCM setup.

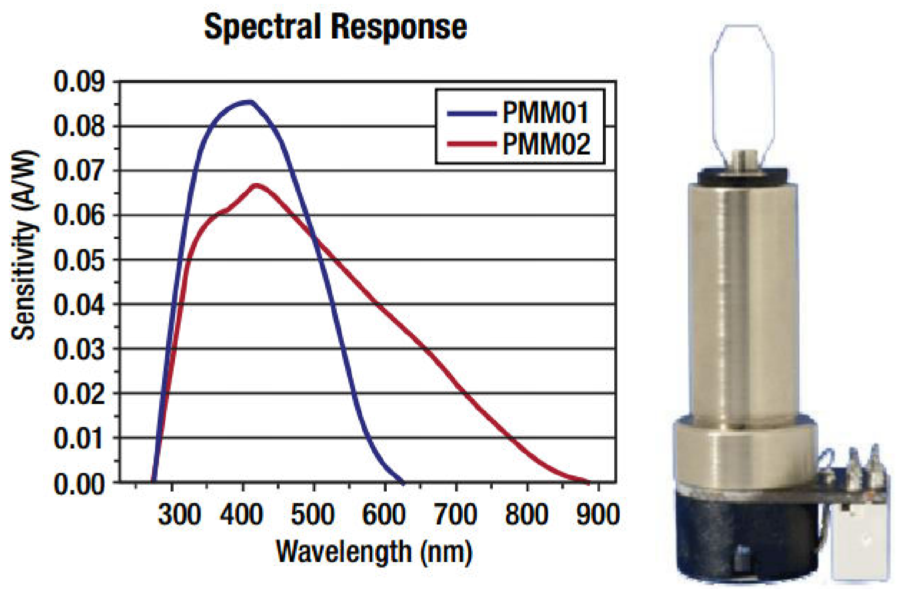


Figure 2.2 Spectral response of PMM02 (left), 6210H series optical scanner (right)

2.2.2 Galvanometer Scanning Mirrors

Galvanometer Scanning Mirrors used in this setup belong to 6210H Series of Cambridge Technology. A single scanner is shown in Figure 2.2. These scanners are used in applications where high-speed and accuracy is needed. Normally, electrical and mechanical properties of these mirrors would be very important and critical parameters. However, in this setup, these scanning mirrors are paired with MicroMax 673 Series Dual Axis Driver. This driver automatically handles the control of the mirrors and handles all scenarios for the end user. Table 2.2 gives the electrical specifications of this driver board.

Parameter	**Conditions**	**Input Configuration**	**Units**
Command Input Impedance	*Differential*	200	Kohm
Position Output Impedance	*Typ*	2	Kohm
Analog Input Range	*Max*	± 10	Volts
Position Offset Trimpot Range	*Typ*	± 0.5	Volt
Position Output Scale Factor	*Std*	0.500	Volts/° mech.
Fault Outputs	*Typ*	12V CMOS logic through 4.75 kohm	-
Temperature Stability	*0-50°C Ambient Temp*	20	ppm/°C
Absolute Maximum Supply Voltage	*± 18 to ± 30 Range*	30	VDC
	± 15 to ± 18 Range	30	
Minimum Operating Voltage	*± 18 to ± 30 Range*	18	VDC
	± 15 to ± 18 Range	15	

Output RMS Current	*Typ*	5	A
Output Peak Current	*Typ*	11.5	A
Output Peak Current	*Guaranteed*	7	A
Supply Current	*Without Scanner*	± 200	mA
Short Circuit Protection (Fuse)	*Typ*	Scanner RMS x1.25	A
Over-position Protection	*Typ*	Scanner Field size + 1°	Deg
Under-voltage Protection	*± 18 to ± 30 Range* *± 15 to ± 18 Range*	17 12.5	V
Over-Temperature/Over current Protection	*Scanner RMS*	1-3	Sec
Ambient Temperature Range	*Max limits*	0-50	°C

Table 2.2 MicroMax 673 Series Driver Specifications

Since CDAQ provides the necessary control signals for the driver, these specifications play a key role while designing the CDAQ. MicroMax 673 Series does not come with a power supply, therefore, it becomes CDAQ's responsibility to power this driver. To power the driver ± 15 V and 11.5 A are needed. To drive the mirrors, ± 10 V control signals are needed for X and Y input channels. These input signals, when multiplied with position output scale factor, can tilt mirrors up to ± 5 degrees. Applied control signals and other case dependent parameters were determined by testing the mirrors in imaging the surface of the sample. In the next section, these parameters will be determined while creating the first prototype.

2.3 Prototypes of CDAQ

To understand requirements and specifications of the final system, prototyping is performed. It is important to note that prototyping is a very important step in system-wide designs because it forewarns the designer about ill-considered system parameters and leads to a more solid design or end product. Therefore a prototype is designed to meet the requirements that are mentioned in the previous section. Main objective of the CDAQ is to control the mirrors and send PMT readings to PC for further processing. To meet with these objectives block diagram shown in Figure 2.3 is considered.

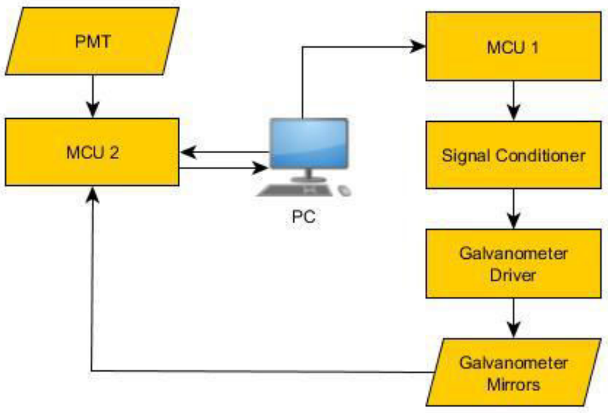


Figure 2.3 Block diagram of the prototype

According to this block diagram, design employs two microcontrollers. MCU 1 generates the scan waveform by passing its output to a signal conditioning circuit. This circuit takes the signal and turns it into a valid waveform to perform a periodic scan. In this prototype, sawtooth waveforms for driving the X and Y channels are generated by employing the PWM (Pulse Width Modulation) technique. Later these signals are fed into galvanometer driver. MCU 2 records PMT readings and galvanometer scanning mirror position

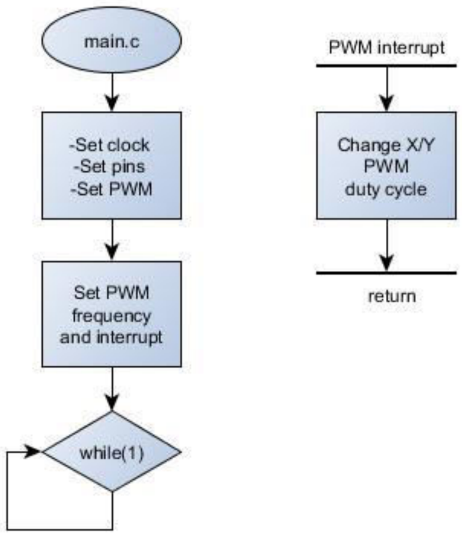


Figure 2.4 MCU 1 firmware flowchart

signals. These data are later transferred to PC in order to form the confocal image.

MCU 1 used in this design is a Texas Instruments Tiva C series microcontroller TM4C123G. The flowchart of the firmware that runs in this microcontroller is available in Figure 2.4. Firmware is also available in Appendix.

To generate analog waveforms for driving X and Y channels, PWM analog conversion technique is applied. This technique is chosen because of its simplicity and efficient power consumption. PWM signals are determined by two parameters, one of them is the period and the other one of them is the duty cycle. Duty cycle of a PWM signal is given by the following relation:

$$Duty\ cycle = \frac{\tau_{on}}{\tau} \text{ x } 100$$

Where τ is the period of signal and τ_{on} is the time where signal is logic high. Duty cycle can be between 0 – 100. Sample PWM signals with various duty cycles are shown in Figure 2.5. 50%, 75%, 25% duty cycles mean that signal is logic high for half of the period, 0.75 times of the period and quarter of the period respectively.

By varying and integrating the PWM signals, desired waveform is obtained. If variations of the duty cycle are constant, then ramp or sawtooth waveform is obtained. PWM DAC process is detailed in Nisarga's application report [13]. While scanning the sample with galvanometer scanning mirrors, two waveforms are generated. One of them should scan the sample slow (in Y axis) and one of them should scan the sample fast (in X axis). This is known as raster scan and it is illustrated in Figure 2.6.

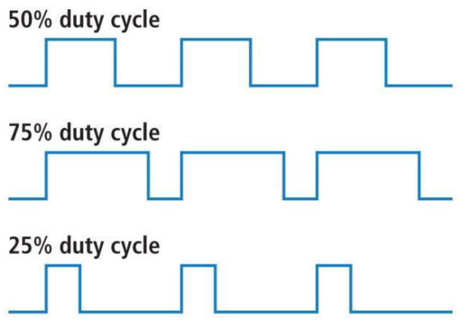


Figure 2.5 Different PWM duty cycles

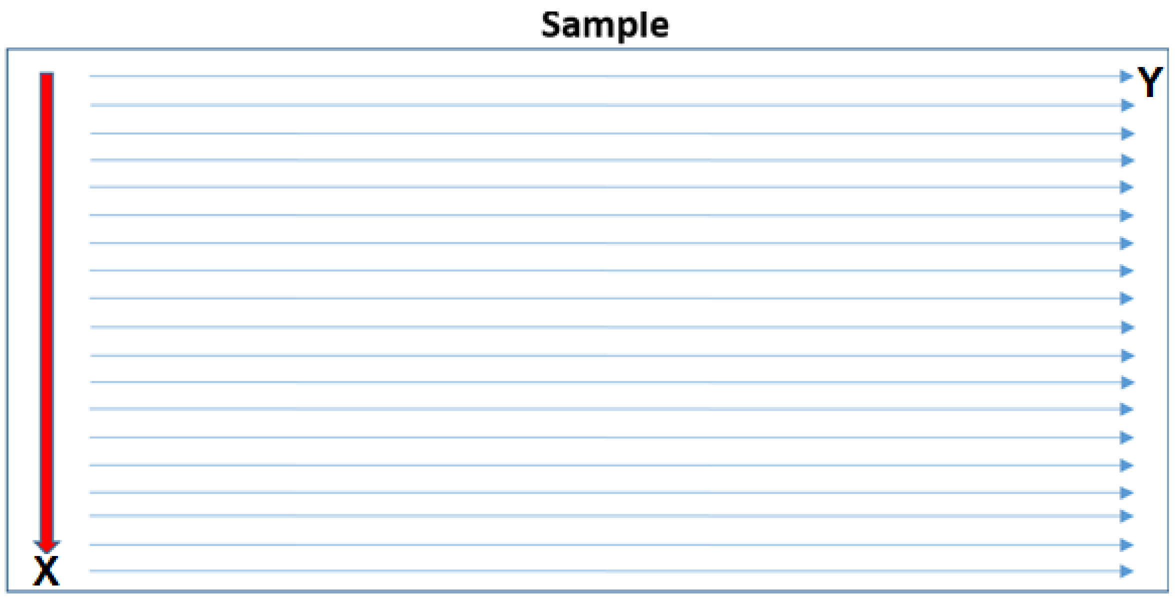


Figure 2.6 Raster scan, Y (slow signal), X (fast signal)

In his application report, Nisarga addresses DAC technique via PWM [13]. PWM DAC technique is chosen because it is simple to build, low cost and energy efficient. It basically maps phase modulations to voltage modulations. This is performed by passing the signal through a low pass filter. Similar technique is followed and frequencies of the signals are calculated. These parameters are shown in Table 2.3. As it is depicted in this table, ratio of X and Y scanning frequencies is approximately 1:100.

Table 2.3 PWM DAC signal parameters

Channel	Resolution	Repeat	τ_{PWM} [ms]	f_{PWM} [kHz]	$f_{sawtooth}$
X	100	100	1	1	0.1 Hz
Y	100	10	0.1	10	10 Hz

Figure 2.7 explains, *resolution, repeat,* τ_{PWM} and *period of signal* parameters of Table 2.3 below. Resolution is the step size of signal, which starts from V_{min} and increases to V_{max}. Repeat is the repeat count of signal that is waited before incrementing to next voltage level. In this context, resolution and repeat parameters are unitless.

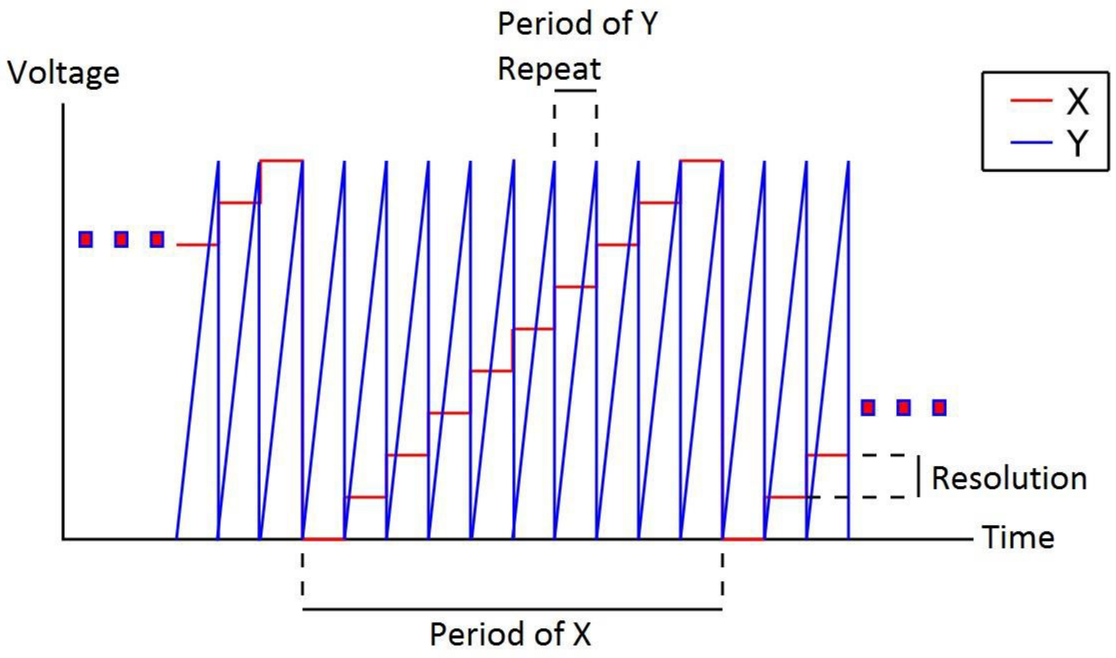


Figure 2.7 Illustration of PWM DAC parameters

Thus, these parameters cannot be selected independently from each other and must satisfy a condition. This condition is generated for this application with the help of Figure 2.7. X signal should wait for a single scan of Y. This time period is given by $X_{repeat} * X_{\tau_{PWM}}$.

This time period should match to the period of Y signal, which is given by $Y_{repeat} * Y_{\tau_{PWM}} * Y_{resolution}$. Overall condition is captured by the following relation:

$$X_{repeat} * X_{\tau_{PWM}} = Y_{repeat} * Y_{\tau_{PWM}} * Y_{resolution}$$

To generate the desired sawtooth waveforms, PWM signal is passed through a signal conditioning circuit. This circuit must integrate the given signal and also adjust the level of the signal. Therefore, a low pass filter with a level shifter and amplifier is constructed. Signal conditioner circuit is available in Figure 2.8.

It contains second order low pass filters which are used for eliminating the f_{PWM} and passing the real frequency of the signal $f_{sawtooth}$. As Nisarga mentions, time constants of the first and the second stage of these filters must be equal to each other for peak performance. Time constant is calculated via:

$$\tau = RC$$

For 0.1 Hz signal R and C values are 2.2 kΩ with 100 µF and 1 MΩ with 220 nF. For 10 Hz signal R and C values are 3.3 kΩ with 680 nF and 1 MΩ with 2 nF. Transfer characteristics of these filters are available in Figure 2.9 and Figure 2.10. These plots determine how these filters respond, and how well-suited they are for the desired application. Bode plots of these filters prove that these filters block the PWM frequency and pass the sawtooth waveform frequency.

Using the circuit presented in Figure 2.8, desired sawtooth waveforms were obtained and they are ready to control galvanometer scanning mirrors. 10 Hz sawtooth waveform was measured and captured in the oscilloscope screen. Waveform image proves the functionality of the signal conditioner circuit and it is available in Figure 2.11.

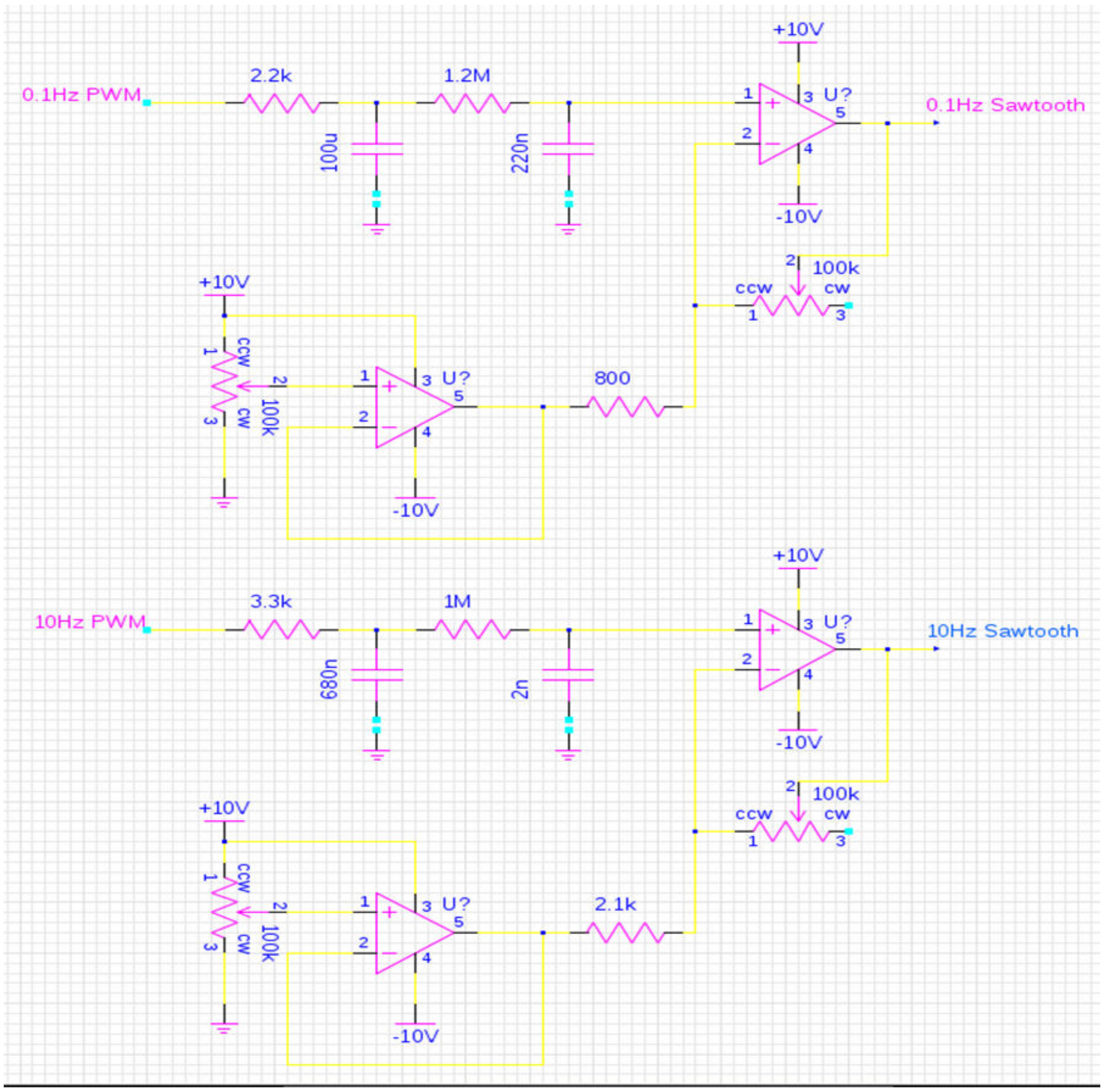


Figure 2.8 Signal conditioning circuit

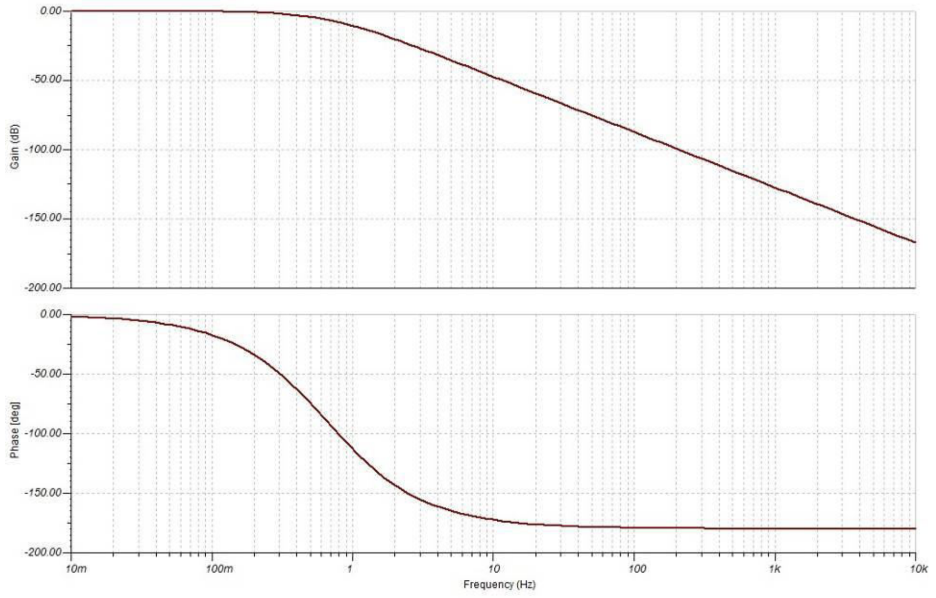


Figure 2.9 Bode plot of X channel filter

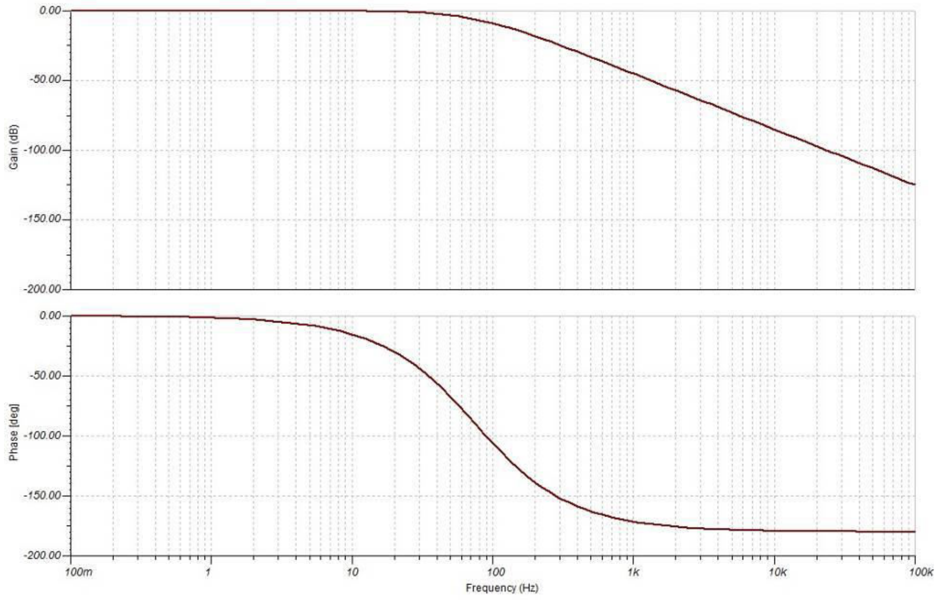


Figure 2.10 Bode plot of Y channel filter

MCU 2 used in this design is Texas Instruments Stellaris series microcontroller LM4F120. MCU 1 and MCU 2 are microcontrollers that have ARM Cortex-M family processors and they are widely adapted in data acquisition and control applications. The firmware written for this microcontroller has the flowchart that is available in Figure 2.12. Firmware is also available in Appendix. MCU 2 samples the PMT and motor position readings, then transfers them to PC. For this task, differential sampling technique is used at 1 MSPS (megasample/s). Recorded data are transferred to PC via serial UART (Universal Asynchronous Receiver/Transmitter) protocol.

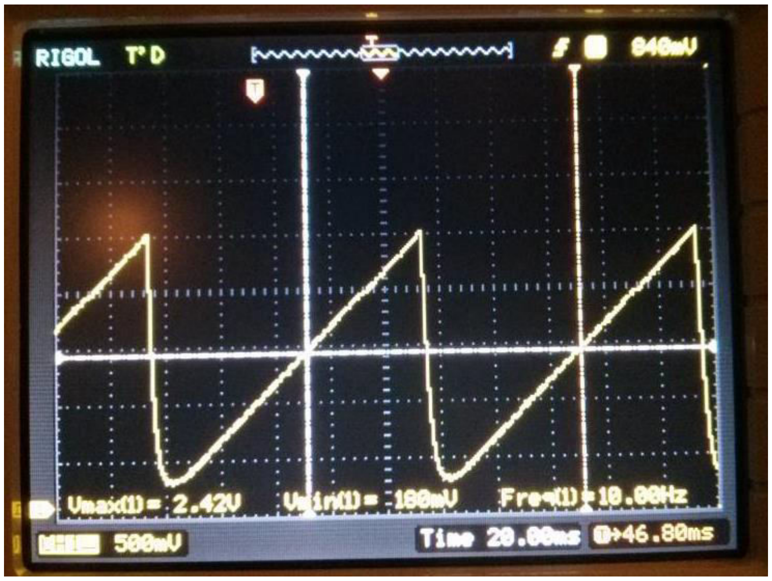


Figure 2.11 Waveform generated with PWM DAC technique

All procedures are controlled by the Python script which is running on the PC. When this program begins, mirrors

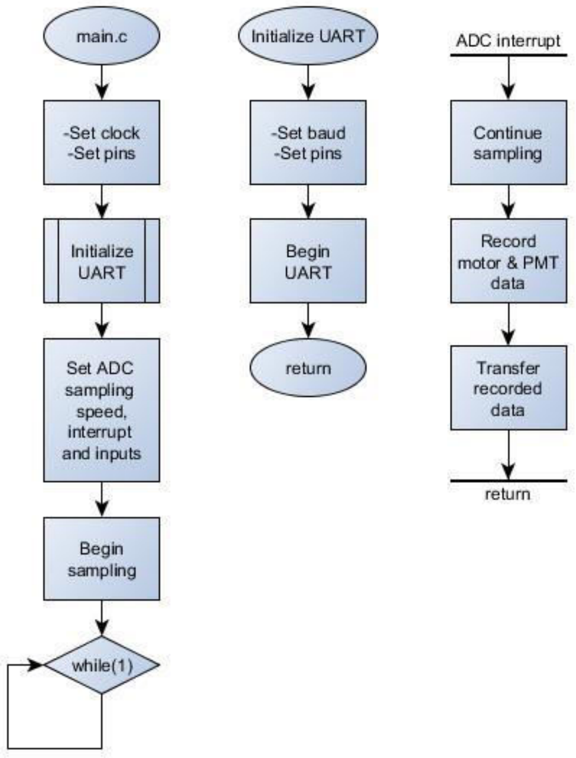


Figure 2.12 MCU 2 firmware flowchart

start to move and sampling is performed. After whole scan completes, a MATLAB routine forms the image. To perform these tasks, pyserial and matplotlib libraries are used besides Python standard library and MATLAB. Flowchart of these Python and MATLAB scripts are available in Figure 2.13. These scripts are also available in Appendix.

Forming the image requires PMT and galvanometer scanner position data. 12-bit ADC of Stellaris microcontroller, samples these data and transfers them to PC. Figure 2.14 shows the captured waveforms of PMT and galvanometer scanner position. As expected, slow and fast saw tooth waveforms are recorded. X axis motor is driven by slow signal, and Y axis motor is driven by fast signal. Y signal waits for one scan of X and then increases its value. Interpolation of these two data according to MATLAB routine of Figure 2.13 displays the final image shown in Figure 2.15. This image consists of 100 x 100 pixels.

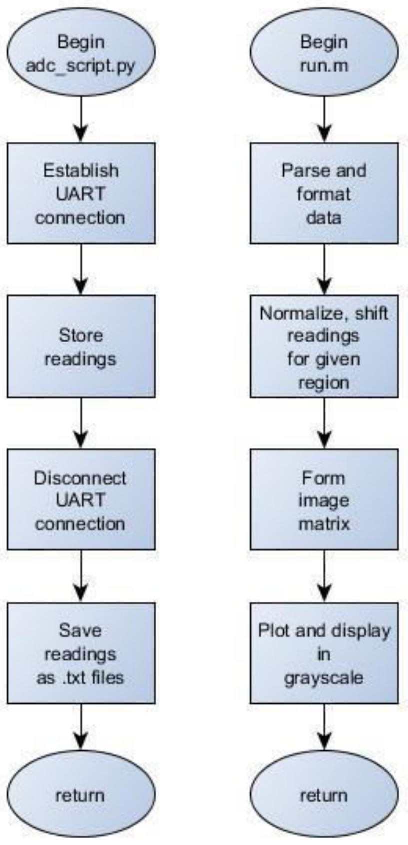

Begin adc_script.py
Establish UART connection
Store readings
Disconnect UART connection
Save readings as .txt files
return
Begin run.m
Parse and format data
Normalize, shift readings for given region
Form image matrix
Plot and display in grayscale
return

Figure 2.13 Python (left) and MATLAB (right) scripts' flowcharts

Since this is a prototype, quantitative analysis of the image is postponed for the final design and only qualitative analysis is performed. Sample used for this scan is a transparent ruler array with 10 μm distance between each marking. Obtained LSCM image and wide-field image of the ruler are shown in Figure 2.15 left and right, respectively. Shortcomings of this image stem from optical alignment, electrical noise and non-precision of filtering. Filters cause different phase margins and therefore break the symmetry.

However for a prototype this is an acceptable image and it is sufficient enough for obtaining system specifications which are discussed in the next section, where final design is performed.

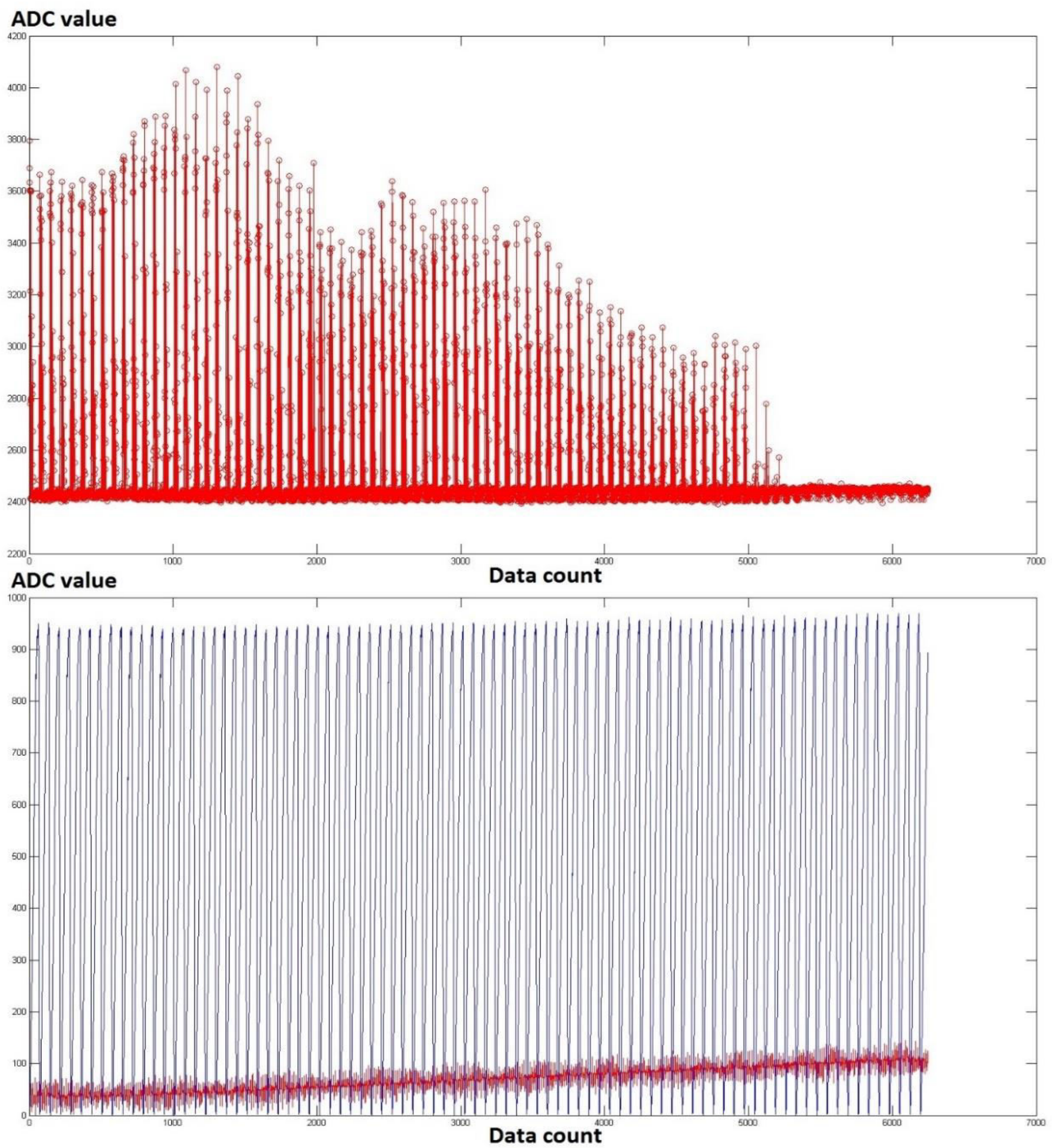


Figure 2.14 Plot of PMT (top), motor position (bottom) readings where X is the slow signal and Y is the fast signal

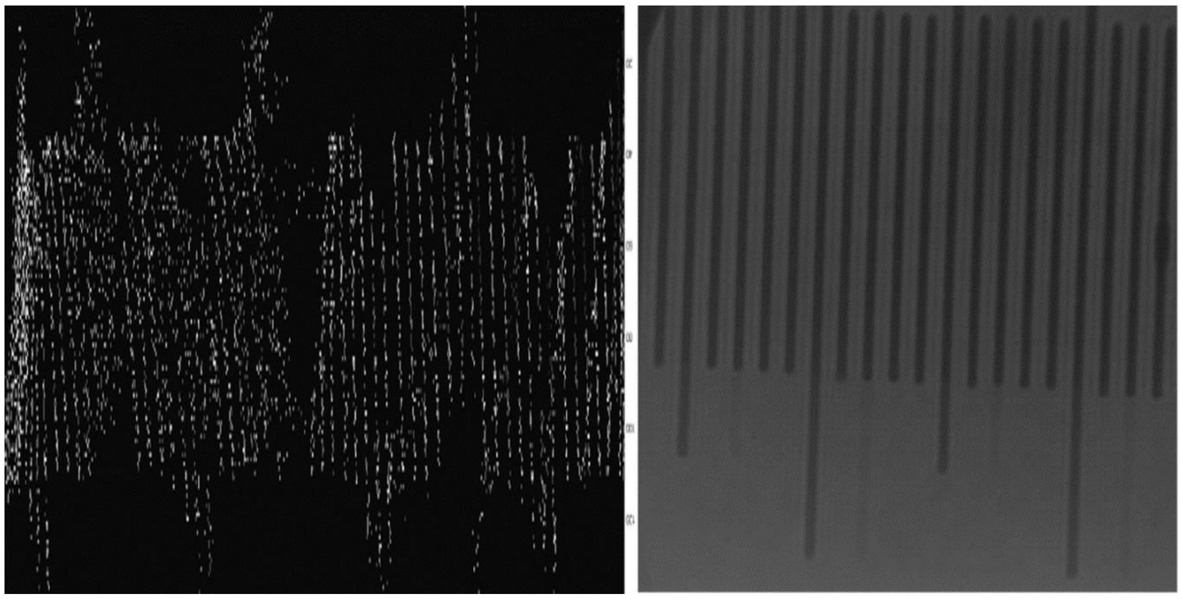


Figure 2.15 Obtained confocal image (left), original image (right)

Chapter 3

HARDWARE DEVELOPMENT OF LSCM

3.1 Requirements and Specifications

A prototype of LSCM has been developed and characterized in the previous chapter. The aim was to find a working solution and necessary specifications for developing the best final custom design. This final implementation aims to compensate all the shortcomings of the prototype and it also aims to deliver a turn-key solution for everyday LSCM user. To implement this final custom design, following requirements and specifications have been obtained from the prototype:

a) *Scanning system*

- Configurable scan area	[length (l) x height (h)]
- Configurable scan resolution	[m (x axis) x n (y axis)]
- Configurable scan period	[t_x and t_y]
- Configurable scan type	[sawtooth, triangular]

b) *Data acquisition system*

- UART-less, USB based data acquisition
- Differential sampling for X, Y and PMT inputs

c) *MCU*
  - Single Tiva C MCU based design
  - Handshake must be performed before start
  - DAC based design for driving galvanometer scanning mirrors
  - ADC based design for reading motor position and PMT

d) *Interfacing circuit*
  - Power management circuitry for DAC
  - Buffer for DAC and ADC channels
  - PMT gain voltage knob
  - Box based design, switches/indicator lights

e) *Physical dimensions*
  - Proper boxing and cabling [2 boxes with connections and cabling]
  - Power supply and galvanometer scanner driver box
  - CDAQ box

These requirements aim to provide a user friendly, stable, sustainable, high quality control and data acquisition (CDAQ) system. Highly configurable scan options, DAC and USB based high speed scanning and data acquisition ability, precision techniques for reducing crosstalk, interference, and noise, errorless bridging between hardware and PC are some of the goals of this design. Overall custom hardware is enclosed for fitting the concept in a systems approach. To meet these requirements, following specifications were derived from the working prototype.

a) Scanning system

Parameter	**Condition**	**Value**	**Unit**
Scan area	Max	X: 205 Y: 165	µm
Scan voltage range	Max	X: -1.46 to + 1.46 Y: -1.06 to +0.88	V
Scan resolution	Min	0.1	µm
Scan period	Std	0 – 100% of DAC speed	s
Scan type	Std	Triangle, sawtooth	-

b) Data acquisition system

Parameter	**Condition**	**Value**	**Unit**
X/Y output range	Max	X: -1.08 to 1.08 Y: -0.77 to 0.64	V
PMT output range	Std	0 – 10	V
PMT control voltage	Std	0 - 1.25	V
Data transfer mode	Std	12	MBps

c) MCU

Parameter	**Condition**	**Description**
MCU	Std	Tiva C Series: EK-TM4C123GXL
ADC	Std	12-bit, 1MSPS, differential, built-in
DAC	Std	16-bit, Texas Instruments 8551

d) Interfacing circuit

Parameter	**Condition**	**Value**	**Unit**
DAC buffer	Std	Maps X: -1.46 to +1.46 Maps Y: -1.06 to +0.88	V
PMT control voltage	Std	Regulator + potentiometer maps to 0 – 1.25	V
PMT input	Std	0 - 10	V
ADC buffer	Std	Capacitive coupling circuit	-

e) Physical dimensions

Parameter	**Limits**	**Description**
Box 1	L: 28.5 cm W: 18 cm H: 15 cm	- 2 x 15 V 150 W power supply - Galvanometer scanner driver - Galvanometer scanner
Box 2	L: 19 cm W: 20.5 cm H: 5.5 cm	- MCU - ±15 V, ground connections - PMT control unit - Motor control unit - Motor reading unit

To illustrate physical properties of these systems, Figure 3.1 is provided. As this figure suggests, Box 1 is responsible of powering the whole system and enclosing the provided galvanometer scanner driver. Box 2 contains the controllers for driving mirrors (DAC), MCU (with built-in ADC) and PMT gain circuitry. It also connects to PC in order to perform in accordance with the user interface software. This physical layout is also available as a functional block diagram in Figure 3.2, where interconnections between building blocks are more visible.

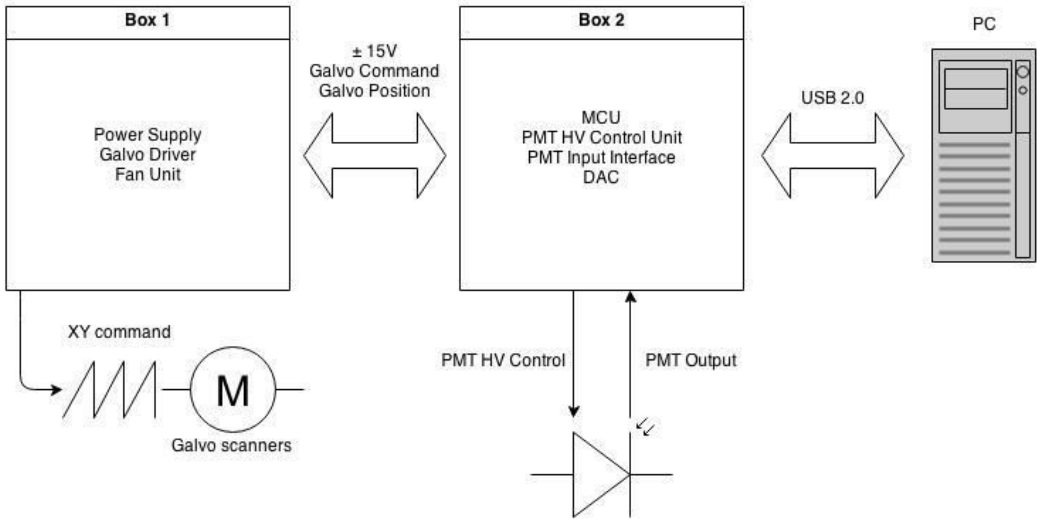


Figure 3.1 CDAQ physical layout

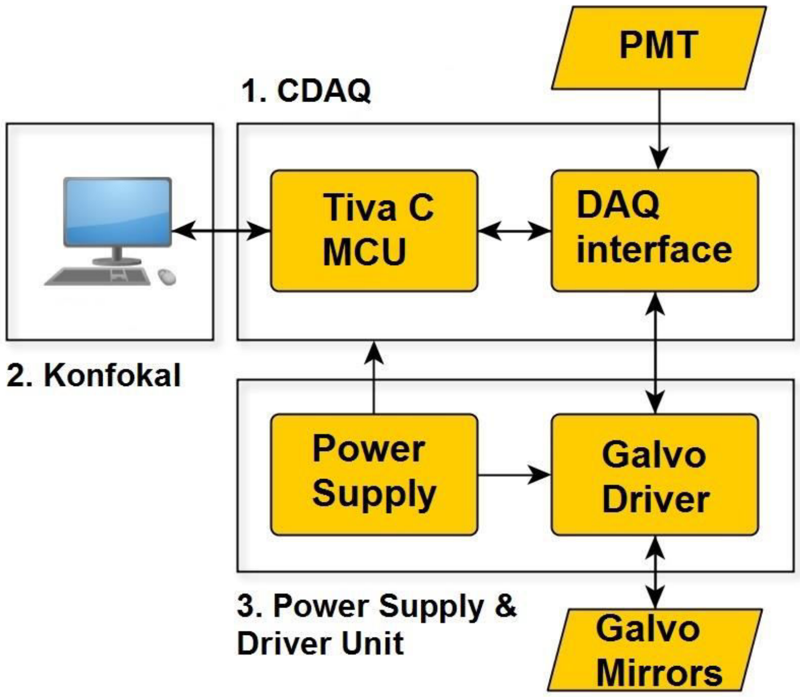


Figure 3.2 Functional block diagram of the system

“Power Supply & Driver Unit” was detailed in previous chapters and “Konfokal” software will be detailed in the next chapter. Since this chapter is on hardware development, only the CDAQ will be detailed here.

3.2 Building Blocks of CDAQ

3.2.1 MCU

MCU used in this design is a Texas Instruments Tiva C Series EK-TM4C123GXL evaluation board. This board is very suitable for control and data acquisition applications because it makes high speed data transfer possible with its built in 12-bit 1MSPS ADC, SPI and USB peripherals. Moreover it comes with the well-known and supported ARM Cortex-M4 processor. Normally, collecting all these features together would not be efficient in terms of cost and time. Therefore this approach is chosen in order to design a reliable and efficient solution.

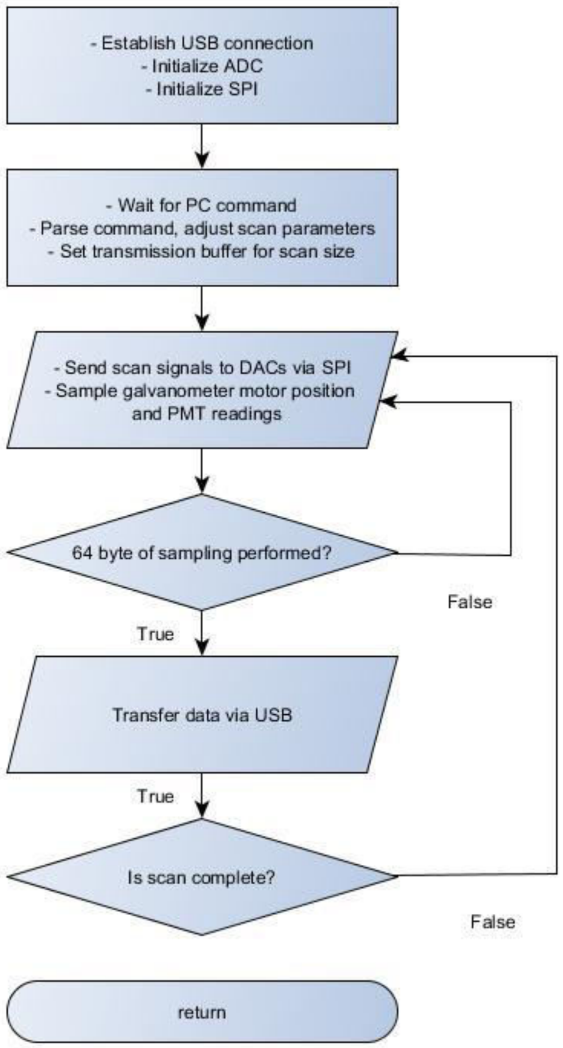

- Establish USB connection
- Initialize ADC
- Initialize SPI
- Wait for PC command
- Parse command, adjust scan parameters
- Set transmission buffer for scan size
- Send scan signals to DACs via SPI
- Sample galvanometer motor position and PMT readings
64 byte of sampling performed?
False
True
Transfer data via USB
True
Is scan complete?
False
return

Figure 3.3 Flowchart of CDAQ firmware

This MCU takes responsibility of every crucial hardware level task. It samples the PMT and X/Y galvanometer scanner position channels, drives mirrors via SPI protocol, and sends recorded scan data to PC.

Actually, this MCU grants the “device” properties to whole system, so that user interface (Konfokal) can perform given tasks. To meet these goals, an interrupt-driven firmware is written for this MCU. The flowchart of this firmware is available in Figure 3.3, also the code for this firmware is available in Appendix.

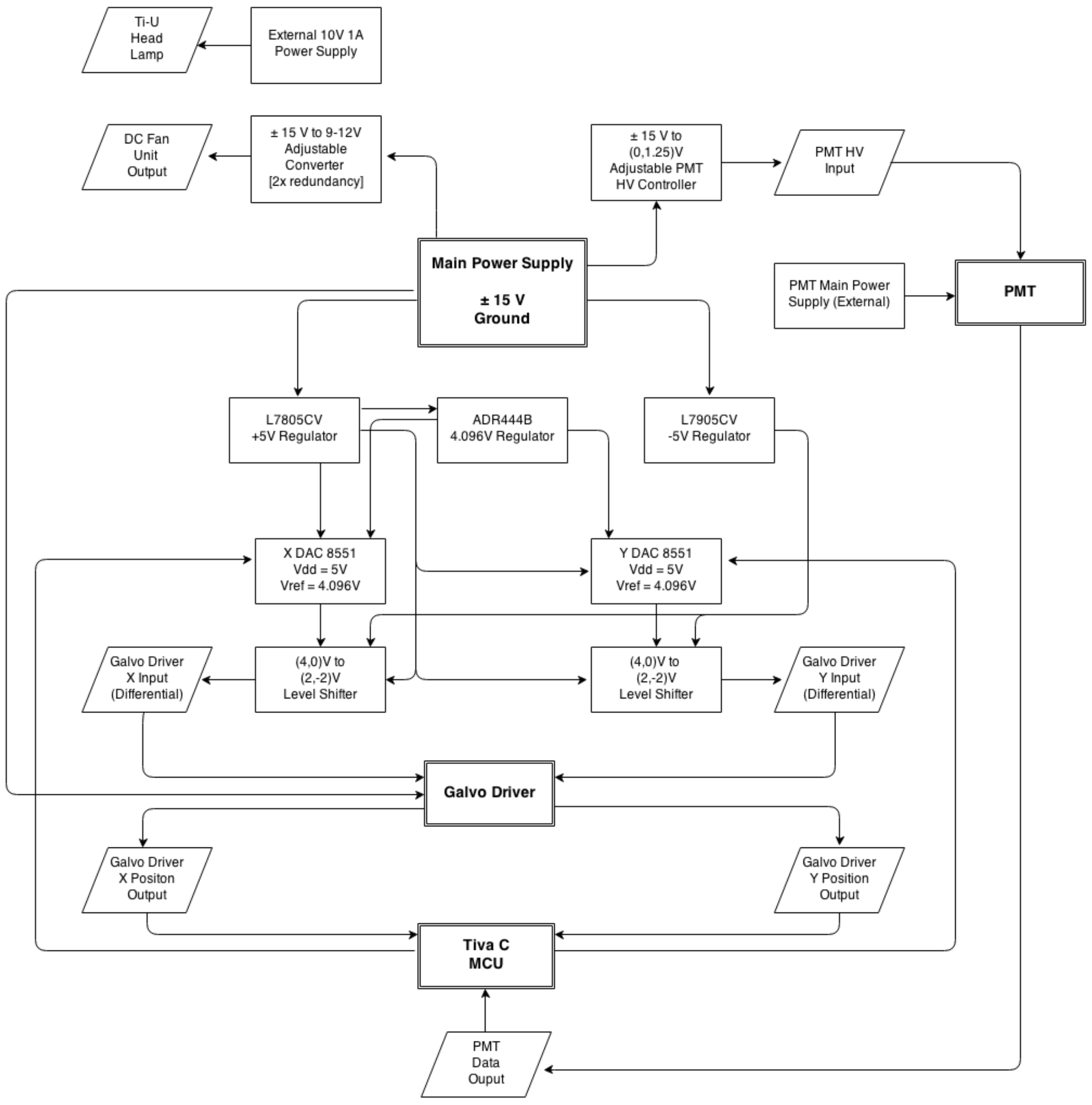

Ti-U Head Lamp
External 10V 1A Power Supply
DC Fan Unit Output
± 15 V to 9-12V Adjustable Converter [2x redundancy]
± 15 V to (0,1.25)V Adjustable PMT HV Controller
PMT HV Input
Main Power Supply
± 15 V Ground
PMT Main Power Supply (External)
PMT
L7805CV +5V Regulator
ADR444B 4.096V Regulator
L7905CV -5V Regulator
X DAC 8551 Vdd = 5V Vref = 4.096V
Y DAC 8551 Vdd = 5V Vref = 4.096V
Galvo Driver X Input (Differential)
(4,0)V to (2,-2)V Level Shifter
(4,0)V to (2,-2)V Level Shifter
Galvo Driver Y Input (Differential)
Galvo Driver
Galvo Driver X Positon Output
Galvo Driver Y Position Output
Tiva C MCU
PMT Data Ouput

Figure 3.4 Electrical schematic of DAQ interface

3.2.2 DAQ Interface

DAQ interface is a highly modularized interfacing circuit that is designed for the CDAQ. It is composed of main board, MCU slot and DAC slot. Components of the main board are through-hole type, however DAC slot components are SMD (surface mount device) type. This hybrid approach supports future updates of MCU and DAC units, so that faster and better DACs can be used with the compatible main board of DAQ interface. Also, glitches that can occur because of failure can be solved by replacing old unit with the new, therefore new cost of designing from the beginning is avoided. Figure 3.4 describes the electrical schematic of DAQ in the best and simplest way possible.

Pluggable DAC unit (with its DAC8551 ICs) delivers better performances in resolution and speed. These DACs map 16-bit to 0 – 5.5 V range and they provide μV level resolution. Also the settling time of DACs are given as 10 μs. This results in a faster scan time. DAC circuit schematic is available in Figure 3.5. Schematic of the mainboard that hosts pluggable DAC unit and MCU unit is available in Figure 3.6. It acts as a power management board with extended ADC buffer unit. Input protection circuit of input peripherals are available in Figure 3.7. The simulations of these units with filters are available in Figure 3.8 and Figure 3.9. Overall view of the completed units are available in Figure 3.10.

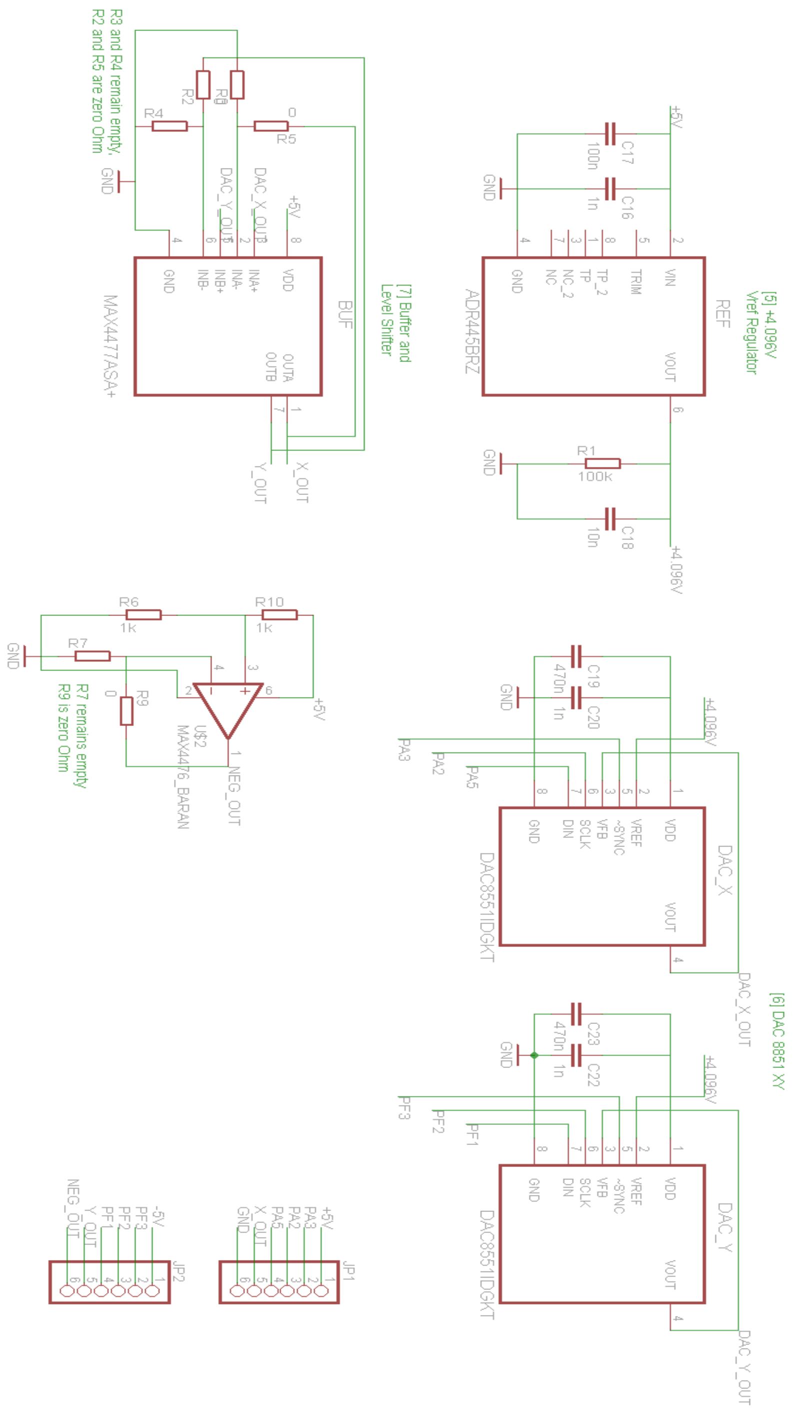


Figure 3.5 DAC unit schematic

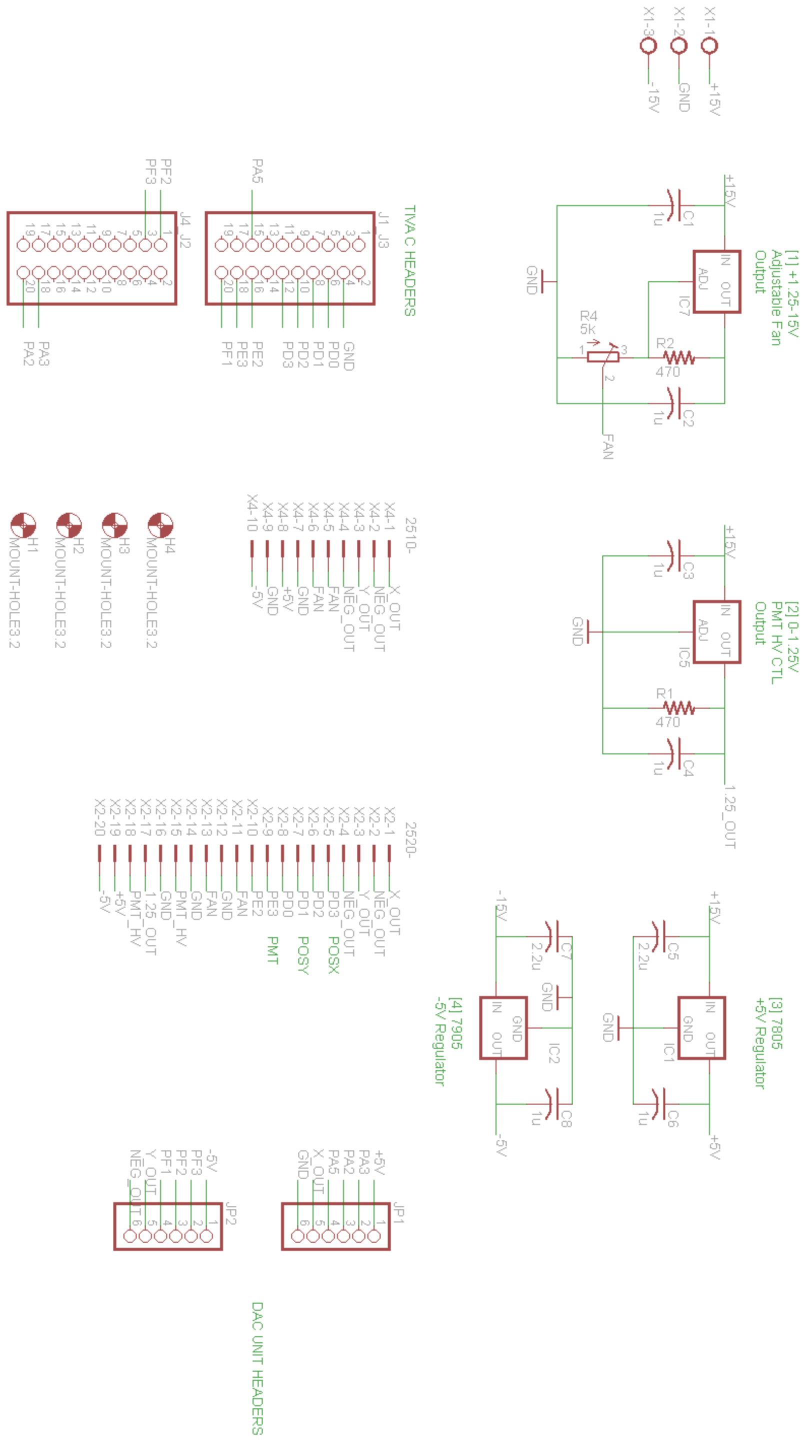


Figure 3.6 CDAQ mainboard

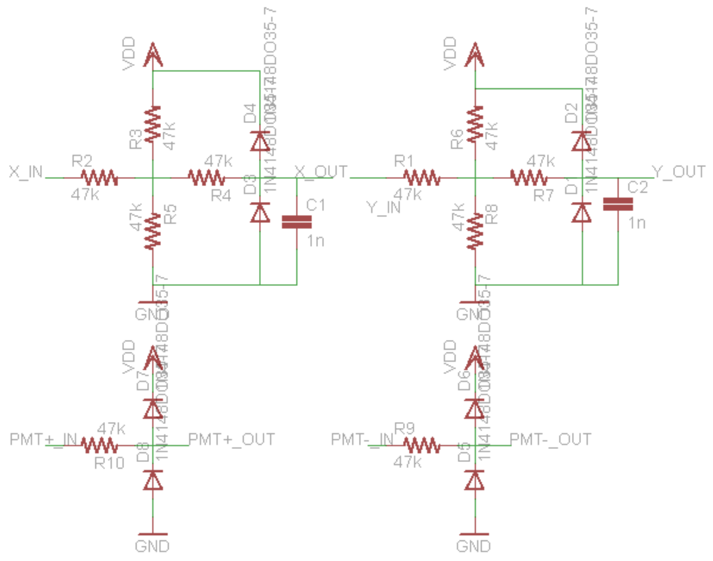


Figure 3.7 Input protection circuit

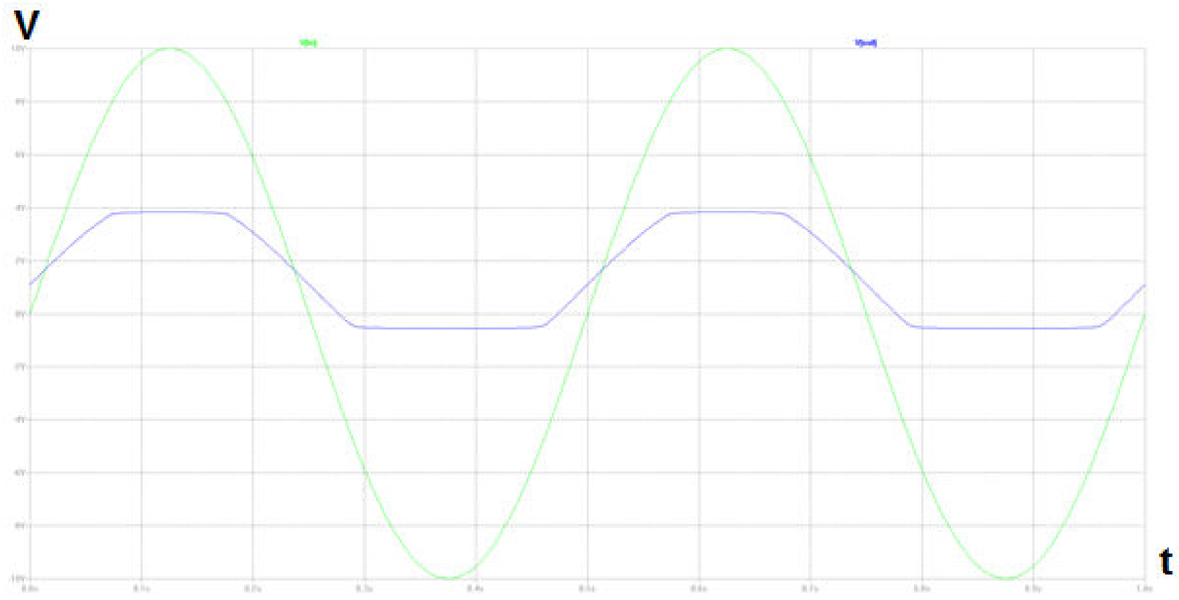


Figure 3.8 Simulation of position signal buffer

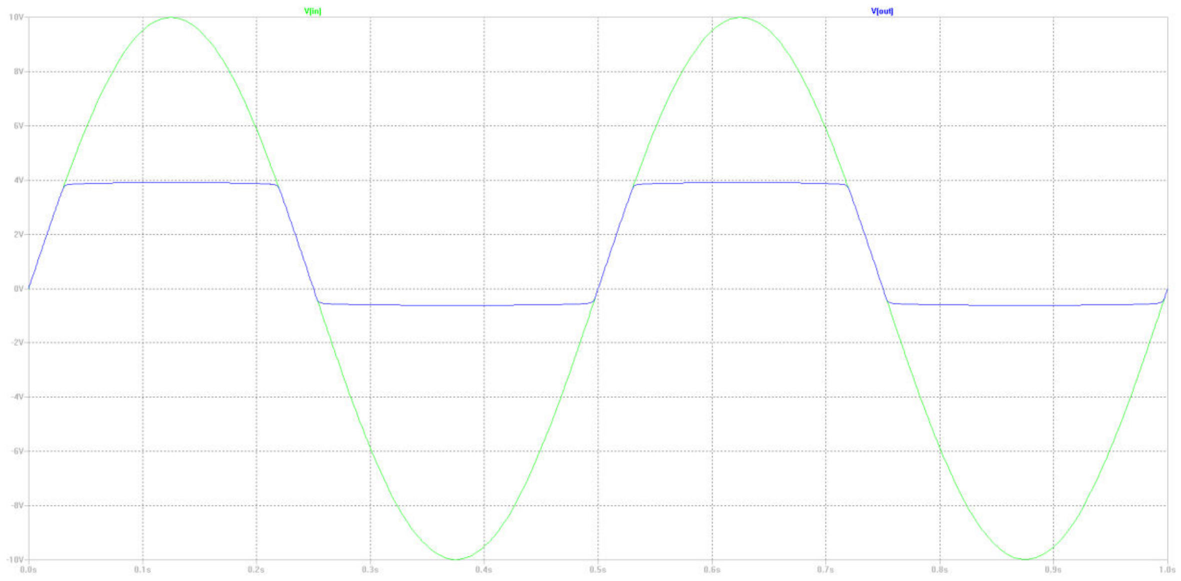


Figure 3.9 Simulation of PMT signal buffer

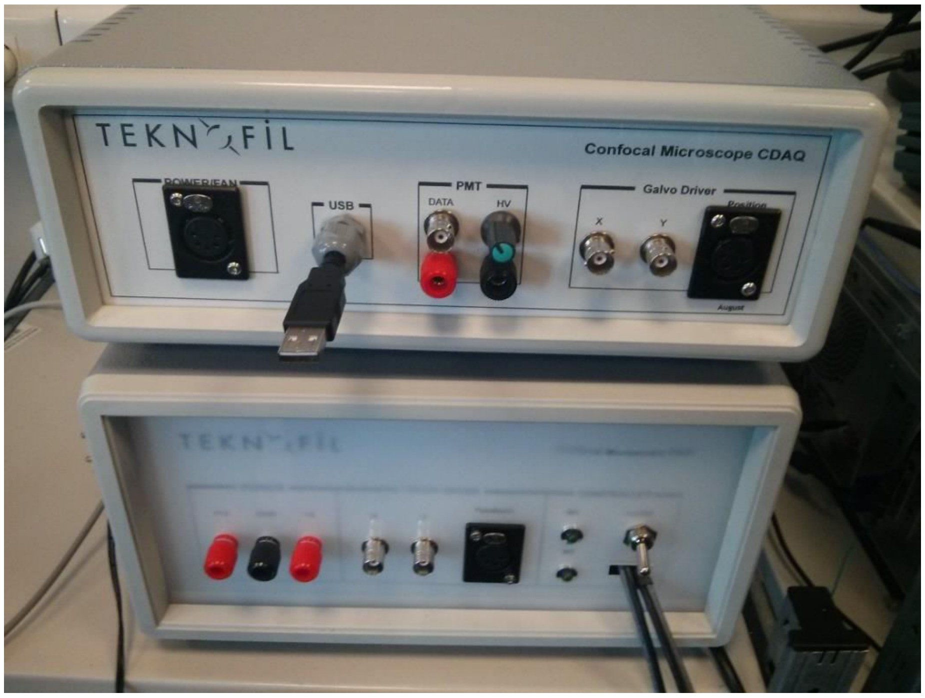


Figure 3.10 Enclosed CDAQ system

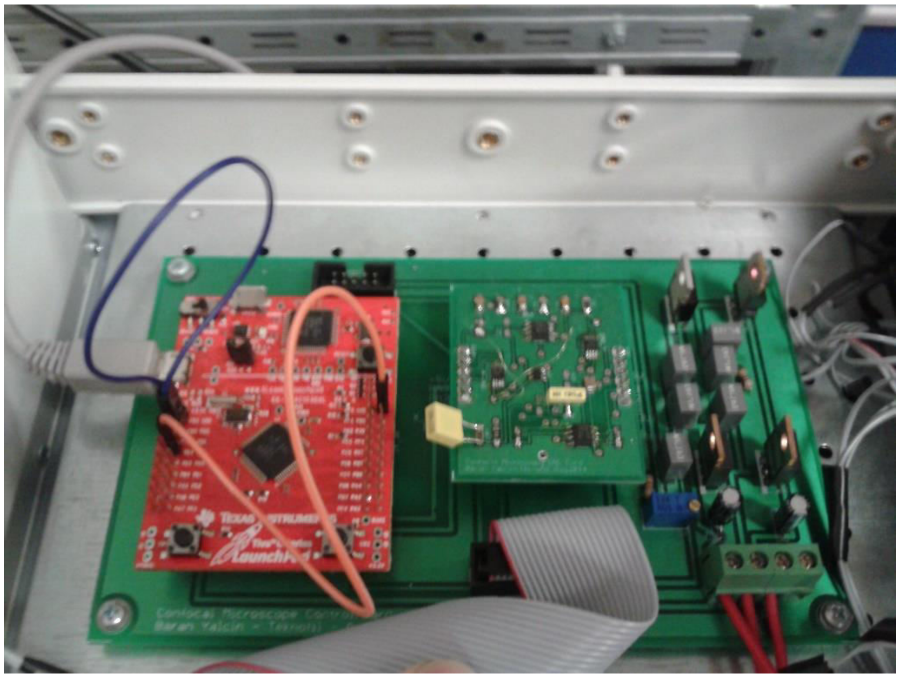


Figure 3.11 Mainboard PCB of CDAQ (Box 2)

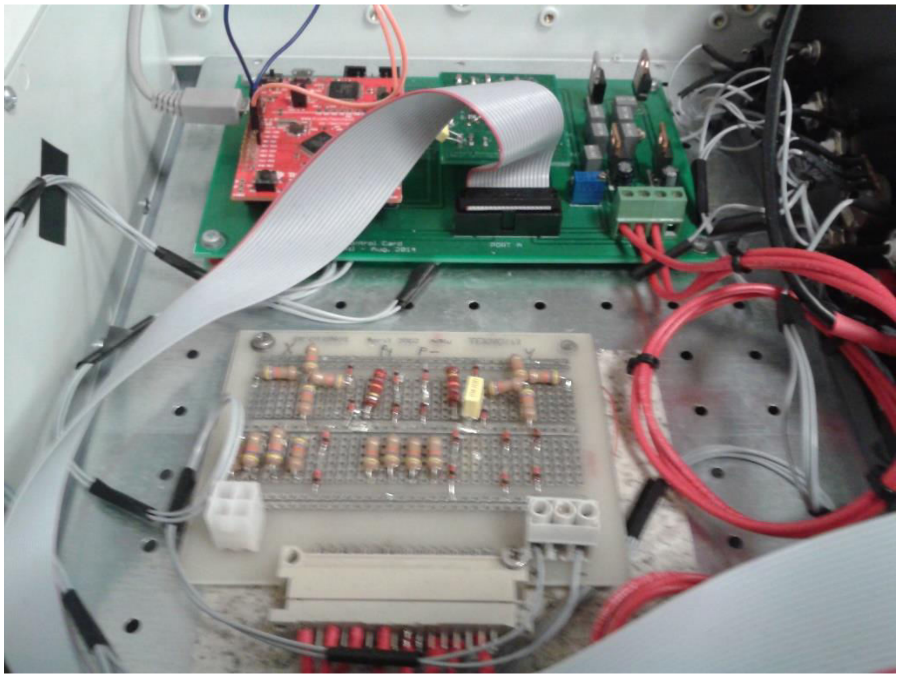


Figure 3.12 Input buffer of CDAQ (Box 2)

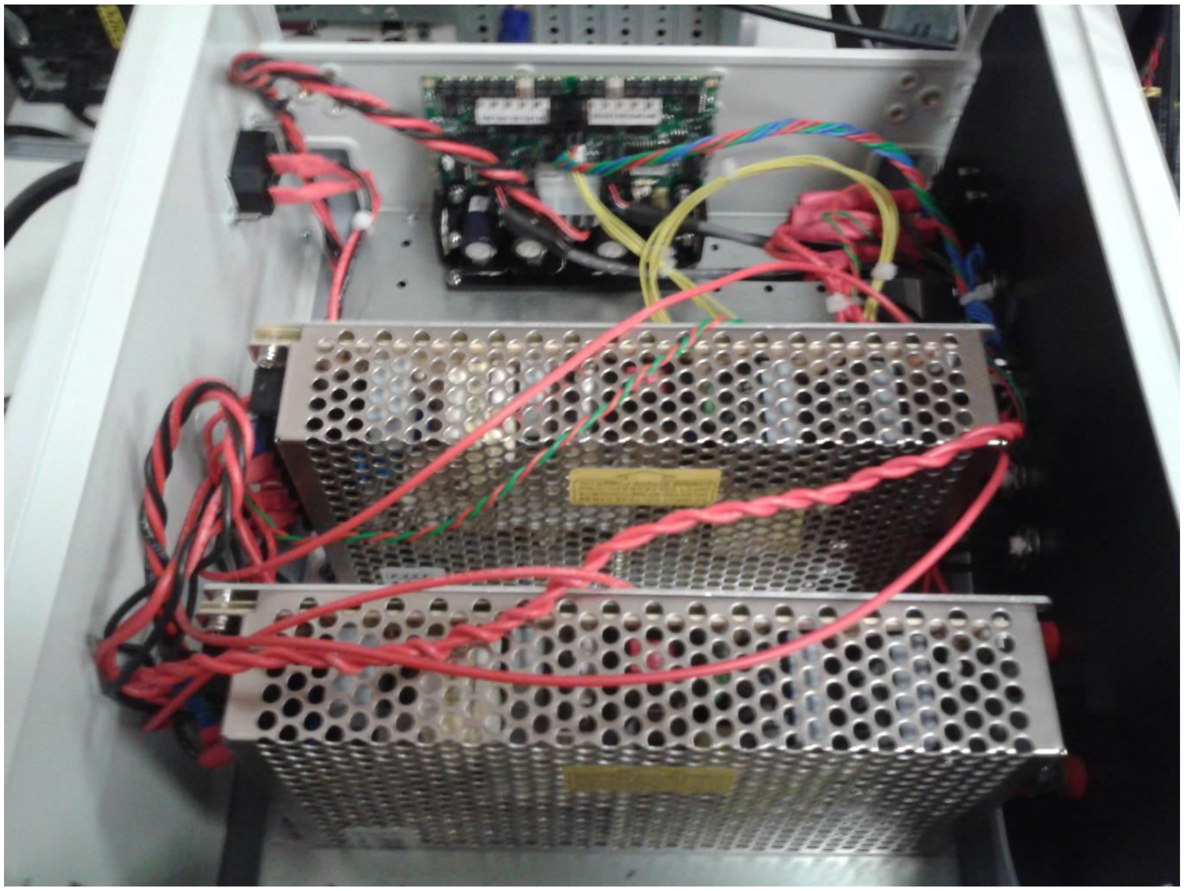


Figure 3.13 Power supply and motor driver units of CDAQ (Box 1)

Chapter 4

SOFTWARE DEVELOPMENT OF LSCM

4.1 “Konfokal” Program

In previous chapter, capabilities of the LSCM hardware have been documented. To have better control over this hardware, a PC application is offered. This software is known as “Konfokal” and its duties are:

- Processing and transferring given scan commands
- Processing received data to form scan output
- Forming image and scan info
- Storing, loading and performing user related actions

In this context, Konfokal, which is a mixture of an LSCM application software and an image editing software, aims to solve all problems with a single application.

Konfokal is written in Python programming language for version 2.7. Its user interface is Qt Framework port for Python, which is known as PyQt 4. Besides these, Konfokal program depends on following external Python modules:

- matplotlib
- NumPy
- SciPy
- PIL

- PyUSB
- PySerial

The program can run under any Python supported platform, however due to driver issues of available hardware, program is written for and tested under Microsoft Windows 7 operating system.

4.2 Architecture of Konfokal Program

Konfokal program is composed of different modules that control different parts of the program. It is composed of a front-end and a back-end. Front-end handles the user interface related components such as layouts, widgets, icons and text. This is grouped under “ui” (user interface) module. Back-end handles device features (dev), utilities (utils) for interfacing with other services and core (core) modules where the inner workings of the program exist. These are hierarchically available in Figure 4.1. Duties of these modules are detailed for a clearer understanding.

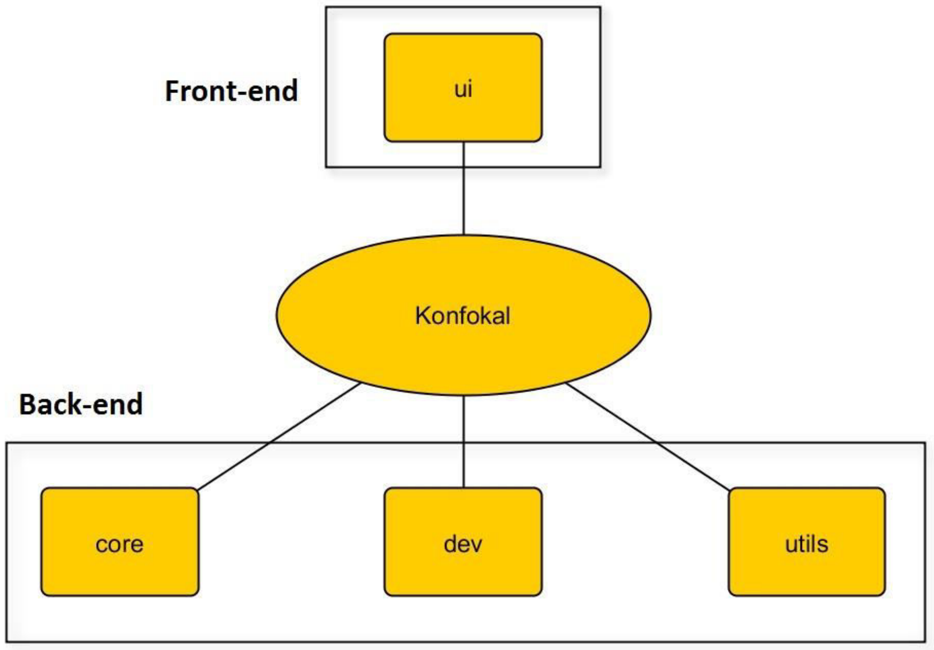


Figure 4.1 Architecture of the program

4.2.1 Front-end

- ui: This module is responsible of graphical user interface (GUI) tasks. Windows, widgets and layouts are drawn to screen with the help of this module. It also sends signals to back-end and displays signals that come from the back-end. Figure 4.2 is the screenshot of the main window that belongs to front-end.

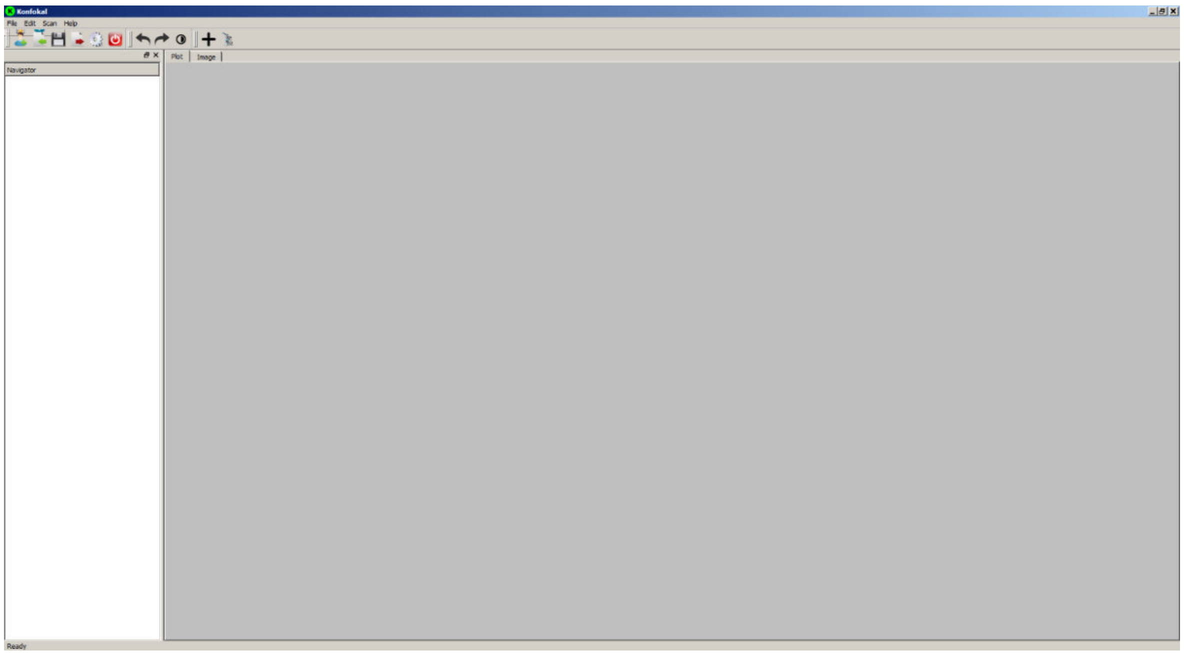


Figure 4.2 Main window of front-end

The screenshot of the menu bar of this front-end is available in Figure 4.3. These menus with their elements are detailed here.

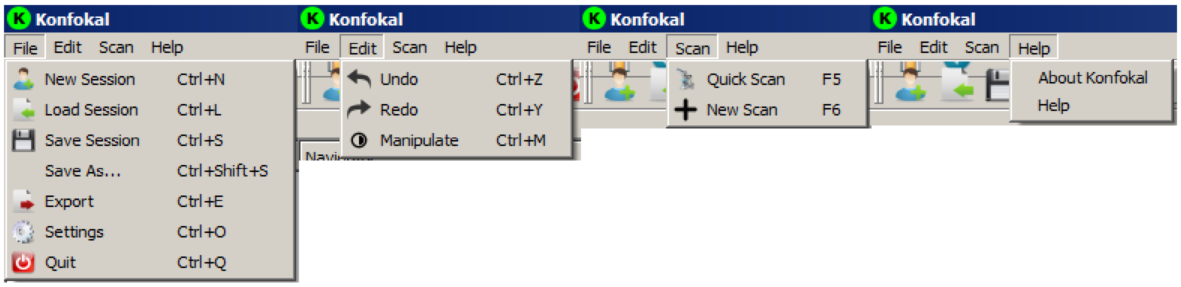
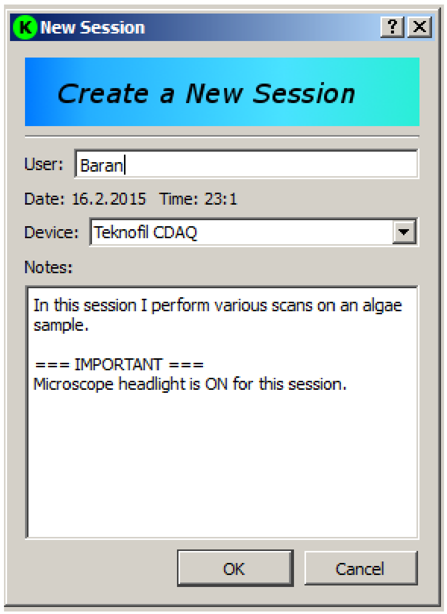


Figure 4.3 Konfokal menus

File menu has following elements:

- *New Session:* Initializes a new session. Konfokal program is based on session and scan notions. Sessions are composed of scans, therefore by creating a new session one can add scans under that session. Figure 4.4 shows the screenshot of *New Session* window.

In a new session one must include the user name, select the device type and also include notes which can become useful while analyzing the data and image.



User: Baran

Date: 16.2.2015 Time: 23:1

Device: Teknofil CDAQ

Notes:

In this session I perform various scans on an algae sample.

=== IMPORTANT ===
Microscope headlight is ON for this session.

Figure 4.4 New Session window

- *Load/Save (As) Session:* Konfokal program saves data in its own file type. This feature loads/saves sessions with scans and data. Loading/saving gives other researchers an opportunity to share files between computers. Figure 4.5 shows the Load/Save (As) window of the program.

- *Export:* This feature exports the scan image, data and plot. It is useful for extracting the raw data.

- *Settings:* Adjusts settings of Konfokal program.

- *Quit:* Exits from program.

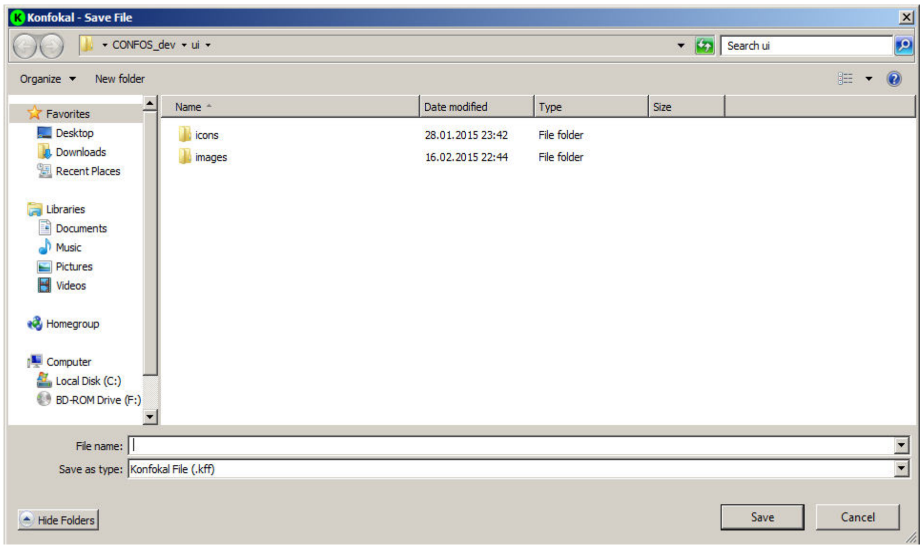


Figure 4.5 Load/Save (As) window

Edit menu has following elements:

- *Undo:* Recovers action.

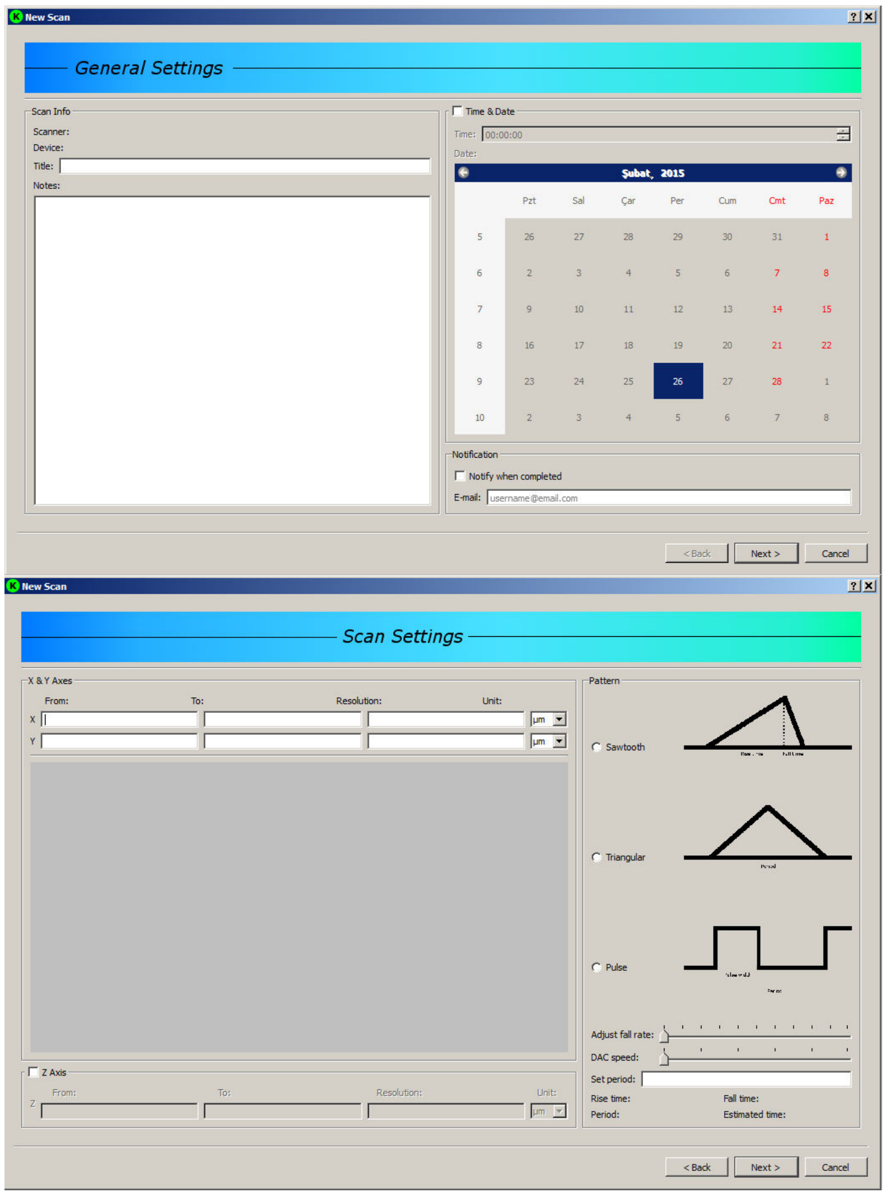
- *Redo:* Repeats action.

- *Manipulate:* This element is used for manipulating the image and data. Its manipulation operations are based on PIL (Python Imaging Library) operations. Manipulate element provides basic image processing operations such as filtering.

Scan menu has following elements:

- *Quick Scan:* This element repeats the last performed scan. It saves the user from entering every detail again.

- *New Scan:* Performs a new scan for the given parameters. These scans are added under the selected session. When user selects new scan, wizard in Figure 4.6 pops up.



	Pzt	Sal	Çar	Per	Cum	Cmt	Paz
5	26	27	28	29	30	31	1
6	2	3	4	5	6	7	8
7	9	10	11	12	13	14	15
8	16	17	18	19	20	21	22
9	23	24	25	26	27	28	1
10	2	3	4	5	6	7	8

Figure 4.6 New Scan wizard

This wizard is composed of two windows. One of them is “General Settings” where title, notes, date/time and notifications are made available for the user. Other one is the “Scan Settings” window where scan range can be selected in micrometers range. Range and resolution information are provided in the last section. Later, scanning waveform is set and if it is a saw tooth wave, then fall rate of the signal is also set. DAC speed is another important parameter that effects the scan time. This can be adjusted between 0 – 100% of DAC’s maximum speed value which is available in the datasheet of the DAC. Once these parameters are selected and set, Konfokal sends start signal to CDAQ.

Help menu has following elements:
- *About Konfokal:* Contains information about the program.
- *Help:* Help contents for the program.

4.2.2 Back-end

- core: Encapsulates essential data types such as “Session”, “Scan”, “ScanData”, “ScanImage”. Contains essential methods and functions for storing, loading and manipulating data. This module has the critical components of Konfokal program.
- dev: Contains device types. Along with CDAQ, additional devices can be added under this module. These devices can be used by wrappers that mimic the generic confocal device.
- utils: This module has the utility functions such as Windows services or other operating system related services. New utility functions can be added under this module for later uses.

4.3 Source Code of Konfokal Program

Konfokal is an open source software under GNU General Public License (GPLv2). The source code is made available in Appendix. Users can modify, change or redistribute the source code as long as they comply with the GPLv2. Below is the license notification:

```
Konfokal: LSCM Software

Copyright (C) 2015  Baran Yalcin


This program is free software; you can redistribute it and/or

modify it under the terms of the GNU General Public License

as published by the Free Software Foundation; either version 2

of the License, or (at your option) any later version.


This program is distributed in the hope that it will be useful,

but WITHOUT ANY WARRANTY; without even the implied warranty of

MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the

GNU General Public License for more details.


You should have received a copy of the GNU General Public License

along with this program; if not, write to the Free Software

Foundation, Inc., 51 Franklin Street, Fifth Floor, Boston, MA  02110-
1301, USA.
```

Chapter 5

RESULTS AND DISCUSSION

5.1 Results obtained with CDAQ and Konfokal

In Chapter 2 a prototype of LSCM was developed and characterized. This prototype was built to control the galvanometer scanning mirrors and to collect PMT signal and galvanometer scanning mirror position feedback readings. This prototype operated successfully with few drawbacks in hardware and software domain. Therefore, a better and final version of this device was developed. The hardware developed for this purpose is called CDAQ (described in Chapter 3) and the software developed is called Konfokal (described in Chapter 4). Results obtained with this device are much better than the developed prototype. In this section these results are provided and annotated.

5.2 Large Area Scans

Figure 5.1 shows one of the results obtained with the latest version of Konfokal program and CDAQ hardware. It is the screenshot of the obtained image from a real scan. Compared with the original test image, this image is much clearer and detailed. The acquired image and its comparison to the conventional wide-field image of the same sample are available in Figure 5.2.

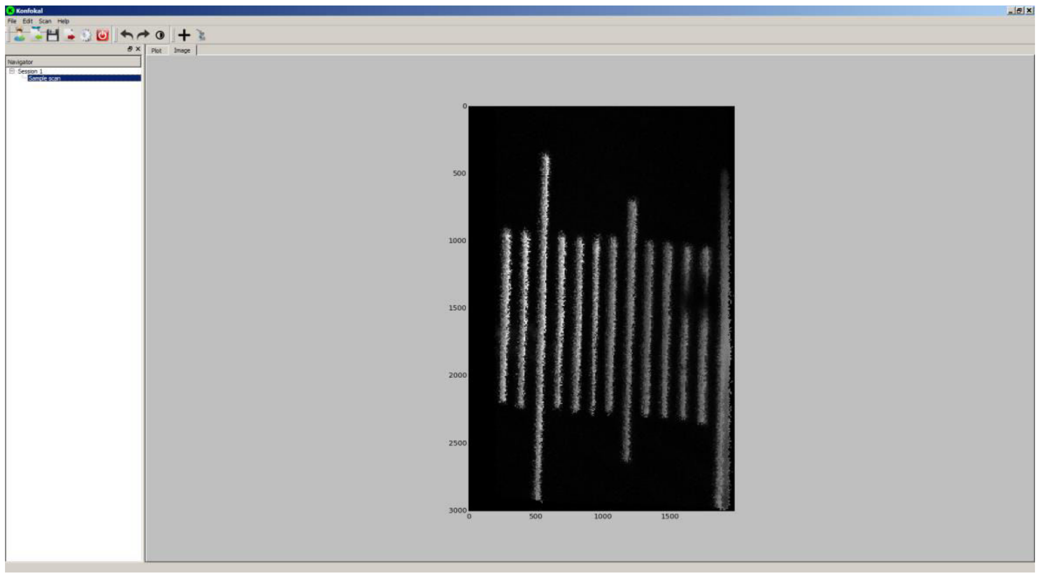


Figure 5.1 Screenshot of the acquired image

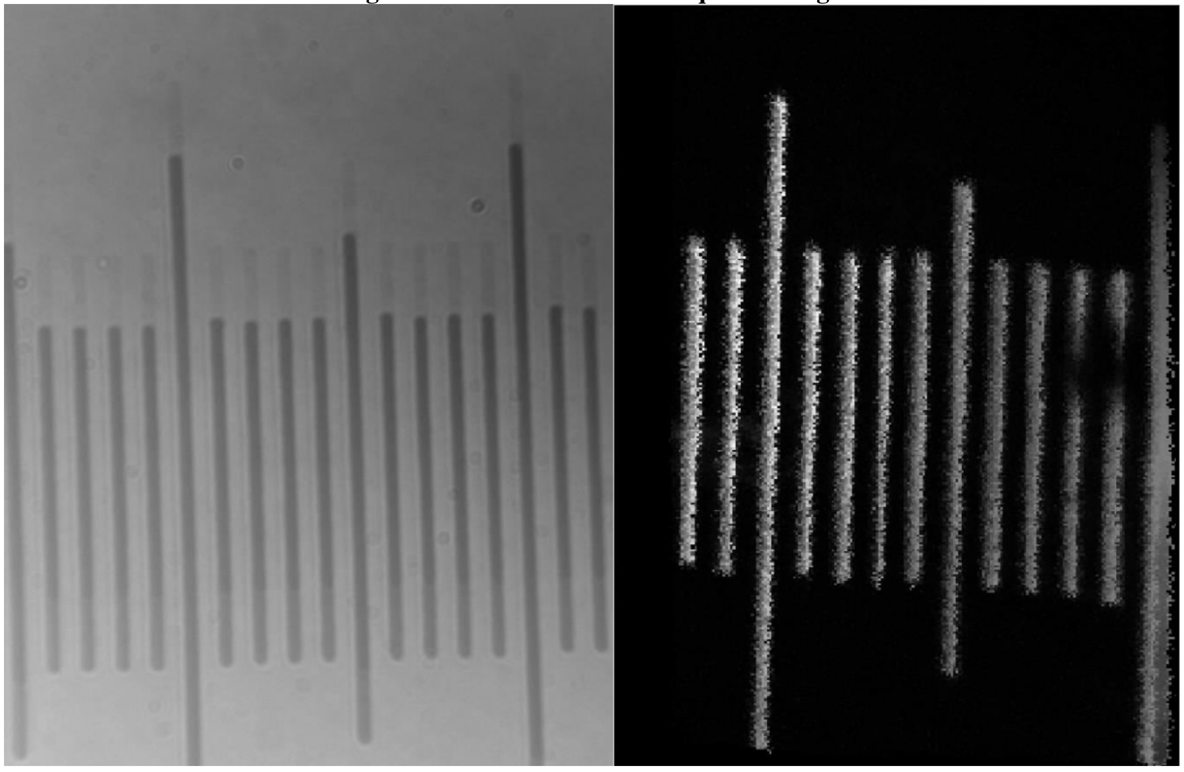


Figure 5.2 Comparison of acquired image

Figure 5.3 shows the image acquired with the prototype. Clearly, new hardware and software have accomplished better and obtained much more successful result. The lines in Figure 5.3 are more noisy, unorganized and asymmetrical, whereas the lines in Figure 5.2 are more solid, perpendicular and symmetrical. The difference between Figure 5.2 and 5.3, or superiority of Figure 5.2 with respect to Figure 5.3 can also be seen from the plot of scan data.

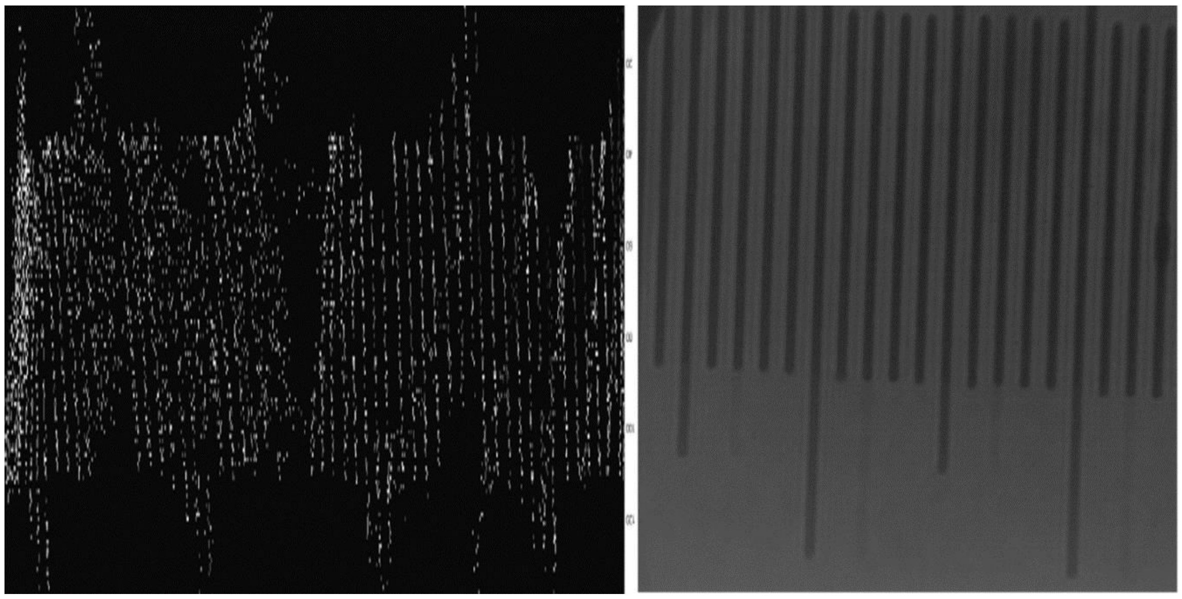


Figure 5.3 Acquired image with protoype

Figure 5.4 and Figure 5.5 show the plot of scan data which are acquired with CDAQ and prototype respectively. An important thing to note here is the distance between lines of Figure 5.3 is 10 µm. Figure 5.4 shows the plot of the scan data, where the top plot belongs to Y channel motor readings, middle plot belongs to X channel motor readings and bottom plot belongs to PMT readings. The same normalization method of prototype is applied while plotting these data. This normalization method shifts the minimum of the signal to zero level and maps signal values to given pixel size. When compared to Figure 5.5, data in Figure 5.4 is much stable and dense from a qualitative perspective.

Because with CDAQ and Konfokal, image acquisition with higher resolution is made possible with the changes introduced in hardware and software domains with the help of DACs, USB protocol and multithreaded firmware.

Minimum theoretical scanning step size, for X channel with the mapped maximum scan range is given by:

$$X_{step} = \frac{205\ \mu m}{2^{16}} = \frac{205\ \mu m}{65536} = 3.13\ nm$$

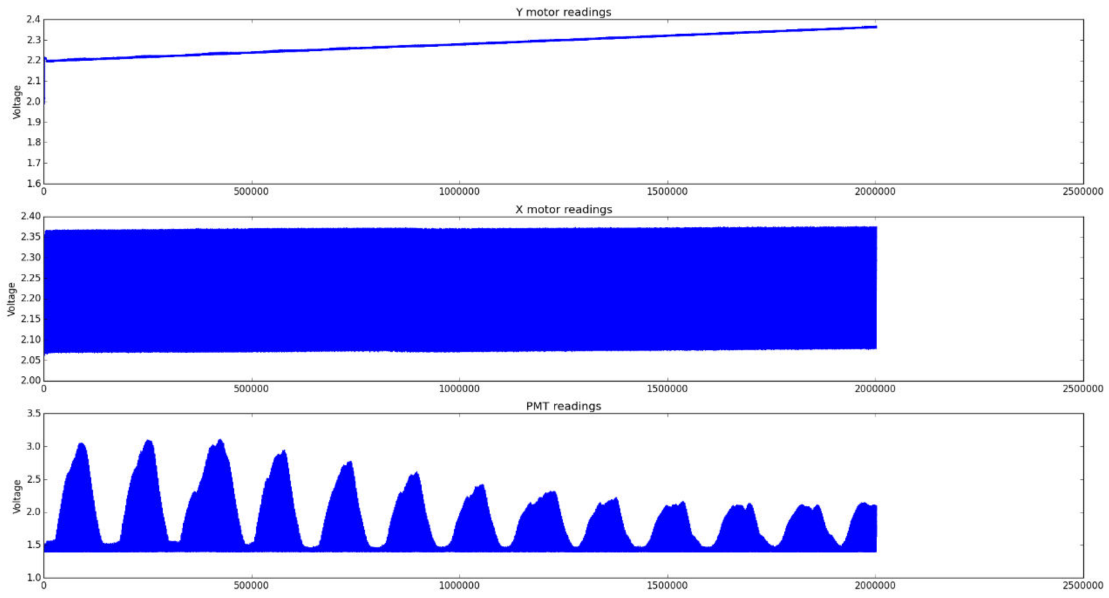


Figure 5.4 Plot of position readings (Y-top, X-middle) and PMT readings (bottom) with CDAQ

For Y channel, minimum theoretical scanning step size is given by:

$$Y_{step} = \frac{165\ \mu m}{2^{16}} = \frac{165\ \mu m}{65536} = 2.52\ nm$$

For the galvanometer scanning mirrors, tilt angle for X and Y axes can be found by multiplying voltage range with position output scale factor:

$$X_{angle} = 0.5 * \left(+1.46 - (-1.46)\right) = +1.46\ ^{\circ}$$

$$Y_{angle} = 0.5 * \left(+0.88 - (-1.06)\right) = +0.97\,°$$

These can be important parameters for optical evaluation of the system.

Another important innovation of CDAQ and Konfokal combination is reduction of channel noise (crosstalk). As one can observe by looking at Figure 5.5's position plot, changes in channel X cause fluctuations in channel Y or vice versa. This occurs because of crosstalk event and it is eliminated with the buffering technique developed for CDAQ. The crosstalk occurs due to parasitic capacitance between two cables and it is removed by introducing shunt capacitors to these input buffers. Another method used for enhancing the image quality finds its use in small area scans. This method does not use motor position readings for forming the image. It uses input position data (rather than feedback data), which holds true for small area scans. This improves the image quality as presented in the next section.

5.3 Small Area Scans

Generally, small areas are of interest in confocal imaging. Therefore, performance of CDAQ and Konfokal have been tested with small area scans. The results suggest that the performance is actually good enough for practical purposes. Acquired images of scans are presented and discussed in this section. In the last section, minimum theoretical scanning step size that can be reached was calculated. Here, step sizes of 50 and 20 out of 65536 ($65536 = 2^{16}$) are the scans performed.

Since these are very little compared to full range, a smaller area has been targeted. Figure 5.6 shows the scan where 50/65536 is set for step size.

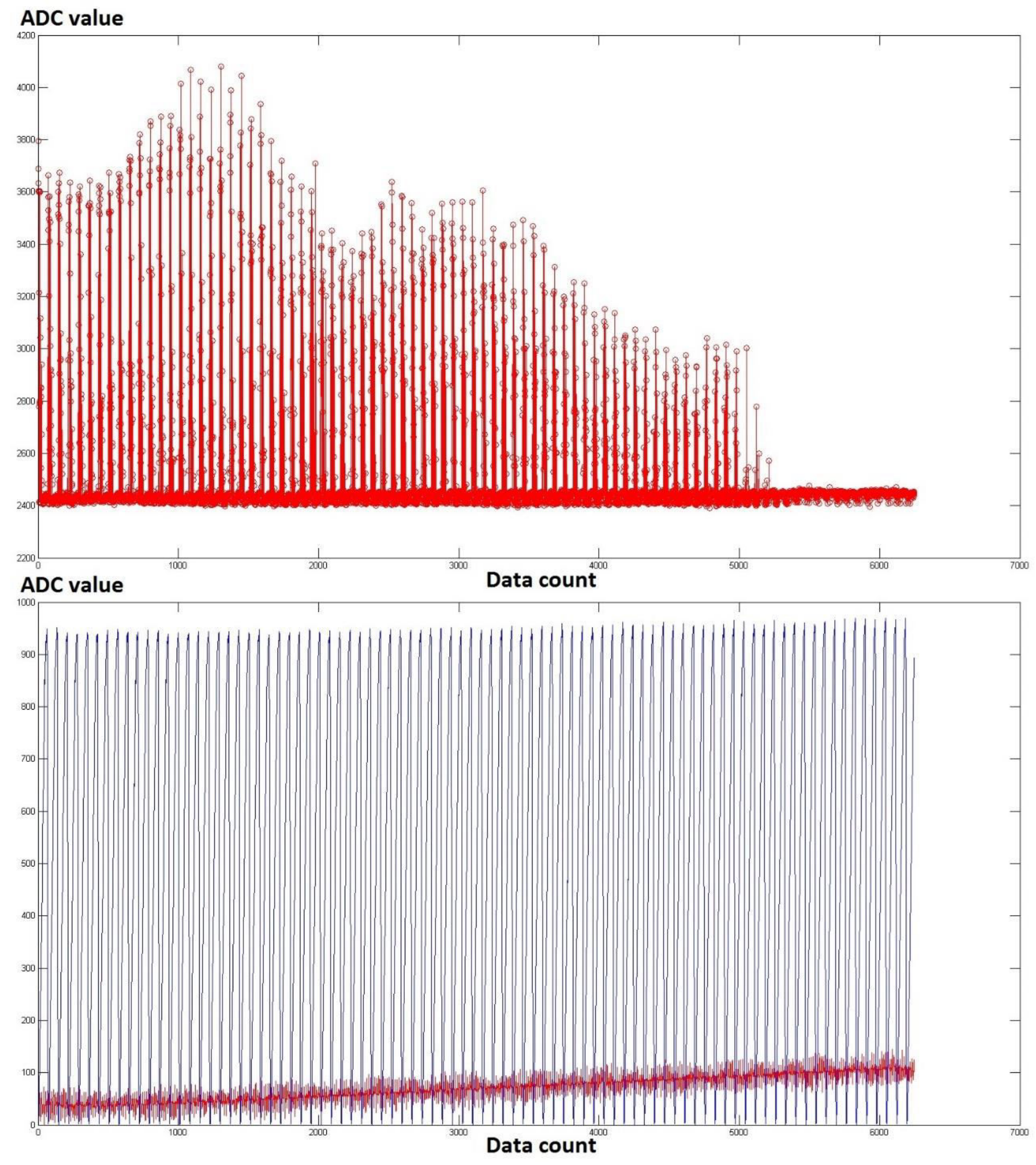


Figure 5.5 Plot of PMT readings (top) and position readings (bottom) with prototype

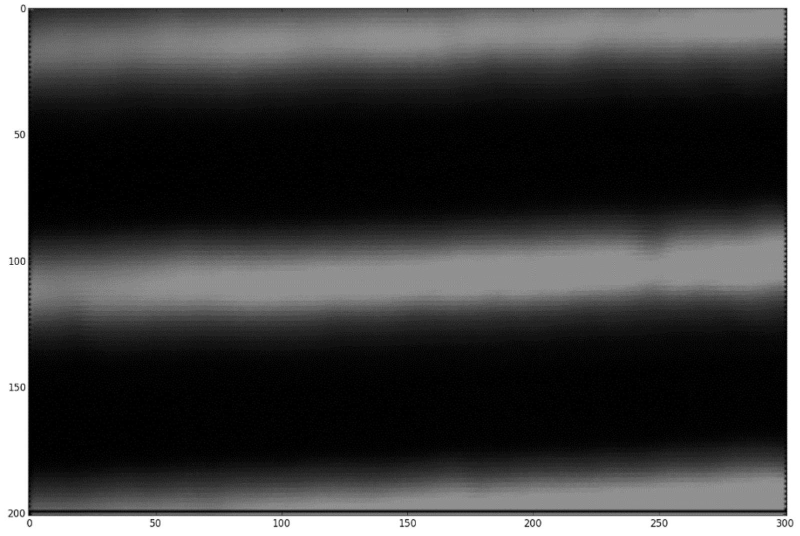


Figure 5.6 50/65536 small area scan

If theoretical step size of Figure 5.6 is to be calculated:

$$X_{step} = 3.13\ nm * 50 = 0.15\ \mu m$$

$$Y_{step} = 2.52\ nm * 50 = 0.13\ \mu m$$

Since Y axis contains three lines and since the distance between two neighboring lines is 10 µm, complete Y axis covers nearly 20 µm with 200 pixels. From here, experimental $Y_{resolution}$ or experimental calibration factor (microns per pixel) can be calculated as:

$$Y_{experimentalresolution} \cong \frac{20\ \mu\mathrm{m}}{200} \cong 0.1\ \mu\mathrm{m}$$

Same can be said for X axis as well. X axis covers nearly 3 lines (because it is an 200 x 300 image), which is around 30 µm. The resulting $X_{resolution}$ or experimental calibration factor (microns per pixel) can be found as:

$$X_{experimentalresolution} \cong \frac{30\ \mu m}{300} \cong 0.1\ \mu m$$

Another small area scan is performed with 20/65536 step size. This can be seen in Figure 5.7. In Figure 5.7, only a small part of a single line is scanned. In this scale, X and Y step sizes can be found as:

$$X_{step} = 3.13\ nm * 20 = 0.06\ \mu m$$

$$Y_{step} = 2.52\ nm * 20 = 0.05\ \mu m$$

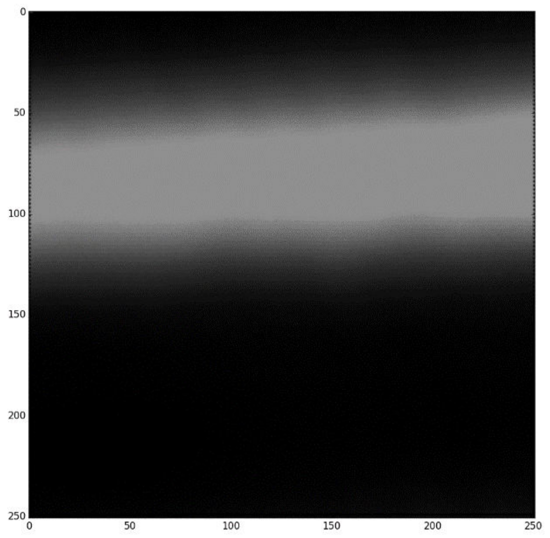


Figure 5.7 20/65536 small area scan (middle of a single line)

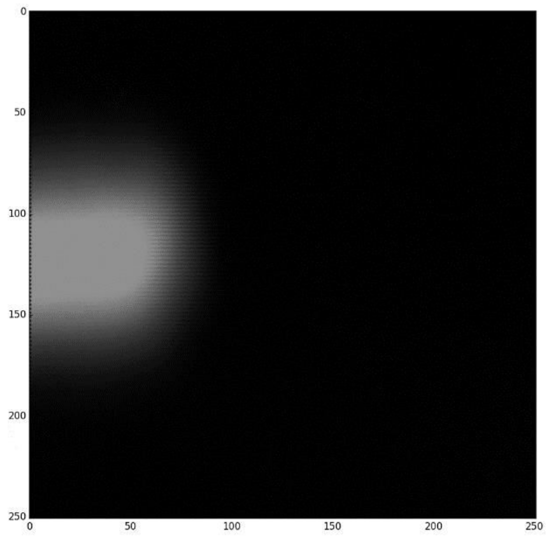


Figure 5.8 20/65536 small area scan (end of a single line)

5.4 Conclusion and Current Status

Results that were detailed in the last section prove that CDAQ and Konfokal combination is able to acquire confocal images successfully. Theoretically and experimentally, they can obtain confocal images at decent resolution. Adjustments for CDAQ can be made for the needed application where the intended resolution or speed is critical. Availability of modification and openness are most important aspects of this design. In this context, CDAQ is an open hardware and it forms a boilerplate to researchers who need to implement an LSCM hardware.

On the other hand, Konfokal also adopts openness philosophy. This makes Konfokal an open source software and it is available for modification, for other use scenarios. One example of this is its compatibility with DMD based confocal microscope which is used as another method in confocal microscopy. Konfokal program is developed for generating scan patterns for DMD based confocal microscope. It can generate stored pattern sequences, which can be used for scanning the sample. One of this patterns is available in Figure 5.9.

All in all, in this thesis a working hardware and software were developed and offered for use. This is done in an “open” way and shared with other researchers/developers. With this hardware/software time and money can be saved while working on a confocal setup. Others can use these schematics and software as a basis for their projects. The validity of these designs and implementations were proved by the results that are detailed in this section.

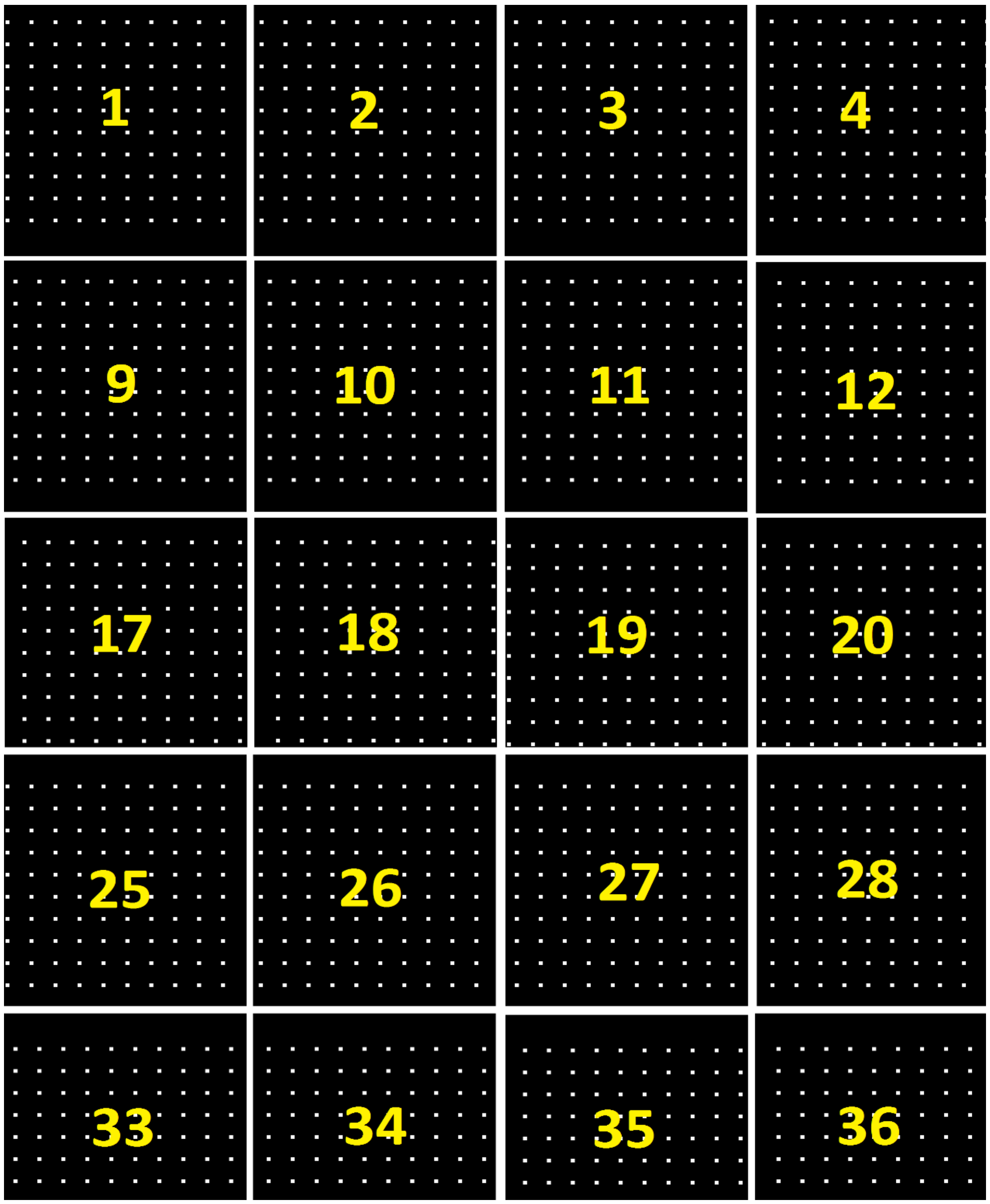


Figure 5.9 Stored pattern generation with Konfokal

APPENDIX

1. Firmware of MCU1:

```
/*
 * KUNRL Confocal Project:
 *                                          PWM Control Software for Galvo Mirrors
 *
 * Author & Date:
 *                                          Baran Yalcin, 09.06.2014
 *
 * Description:
 *                        This program generates the required sawtooth
 *                        waveform for use with MicroMax 673 series m-
 *                        otor control board. The required waveform is
 *                        generated via PWM timer. Oscillator is used
 *                        for system clock and it is set to 16 MHz.
 *                        The measured period of interrupt is found as
 *                        10 us. Therefore, the period of single
 *                        PWM pulse is set to 10us. On the other hand
 *                        PWM pulse has resolution of 0.083 us, which
 *                        gives 0-120 pulse width range.
 *
 * Aspect ratio can be found as:
 *                        aspect_ratio = Wave1_freq / Wave2_freq
 *
 *
 * PD0, PB7: PWM Outputs
 * PDO: (X) Higher Frequency
 * PB7: (Y) Lower Frequency
 * Frequency settings can be
 * changed via PWM_FREQUENCY
 */


#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "inc/hw_ints.h"
#include "driverlib/interrupt.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/debug.h"
```

```
#include "driverlib/pwm.h"
#include "driverlib/pin_map.h"
#include "inc/hw_gpio.h"
#include "driverlib/timer.h"
#include "driverlib/rom.h"

#define ISR_PERIOD 100.0
#define M 100
#define N 10
#define WAVE_FREQUENCY 1000
#define ASPECT_RATIO 1

volatile unsigned long ulCounter = 0;
volatile unsigned int uiResolution_PD0;
volatile float fRatio = ISR_PERIOD / M;
volatile float ufStepsize_PD0;
volatile unsigned int uiResolution_PB7;
volatile float ufStepsize_PB7;


void PWM1IntHandler( void )
{
      ROM_PWMGenIntClear(PWM1_BASE,PWM_GEN_0, PWM_INT_CNT_LOAD);

      ROM_PWMPulseWidthSet(PWM1_BASE, PWM_OUT_0,  (12 * fRatio) * (((ulCounter/N ) %
                           M) + 1));

      ROM_PWMPulseWidthSet(PWM0_BASE, PWM_OUT_1,  (12 * ISR_PERIOD/(M *
                           ASPECT_RATIO)) * (((ulCounter/1000 ) % (M *
                           ASPECT_RATIO))  + 1));


      //ROM_GPIOPinWrite(GPIO_PORTD_BASE, GPIO_PIN_2, 255*(ulCounter % 2));

      ulCounter = ulCounter + 1;
}


int main(void)
{
      unsigned int uiPWMPeriod;
      //
      //     PWM period is necessarry for PWMGenPeriodSet.
      //     It should be found by ISR_PERIOD/0.0625 us.
      //     Here, ISR_PERIOD should be observed with test
      //     equipment and written in definitions
      //     (in microSeconds).
      //
      uiPWMPeriod = (ISR_PERIOD/0.0625);

      //
      //
      //
```

```
uiResolution_PD0 = 1/(0.000001 * ISR_PERIOD * WAVE_FREQUENCY) - 1;
//ufStepsize_PD0 = 120.0/uiResolution_PD0;
uiResolution_PB7 = 1/(0.000001 * ISR_PERIOD * WAVE_FREQUENCY)/ASPECT_RATIO ;

//ufStepsize_PB7 = 120.0/uiResolution_PB7;


//
//     System clock is set to 16 MHz. Main oscillator is used.
//     PWM clock is also set to 16 MHz, giving 62.5 ns clock period.
//
ROM_SysCtlClockSet(SYSCTL_SYSDIV_1|SYSCTL_USE_OSC|SYSCTL_OSC_MAIN|SYSCTL_XTAL_1
                    6MHZ);
ROM_SysCtlPWMClockSet(SYSCTL_PWMDIV_1);

//
//     Peripherals are enabled for PWM0 and PWM1 (PD0 & PB7)
//     and GPIO (PD2) can be used for debugging purposes.
//
ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM0);
ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_PWM1);
ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
ROM_SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOB);

//
//     Direction of PD2 is set as output.
//     Pins (PD0 & PB7) are set for PWM signals.
//     Configurations are set for PWM.
//
ROM_GPIOPinTypeGPIOOutput(GPIO_PORTD_BASE, GPIO_PIN_2);
ROM_GPIOPinTypePWM(GPIO_PORTD_BASE, GPIO_PIN_0);
ROM_GPIOPinTypePWM(GPIO_PORTB_BASE, GPIO_PIN_7);
ROM_GPIOPinConfigure(GPIO_PD0_M1PWM0);
ROM_GPIOPinConfigure(GPIO_PB7_M0PWM1);

//
//     PWM generator configurations are done. Since it is
//     a asymmetric signal, PWM_GEN_MODE_DOWN is selected.
//     PWN_GEN_MODE_GEN_SYNC_GLOBAL done for 2 PWMs.
//
ROM_PWMGenConfigure(PWM0_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN );
ROM_PWMGenConfigure(PWM1_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN );
//ROM_PWMGenConfigure(PWM0_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN |
                      PWM_GEN_MODE_GEN_SYNC_GLOBAL);
//ROM_PWMGenConfigure(PWM1_BASE, PWM_GEN_0, PWM_GEN_MODE_DOWN |
                       PWM_GEN_MODE_GEN_SYNC_GLOBAL);
```

```
        //
        //      Minimum PWM period is set as, Tau_ISR / 62.5ns.
        //      Tau_ISR value is measured by an oscilloscope or
        //      logic analyzer.     Since every Tau_ISR was measured
        //      as 10 us. It is set as 160.
        //      Since we use 2 PWM outputs with different
        //      aspect ratios, both must be set to same period.
        //
        ROM_PWMGenPeriodSet(PWM0_BASE, PWM_GEN_0, uiPWMPeriod*10);
        ROM_PWMGenPeriodSet(PWM1_BASE, PWM_GEN_0, uiPWMPeriod);

        //
        //      PWM output states are both enabled.
        //      PWM generators are both enabled.
        //
        ROM_PWMOutputState(PWM1_BASE, PWM_OUT_0_BIT, true);
        ROM_PWMOutputState(PWM0_BASE, PWM_OUT_1_BIT, true);
        ROM_PWMGenEnable(PWM1_BASE, PWM_GEN_0);
        ROM_PWMGenEnable(PWM0_BASE, PWM_GEN_0);

        //
        //      For interrupt to occur, first it should be
        //      enabled by IntMasterEnable. Then, we can
        //      place PWMIntEnable for PWM1. Trigger is set
        //      for PWM_INT_CNT_LOAD. And finally it can be
        //      enabled. The ISR can be found as PWM1IntHandler
        //      in startup_css.c of the project.
        //
        ROM_IntMasterEnable();
        ROM_PWMIntEnable(PWM1_BASE, PWM_INT_GEN_0);
        ROM_PWMGenIntTrigEnable(PWM1_BASE, PWM_GEN_0, PWM_INT_CNT_LOAD);
        ROM_IntEnable(INT_PWM1_0);

        //
        //      Do nothing in the while loop.
        //
        while(1) {
                //SysCtlDelay(1);
        }

}
```

2. Firmware of MCU2:

```
/*
 * KUNRL Confocal Project:
 *                        DAQ Software for Confocal Project
 *
 * Author & Date:
 *                        Baran Yalcin, 09.06.2014
 *
 * Description:
 *                        This program collects PMT and motor position
 *                        signals, and transfers them through UART.
 *                        UART speed is 115200 Baud/s.
 *                        ADC speed is 1 MSPS.
 *
 * PE3-PE2: Y input
 * PD3-PD2: X input
 * PD1-PD0: PMT input
 */


#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/sysctl.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/timer.h"
#include "driverlib/adc.h"
#include "driverlib/pin_map.h"
#include "utils/uartstdio.h"

unsigned long ulADC0_Value[3];

void ADCIntHandler(void)
{
    ADCProcessorTrigger(ADC0_BASE, 2);

        ADCSequenceDataGet(ADC0_BASE, 2, ulADC0_Value);

        //     PE3-PE2; PD3-PD2; PD1-PD0;
        UARTprintf("%4d %4d %4d\n", ulADC0_Value[0], ulADC0_Value[1], ulADC0_Value[2]);
        ADCIntClear(ADC0_BASE, 2);
}
```

```
void InitConsole(void)
{
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);


    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);
    UARTStdioInit(0);
}

int main(void)
{
    SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
                   SYSCTL_XTAL_16MHZ);

    // Begin UART
    InitConsole();

    // Set pins, peripherals
    SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);

    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);

    GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_2 | GPIO_PIN_3);
    GPIOPinTypeADC(GPIO_PORTD_BASE, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 |
                   GPIO_PIN_3);

    // Set ADC speed and configurations
    SysCtlADCSpeedSet(SYSCTL_ADCSPEED_1MSPS);
    ADCHardwareOversampleConfigure(ADC0_BASE, 64);
    ADCSequenceDisable(ADC0_BASE, 2);
    ADCSequenceConfigure(ADC0_BASE, 2, ADC_TRIGGER_PROCESSOR, 0);

    // PE3 - PE2
    ADCSequenceStepConfigure(ADC0_BASE, 2, 0, ADC_CTL_D | ADC_CTL_CH0);

    // PD3 - PD2
    ADCSequenceStepConfigure(ADC0_BASE, 2, 1, ADC_CTL_D | ADC_CTL_CH2);

    // PD1 - PD0
    ADCSequenceStepConfigure(ADC0_BASE, 2, 2, ADC_CTL_D | ADC_CTL_CH3 | ADC_CTL_IE |
                             ADC_CTL_END);

    // Enable interrupts
    ADCSequenceEnable(ADC0_BASE, 2);

    ADCIntClear(ADC0_BASE, 2);
    ADCIntEnable(ADC0_BASE, 2);
    IntEnable(INT_ADC0SS2);     // Turn on Sequence Interrupts fo Seq. 1
    IntMasterEnable();          // Finally -- enable all interrupts and go.

    while(1)
    {
        // Do nothing
    }
}
```

3. Python script of the prototype:

```
# data acquisition code
# author: Baran Yalcin
# this script gets the
# UART data, parses
# and saves it to .txt files

import time
import serial
import os

# ========CONTROL========
# number of samples taken
sample_number = 20001
# write to file 0/1
write_to_file = 1
# ========CONTROL========

#ser_data for storing the data transfer
#x_data   for storing Galvo_X_POS
#y_data   for storing Galvo_Y_POS
#img_data for storing only PMT data
ser_data = []
x_data   = []
y_data   = []
pmt_data = []

#connect to COM3 w/ baud=115200, timeout=1
ser = serial.Serial(2, 115200, timeout=1)

#for calculating time elapsed, start a timer
start = time.clock()

#get 10k samples, write to ser_data
for i in range(sample_number):
    ser_data.append(ser.readline())

#unpacking ser_data to x_data, y_data, img_data
for k in range(1,sample_number):
    y_data.append(ser_data[k].split()[0])
    x_data.append(ser_data[k].split()[1])
    pmt_data.append(ser_data[k].split()[2])

#calculate the elapsed time for process
end = time.clock()
print "Elapsed time is: " , (end - start)

#close the serial port
ser.close()
```

```
#for writing to a file(optional)
if write_to_file:

    #create a directory for storing txt files
    os.mkdir("./ok_run3")



    #change dir and create files
    os.chdir("./ok_run3")
    x_file   = open("x_data.txt", "w")
    y_file   = open("y_data.txt", "w")
    pmt_file = open("pmt_data.txt", "w")

    #write to files
    for item in range(sample_number-1):
        y_file.write("%s\n" % x_data[item])
        x_file.write("%s\n" % y_data[item])
        pmt_file.write("%s\n" % pmt_data[item])

    x_file.close()
    y_file.close()
    pmt_file.close()

    #get back to old dir
    os.chdir("..")    os.chdir("..")
```

4. MATLAB script of the prototype:

```
%
%image formation script:
%
%author: Baran Yalcin
%
%this  code  generates  image  using  x,  y  and  pmt  readings.  (position  refinement:
applied!)
%
%1. define an interval of full-scan(lim_beg and lim_end)
%2. clip the given x, y, pmt arrays
%3. normalize x and y arrays ( [a,b] --> [0, b-a] )
%4. adjust the shifting for x array (use debug lines for seeing traces)
%5. form the image
%6. use pcolor to plot image (use debug lines for pmt readings)

close all
clear all

%load the data in the /run dir
load x_data.txt;
load y_data.txt;
load pmt_data.txt;

%define lim_beg, lim_end
lim_beg = 2500;
lim_end = 8745;

%number of samples
sample_no = lim_end - lim_beg;
sample_vector = [1 : sample_no + 1];

%clipping
x_clipped   = x_data(   lim_beg : lim_end );
y_clipped   = y_data(   lim_beg : lim_end );
pmt_clipped = pmt_data( lim_beg : lim_end );

%normalization
x_normalized = x_clipped - (min(x_clipped) - 1).*ones( length( x_clipped ), 1 );
y_normalized = y_clipped - (min(y_clipped) - 1).*ones( length( y_clipped ), 1 );

k = 1;
while k <= sample_no + 1
    x_shifted(k) = x_normalized(k);
    k = k + 1;
end
```

```
%normalize and scale
%x_shifted = (max(y_normalized)/max(x_shifted))*(-1*min(x_shifted) + x_shifted);

% ===== DEBUG LINES FOR TRACES =====
plot(sample_vector, floor(x_shifted), '-r');
hold on
plot(sample_vector, y_normalized, '-b')
figure
% ===== DEBUG LINES FOR TRACES =====

%create an empty image matrix
%img = zeros( max( y_normalized ), max( x_normalized ));

%form the image
i = 1;
while i < length(pmt_clipped)
    %img(floor(x_shifted(i))+1, y_normalized(i)) = pmt_clipped(i);
    img(x_normalized(i), y_normalized(i)) = pmt_clipped(i);
    i = i + 1;
end

pcolor(img)
shading flat

% ===== DEBUG LINES FOR PMT DATA =====
figure
plot(sample_vector, pmt_clipped, '-or')
% ===== DEBUG LINES FOR PMT DATA =====

%for grayscale
figure
I = mat2gray(img, [2600 2600]);
imshow(I)
```

5. Firmware of CDAQ:

5.1. main.c

```
/*
 * KUNRL Confocal Project:
 *                                                  main.c of CDAQ Firmware
 *
 * Author & Date:
 *                                                  Baran Yalcin, 14.10.2014
 *
 */

#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/fpu.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/systick.h"
#include "driverlib/timer.h"
#include "driverlib/uart.h"
#include "driverlib/rom.h"
#include "usblib/usblib.h"
#include "usblib/usb-ids.h"
#include "usblib/device/usbdevice.h"
#include "usblib/device/usbdbulk.h"
#include "utils/uartstdio.h"
#include "utils/ustdlib.h"
#include "usb_bulk_structs.h"
#include "usb_functions.h"
#include "driverlib/ssi.h"
#include "driverlib/adc.h"

int main(void)
{
    volatile uint32_t ui32Loop;
    uint32_t ui32TxCount;
    uint32_t ui32RxCount;

    FPULazyStackingEnable();
```

```
        // Set clock to run from the PLL at 200MHz
        SysCtlClockSet(SYSCTL_SYSDIV_1 | SYSCTL_USE_PLL | SYSCTL_OSC_MAIN |
                       SYSCTL_XTAL_16MHZ);


        // Open UART0 and show the application name on the UART.
        ConfigureUART();

        // Enable the GPIO peripheral used for USB, and configure the USB pins.
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);
        GPIOPinTypeUSBAnalog(GPIO_PORTD_BASE, GPIO_PIN_4 | GPIO_PIN_5);

        ConfigureUSB();

        // Configure ADCs
        SysCtlPeripheralEnable(SYSCTL_PERIPH_ADC0);

        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOE);
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOD);

        GPIOPinTypeADC(GPIO_PORTE_BASE, GPIO_PIN_2 | GPIO_PIN_3);
        GPIOPinTypeADC(GPIO_PORTD_BASE, GPIO_PIN_0 | GPIO_PIN_1 | GPIO_PIN_2 |
                       GPIO_PIN_3);

        ADCHardwareOversampleConfigure(ADC0_BASE, 8);
        ADCSequenceDisable(ADC0_BASE, 2);
        ADCSequenceConfigure(ADC0_BASE, 2, ADC_TRIGGER_PROCESSOR, 0);

        ADCSequenceStepConfigure(ADC0_BASE, 2, 0, ADC_CTL_D | ADC_CTL_CH0);
                                                // PE3 - PE2

        ADCSequenceStepConfigure(ADC0_BASE, 2, 1, ADC_CTL_D | ADC_CTL_CH2);
                                                // PD3 - PD2

        ADCSequenceStepConfigure(ADC0_BASE, 2, 2, ADC_CTL_D | ADC_CTL_CH3 | ADC_CTL_IE
                                 | ADC_CTL_END);
                                                // PD1 - PD0

        ADCSequenceEnable(ADC0_BASE, 2);
        ADCIntClear(ADC0_BASE, 2);

        // Configure SSI
        // Initialize SSI0 and SSI1
        SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI0);
        SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI1);

        // Initialize PORTA for SSI0 and PORTF for SSI1
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
        SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);
        // Configure SSI0 pins
        GPIOPinConfigure(GPIO_PA2_SSI0CLK);
        GPIOPinConfigure(GPIO_PA3_SSI0FSS);
        GPIOPinConfigure(GPIO_PA5_SSI0TX);
        GPIOPinTypeSSI(GPIO_PORTA_BASE,GPIO_PIN_5|GPIO_PIN_3|GPIO_PIN_2);

        // Configure SSI1 pins
        GPIOPinConfigure(GPIO_PF2_SSI1CLK);
        GPIOPinConfigure(GPIO_PF3_SSI1FSS);
        GPIOPinConfigure(GPIO_PF1_SSI1TX);
        GPIOPinTypeSSI(GPIO_PORTF_BASE,GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1);

        // Configure and enable SSI0
        SSIConfigSetExpClk(SSI0_BASE, SysCtlClockGet(),    SSI_FRF_MOTO_MODE_1,
                         SSI_MODE_MASTER, 10000, 8);
        SSIEnable(SSI0_BASE);

        // Configure and enable SSI1
        SSIConfigSetExpClk(SSI1_BASE, SysCtlClockGet(),    SSI_FRF_MOTO_MODE_1,
                          SSI_MODE_MASTER, 10000, 8);
        SSIEnable(SSI1_BASE);



        // Clear our local byte counters.
        ui32RxCount = 0;
        ui32TxCount = 0;

        // Main application loop.
        while(1)
        {
        }
}
```

5.2. scan_functions.c

```
/*
 * KUNRL Confocal Project:
 *                                            scan_functions.c of CDAQ Firmware
 *
 * Author & Date:
 *                                            Baran Yalcin, 14.10.2014
 *
 */
#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/fpu.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/systick.h"
#include "driverlib/timer.h"
#include "driverlib/uart.h"
#include "driverlib/rom.h"
#include "usblib/usblib.h"
#include "usblib/usb-ids.h"
#include "usblib/device/usbdevice.h"
#include "usblib/device/usbdbulk.h"
#include "utils/uartstdio.h"
#include "utils/ustdlib.h"
#include "usb_bulk_structs.h"
#include "usb_functions.h"
#include "driverlib/ssi.h"
#include "driverlib/adc.h"
#include "driverlib/usb.h"

uint32_t ui32ADC0_Value[3];
uint8_t ui32RxQueue[64];
bool gbLAST = 0;

// Performs scan operation for sawtooth waveform
void scan_sawtooth(uint16_t ui16X_BEG, uint16_t ui16X_END, uint16_t ui16X_RES,
                   uint16_t ui16Y_BEG, uint16_t ui16Y_END, uint16_t ui16Y_RES,
                   uint16_t ui16DAMP)
{
       uint16_t ui16X_NOW = ui16X_BEG;
       uint16_t ui16Y_NOW = ui16Y_BEG;
       uint32_t ui32Round = 0;
       uint16_t ui16Damping = ui16DAMP;
       uint8_t ui8Count;
	while (ui16Y_NOW <= ui16Y_END) {
		ui8Count = 0;
		while(ui16X_NOW <= ui16X_END) {

			/*
			if(ui16Y_NOW == ui16Y_END && ui16X_NOW == ui16X_END)
			{
				gbLAST = 1;
			}
			*/
			move(ui16X_NOW, ui16Y_NOW);



			if( ui32Round >= 60)
			{
				// If full, send and flush
				send(ui32RxQueue, 60);
				ui32Round = 0;
			}
			sample(ui32Round);
			ui32Round += 12;

			ui16X_NOW += ui16X_RES;
		}

		while(ui8Count < ui16Damping)
		{
			ui8Count++;
			move(ui16X_END-ui8Count*(ui16X_END-ui16X_BEG)/ui16Damping,
			     ui16Y_NOW);
		}
		ui16X_NOW = ui16X_BEG;
		ui16Y_NOW += ui16Y_RES;
	}
}

// Performs scan operation for sawtooth waveform
void scan_triangle(uint16_t ui16X_BEG, uint16_t ui16X_END, uint16_t ui16X_RES,
                   uint16_t ui16Y_BEG, uint16_t ui16Y_END, uint16_t ui16Y_RES)
{
	uint16_t ui16X_NOW = ui16X_BEG;
	uint16_t ui16Y_NOW = ui16Y_BEG;
	uint32_t ui32Round = 0;
	uint8_t ui8Count;
```

```
        while (ui16Y_NOW <= ui16Y_END) {

                while(ui16X_NOW < ui16X_END) {

                        move(ui16X_NOW, ui16Y_NOW);

                        if( ui32Round >= 60)
                        {
                                // If full, send and flush
                                send(ui32RxQueue, 60);
                                ui32Round = 0;
                        }
                        sample(ui32Round);
                        ui32Round += 12;
                        ui16X_NOW += ui16X_RES;
                }

                while(ui16X_NOW > ui16X_BEG){

                        move(ui16X_NOW, ui16Y_NOW);

                        if( ui32Round >= 60)
                        {
                                // If full, send and flush
                                send(ui32RxQueue, 60);
                                ui32Round = 0;
                        }
                        sample(ui32Round);
                        ui32Round += 12;
                        ui16X_NOW -= ui16X_RES;
                }

                        ui16X_NOW = ui16X_BEG;
                        ui16Y_NOW += ui16Y_RES;
        }

}


// Moves the spot to given place
void move(uint16_t ui16X_NOW, uint16_t ui16Y_NOW)
{
        int valX_high = (ui16X_NOW>>8) & 0x0000FF;
        int valX_low  = ui16X_NOW & 0x0000FF;

        int valY_high = (ui16Y_NOW>>8) & 0x0000FF;
        int valY_low  = ui16Y_NOW & 0x0000FF;

        while(SSIBusy(SSI0_BASE));
        SSIDataPut(SSI0_BASE, 0x00);
        SSIDataPut(SSI0_BASE, valX_high);
        SSIDataPut(SSI0_BASE, valX_low);
```

```
       while(SSIBusy(SSI1_BASE));
       SSIDataPut(SSI1_BASE, 0x00);
       SSIDataPut(SSI1_BASE, valY_high);
       SSIDataPut(SSI1_BASE, valY_low);
}

void sample(uint32_t ui32Round)
{
    // Trigger the ADC conversion.
    ADCProcessorTrigger(ADC0_BASE, 2);
    // Wait for conversion to be completed.
    while(!ADCIntStatus(ADC0_BASE, 2, false));
    // Clear the ADC interrupt flag.
    ADCIntClear(ADC0_BASE, 2);
    // Read ADC Value.
    ADCSequenceDataGet(ADC0_BASE, 2, ui32ADC0_Value);


    // Store to RX queue
    // Convert first int to 4 bytes
    ui32RxQueue[ui32Round]    = ui32ADC0_Value[0]/1000 + '0';
    ui32RxQueue[ui32Round+1]  = (ui32ADC0_Value[0]%1000)/100 + '0';
    ui32RxQueue[ui32Round+2]  = (ui32ADC0_Value[0]%100)/10 + '0';
    ui32RxQueue[ui32Round+3]  = (ui32ADC0_Value[0]%10) + '0';

    // Convert second int to 4 bytes
    ui32RxQueue[ui32Round+4]    = ui32ADC0_Value[1]/1000 + '0';
    ui32RxQueue[ui32Round+5]  = (ui32ADC0_Value[1]%1000)/100 + '0';
    ui32RxQueue[ui32Round+6]  = (ui32ADC0_Value[1]%100)/10 + '0';
    ui32RxQueue[ui32Round+7]  = (ui32ADC0_Value[1]%10) + '0';

    // Convert third int to 4 bytes
    ui32RxQueue[ui32Round+8]    = ui32ADC0_Value[2]/1000 + '0';
    ui32RxQueue[ui32Round+9]  = (ui32ADC0_Value[2]%1000)/100 + '0';
    ui32RxQueue[ui32Round+10]  = (ui32ADC0_Value[2]%100)/10 + '0';
    ui32RxQueue[ui32Round+11]  = (ui32ADC0_Value[2]%10) + '0';

}

void send(uint8_t *pui8Data, uint32_t ui32NumBytes)
{
       int8_t USB_status;

       ui32RxQueue[60] = 'O';
       ui32RxQueue[61] = 'K';
       ui32RxQueue[62] = 'O';
       ui32RxQueue[63] = 'K';

    USB_status = USBEndpointDataPut(USB0_BASE, USB_EP_1, ui32RxQueue, 64);

    while (USB_status == -1) {
       USB_status = USBEndpointDataPut(USB0_BASE, USB_EP_1,ui32RxQueue, 64);
    }
    USB_status = USBEndpointDataSend(USB0_BASE, USB_EP_1,USB_TRANS_OUT);

    while (USB_status == -1) {
       USB_status = USBEndpointDataSend(USB0_BASE, USB_EP_1,USB_TRANS_OUT);
    }
}
```

5.3. scan_functions.h

```
/*
 * KUNRL Confocal Project:
 *                                            scan_functions.h of CDAQ Firmware
 *
 * Author & Date:
 *                                            Baran Yalcin, 14.10.2014
 *
 */

#ifndef SCAN_FUNCTIONS_H_
#define SCAN_FUNCTIONS_H_


extern uint32_t ui32ADC0_Value[3];
extern uint8_t ui32RxQueue[1200];
extern void scan_sawtooth(uint16_t ui16X_BEG, uint16_t ui16X_END,
             uint16_t ui16X_RES, uint16_t ui16Y_BEG,
             uint16_t ui16Y_END, uint16_t ui16Y_RES, uint16_t ui16DAMP);
extern void scan_triangle(uint16_t ui16X_BEG, uint16_t ui16X_END,
             uint16_t ui16X_RES, uint16_t ui16Y_BEG,
             uint16_t ui16Y_END, uint16_t ui16Y_RES);
extern void move(uint16_t ui16X_POS, uint16_t ui16Y_POS);
extern void sample(uint32_t ui32Round);
extern void send(uint8_t *pui8Data, uint32_t ui32NumBytes);
#endif /* SCAN_FUNCTIONS_H_ */
```

5.4. usb_bulk_structs.c

```
/*
 * KUNRL Confocal Project:
 *                                          usb_bulk_structs.c of CDAQ Firmware
 *
 * Author & Date:
 *                                          Baran Yalcin, 14.10.2014
 *
 */

#include <stdint.h>
#include <stdbool.h>
#include "inc/hw_types.h"
#include "driverlib/usb.h"
#include "usblib/usblib.h"
#include "usblib/usb-ids.h"
#include "usblib/device/usbdevice.h"
#include "usblib/device/usbdbulk.h"
#include "usb_bulk_structs.h"

// The language supported by this device.
const uint8_t g_pui8LangDescriptor[] =
{
             4,
             USB_DTYPE_STRING,
             USBShort(USB_LANG_EN_US)
};

// The manufacturer string.
const uint8_t g_pui8ManufacturerString[] =
{
           (17 + 1) * 2,
           USB_DTYPE_STRING,
           'T', 0, 'e', 0, 'x', 0, 'a', 0, 's', 0, ' ', 0, 'I', 0, 'n', 0, 's', 0,
           't', 0, 'r', 0, 'u', 0, 'm', 0, 'e', 0, 'n', 0, 't', 0, 's', 0,
};

// The product string.
const uint8_t g_pui8ProductString[] =
{
           (19 + 1) * 2,
           USB_DTYPE_STRING,
           'G', 0, 'e', 0, 'n', 0, 'e', 0, 'r', 0, 'i', 0, 'c', 0, ' ', 0, 'B', 0,
           'u', 0, 'l', 0, 'k', 0, ' ', 0, 'D', 0, 'e', 0, 'v', 0, 'i', 0, 'c', 0,
           'e', 0
};
```

```
// The serial number string.
const uint8_t g_pui8SerialNumberString[] =
{
          (8 + 1) * 2,
          USB_DTYPE_STRING,
          '1', 0, '2', 0, '3', 0, '4', 0, '5', 0, '6', 0, '7', 0, '8', 0
};

// The data interface description string.
const uint8_t g_pui8DataInterfaceString[] =
{
          (19 + 1) * 2,
          USB_DTYPE_STRING,
          'B', 0, 'u', 0, 'l', 0, 'k', 0, ' ', 0, 'D', 0, 'a', 0, 't', 0,
          'a', 0, ' ', 0, 'I', 0, 'n', 0, 't', 0, 'e', 0, 'r', 0, 'f', 0,
          'a', 0, 'c', 0, 'e', 0
};

// The configuration description string.
const uint8_t g_pui8ConfigString[] =
{
          (23 + 1) * 2,
          USB_DTYPE_STRING,
          'B', 0, 'u', 0, 'l', 0, 'k', 0, ' ', 0, 'D', 0, 'a', 0, 't', 0,
          'a', 0, ' ', 0, 'C', 0, 'o', 0, 'n', 0, 'f', 0, 'i', 0, 'g', 0,
          'u', 0, 'r', 0, 'a', 0, 't', 0, 'i', 0, 'o', 0, 'n', 0
};

// The descriptor string table.
const uint8_t * const g_ppui8StringDescriptors[] =
{
          g_pui8LangDescriptor,
          g_pui8ManufacturerString,
          g_pui8ProductString,
          g_pui8SerialNumberString,
          g_pui8DataInterfaceString,
          g_pui8ConfigString
};

#define NUM_STRING_DESCRIPTORS (sizeof(g_ppui8StringDescriptors) / sizeof(uint8_t *))

// The bulk device initializatiın and customization structures.
extern const tUSBBuffer g_sTxBuffer;
extern const tUSBBuffer g_sRxBuffer;
```

```
tUSBDBulkDevice g_sBulkDevice =
{
             USB_VID_TI_1CBE,
             USB_PID_BULK,
             500,
             USB_CONF_ATTR_SELF_PWR,
             USBBufferEventCallback,
             (void *)&g_sRxBuffer,
             USBBufferEventCallback,
             (void *)&g_sTxBuffer,
             g_ppui8StringDescriptors,
             NUM_STRING_DESCRIPTORS
};

// Receive buffer(from the USB perspective).
uint8_t g_pui8USBRxBuffer[BULK_BUFFER_SIZE];
uint8_t g_pui8RxBufferWorkspace[USB_BUFFER_WORKSPACE_SIZE];
const tUSBBuffer g_sRxBuffer =
{
             false,
             RxHandler,
             (void *)&g_sBulkDevice,
             USBDBulkPacketRead,
             USBDBulkRxPacketAvailable,
             (void *)&g_sBulkDevice,
             g_pui8USBRxBuffer,
             BULK_BUFFER_SIZE,
             g_pui8RxBufferWorkspace
};

// Transmit buffer.
uint8_t g_pui8USBTxBuffer[BULK_BUFFER_SIZE];
uint8_t g_pui8TxBufferWorkspace[USB_BUFFER_WORKSPACE_SIZE];
const tUSBBuffer g_sTxBuffer =
{
             true,
             TxHandler,
             (void *)&g_sBulkDevice,
             USBDBulkPacketWrite,
             USBDBulkTxPacketAvailable,
             (void *)&g_sBulkDevice,
             g_pui8USBTxBuffer,
             BULK_BUFFER_SIZE,
             g_pui8TxBufferWorkspace
};
```

5.5. usb_bulk_structs.h

```
/*
 * KUNRL Confocal Project:
 *                                            usb_bulk_structs.h of CDAQ Firmware
 *
 * Author & Date:
 *                                            Baran Yalcin, 14.10.2014
 *
 */

#ifndef _USB_BULK_STRUCTS_H_
#define _USB_BULK_STRUCTS_H_


#define BULK_BUFFER_SIZE 64

extern uint32_t RxHandler(void *pvCBData, uint32_t ui32Event,
                          uint32_t ui32MsgValue, void *pvMsgData);
extern uint32_t TxHandler(void *pvi32CBData, uint32_t ui32Event,
                          uint32_t ui32MsgValue, void *pvMsgData);

extern const tUSBBuffer g_sTxBuffer;
extern const tUSBBuffer g_sRxBuffer;
extern tUSBDBulkDevice g_sBulkDevice;
extern uint8_t g_pui8USBTxBuffer[];
extern uint8_t g_pui8USBRxBuffer[];

#endif
```

5.6. usb_functions.c

```
/*
 * KUNRL Confocal Project:
 *                                              usb_functions.c of CDAQ Firmware
 *
 * Author & Date:
 *                                              Baran Yalcin, 14.10.2014
 *
 */

/* Define all USB related functions/interrupts here */

#include <stdbool.h>
#include <stdint.h>
#include "inc/hw_ints.h"
#include "inc/hw_memmap.h"
#include "inc/hw_types.h"
#include "driverlib/debug.h"
#include "driverlib/fpu.h"
#include "driverlib/gpio.h"
#include "driverlib/interrupt.h"
#include "driverlib/pin_map.h"
#include "driverlib/sysctl.h"
#include "driverlib/systick.h"
#include "driverlib/timer.h"
#include "driverlib/uart.h"
#include "driverlib/rom.h"
#include "usblib/usblib.h"
#include "usblib/usb-ids.h"
#include "usblib/device/usbdevice.h"
#include "usblib/device/usbdbulk.h"
#include "utils/uartstdio.h"
#include "utils/ustdlib.h"
#include "usb_bulk_structs.h"
#include "scan_functions.h"
#include "driverlib/ssi.h"


// Keep track of receive and transmit counts.
volatile uint32_t g_ui32TxCount = 0;
volatile uint32_t g_ui32RxCount = 0;

// Flags for passing commands from interrupt context to the main loop.
#define COMMAND_PACKET_RECEIVED 0x00000001
#define COMMAND_STATUS_UPDATE    0x00000002

volatile uint32_t g_ui32Flags = 0;
```

```
// Flag for USB configuration has been set.
static volatile bool g_bUSBConfigured = false;

void ReconfigureSSI(uint32_t ui32SPEED){
	// Configure SSI
	// Initialize SSI0 and SSI1
	SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI0);
	SysCtlPeripheralEnable(SYSCTL_PERIPH_SSI1);

	// Initialize PORTA for SSI0 and PORTF for SSI1
	SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);
	SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOF);

	// Configure SSI0 pins
	GPIOPinConfigure(GPIO_PA2_SSI0CLK);
	GPIOPinConfigure(GPIO_PA3_SSI0FSS);
	GPIOPinConfigure(GPIO_PA5_SSI0TX);
	GPIOPinTypeSSI(GPIO_PORTA_BASE,GPIO_PIN_5|GPIO_PIN_3|GPIO_PIN_2);

	// Configure SSI1 pins
	GPIOPinConfigure(GPIO_PF2_SSI1CLK);
	GPIOPinConfigure(GPIO_PF3_SSI1FSS);
	GPIOPinConfigure(GPIO_PF1_SSI1TX);
	GPIOPinTypeSSI(GPIO_PORTF_BASE,GPIO_PIN_3|GPIO_PIN_2|GPIO_PIN_1);

	// Configure and enable SSI0
	SSIConfigSetExpClk(SSI0_BASE, SysCtlClockGet(),    SSI_FRF_MOTO_MODE_1,
	                   SSI_MODE_MASTER, ui32SPEED, 8);
	SSIEnable(SSI0_BASE);

	// Configure and enable SSI1
	SSIConfigSetExpClk(SSI1_BASE, SysCtlClockGet(),    SSI_FRF_MOTO_MODE_1,
	                  SSI_MODE_MASTER, ui32SPEED, 8);
	SSIEnable(SSI1_BASE);
}


// Parses X_BEG, X_END, X_RES, Y_BEG, Y_END, Y_RES from received data
static void ParseData(tUSBDBulkDevice *psDevice, uint8_t *pui8Data, uint32_t
ui32NumBytes)
{
	uint16_t ui16X_BEG, ui16X_END, ui16X_RES, ui16Y_BEG, ui16Y_END, ui16Y_RES,
	         ui16DAMP;
	uint32_t ui32SPEED;

	uint32_t ui32ReadIndex;

	// Update receive counter.
	g_ui32RxCount += ui32NumBytes;
```

```
// Set up to process the characters by directly accessing the USB buffers.
ui32ReadIndex = (uint32_t)(pui8Data - g_pui8USBRxBuffer);
tUSBRingBufObject sTxRing;

USBBufferInfoGet(&g_sTxBuffer, &sTxRing);

if(g_pui8USBRxBuffer[ui32ReadIndex] == 'S') {
       ui32ReadIndex++;
       // First for X
       ui16X_BEG = 10000*(g_pui8USBRxBuffer[ui32ReadIndex] - '0')
                    +1000*(g_pui8USBRxBuffer[ui32ReadIndex+1] - '0')
                    +100*(g_pui8USBRxBuffer[ui32ReadIndex+2] - '0')
                    +10*(g_pui8USBRxBuffer[ui32ReadIndex+3] - '0')
                    +1*(g_pui8USBRxBuffer[ui32ReadIndex+4] - '0');

       ui16X_END = 10000*(g_pui8USBRxBuffer[ui32ReadIndex+6] - '0')
                    +1000*(g_pui8USBRxBuffer[ui32ReadIndex+7] - '0')
                    +100*(g_pui8USBRxBuffer[ui32ReadIndex+8] - '0')
                    +10*(g_pui8USBRxBuffer[ui32ReadIndex+9] - '0')
                    +1*(g_pui8USBRxBuffer[ui32ReadIndex+10] - '0');

       ui16X_RES = 10000*(g_pui8USBRxBuffer[ui32ReadIndex+12] - '0')
                    +1000*(g_pui8USBRxBuffer[ui32ReadIndex+13] - '0')
                    +100*(g_pui8USBRxBuffer[ui32ReadIndex+14] - '0')
                    +10*(g_pui8USBRxBuffer[ui32ReadIndex+15] - '0')
                    +1*(g_pui8USBRxBuffer[ui32ReadIndex+16] - '0');

       // Then for Y
       ui16Y_BEG = 10000*(g_pui8USBRxBuffer[ui32ReadIndex+18] - '0')
                    +1000*(g_pui8USBRxBuffer[ui32ReadIndex+19] - '0')
                    +100*(g_pui8USBRxBuffer[ui32ReadIndex+20] - '0')
                    +10*(g_pui8USBRxBuffer[ui32ReadIndex+21] - '0')
                    +1*(g_pui8USBRxBuffer[ui32ReadIndex+22] - '0');

       ui16Y_END = 10000*(g_pui8USBRxBuffer[ui32ReadIndex+24] - '0')
                    +1000*(g_pui8USBRxBuffer[ui32ReadIndex+25] - '0')
                    +100*(g_pui8USBRxBuffer[ui32ReadIndex+26] - '0')
                    +10*(g_pui8USBRxBuffer[ui32ReadIndex+27] - '0')
                    +1*(g_pui8USBRxBuffer[ui32ReadIndex+28] - '0');

       ui16Y_RES = 10000*(g_pui8USBRxBuffer[ui32ReadIndex+30] - '0')
                    +1000*(g_pui8USBRxBuffer[ui32ReadIndex+31] - '0')
                    +100*(g_pui8USBRxBuffer[ui32ReadIndex+32] - '0')
                    +10*(g_pui8USBRxBuffer[ui32ReadIndex+33] - '0')
                    +1*(g_pui8USBRxBuffer[ui32ReadIndex+34] - '0');
```

```
        ui32SPEED = 1000000*(g_pui8USBRxBuffer[ui32ReadIndex+36] - '0')
                     +100000*(g_pui8USBRxBuffer[ui32ReadIndex+37] - '0')
                     +10000*(g_pui8USBRxBuffer[ui32ReadIndex+38] - '0')
                     +1000*(g_pui8USBRxBuffer[ui32ReadIndex+39] - '0')
                     +100*(g_pui8USBRxBuffer[ui32ReadIndex+40] - '0')
                     +10*(g_pui8USBRxBuffer[ui32ReadIndex+41] - '0')
                     +1*(g_pui8USBRxBuffer[ui32ReadIndex+42] - '0');


        ui16DAMP = 100*(g_pui8USBRxBuffer[ui32ReadIndex+44] - '0')
                        +10*(g_pui8USBRxBuffer[ui32ReadIndex+45] - '0')
                            +1*(g_pui8USBRxBuffer[ui32ReadIndex+46] - '0');

        ReconfigureSSI(ui32SPEED);
        scan_sawtooth(ui16X_BEG, ui16X_END, ui16X_RES, ui16Y_BEG, ui16Y_END,
                      ui16Y_RES , ui16DAMP);
}

if(g_pui8USBRxBuffer[ui32ReadIndex] == 'T') {
        ui32ReadIndex++;
        // First for X
        ui16X_BEG = 10000*(g_pui8USBRxBuffer[ui32ReadIndex] - '0')
                     +1000*(g_pui8USBRxBuffer[ui32ReadIndex+1] - '0')
                     +100*(g_pui8USBRxBuffer[ui32ReadIndex+2] - '0')
                     +10*(g_pui8USBRxBuffer[ui32ReadIndex+3] - '0')
                     +1*(g_pui8USBRxBuffer[ui32ReadIndex+4] - '0');

        ui16X_END = 10000*(g_pui8USBRxBuffer[ui32ReadIndex+6] - '0')
                     +1000*(g_pui8USBRxBuffer[ui32ReadIndex+7] - '0')
                     +100*(g_pui8USBRxBuffer[ui32ReadIndex+8] - '0')
                     +10*(g_pui8USBRxBuffer[ui32ReadIndex+9] - '0')
                     +1*(g_pui8USBRxBuffer[ui32ReadIndex+10] - '0');

        ui16X_RES = 10000*(g_pui8USBRxBuffer[ui32ReadIndex+12] - '0')
                     +1000*(g_pui8USBRxBuffer[ui32ReadIndex+13] - '0')
                     +100*(g_pui8USBRxBuffer[ui32ReadIndex+14] - '0')
                     +10*(g_pui8USBRxBuffer[ui32ReadIndex+15] - '0')
                     +1*(g_pui8USBRxBuffer[ui32ReadIndex+16] - '0');

        // Then for Y
        ui16Y_BEG = 10000*(g_pui8USBRxBuffer[ui32ReadIndex+18] - '0')
                     +1000*(g_pui8USBRxBuffer[ui32ReadIndex+19] - '0')
                     +100*(g_pui8USBRxBuffer[ui32ReadIndex+20] - '0')
                     +10*(g_pui8USBRxBuffer[ui32ReadIndex+21] - '0')
                     +1*(g_pui8USBRxBuffer[ui32ReadIndex+22] - '0');
```

```
            ui16Y_END = 10000*(g_pui8USBRxBuffer[ui32ReadIndex+24] - '0')
                         +1000*(g_pui8USBRxBuffer[ui32ReadIndex+25] - '0')
                         +100*(g_pui8USBRxBuffer[ui32ReadIndex+26] - '0')
                         +10*(g_pui8USBRxBuffer[ui32ReadIndex+27] - '0')
                         +1*(g_pui8USBRxBuffer[ui32ReadIndex+28] - '0');

            ui16Y_RES = 10000*(g_pui8USBRxBuffer[ui32ReadIndex+30] - '0')
                         +1000*(g_pui8USBRxBuffer[ui32ReadIndex+31] - '0')
                         +100*(g_pui8USBRxBuffer[ui32ReadIndex+32] - '0')
                         +10*(g_pui8USBRxBuffer[ui32ReadIndex+33] - '0')
                         +1*(g_pui8USBRxBuffer[ui32ReadIndex+34] - '0');

            ui32SPEED = 1000000*(g_pui8USBRxBuffer[ui32ReadIndex+36] - '0')
                         +100000*(g_pui8USBRxBuffer[ui32ReadIndex+37] - '0')
                         +10000*(g_pui8USBRxBuffer[ui32ReadIndex+38] - '0')
                         +1000*(g_pui8USBRxBuffer[ui32ReadIndex+39] - '0')
                         +100*(g_pui8USBRxBuffer[ui32ReadIndex+40] - '0')
                         +10*(g_pui8USBRxBuffer[ui32ReadIndex+41] - '0')
                         +1*(g_pui8USBRxBuffer[ui32ReadIndex+42] - '0');

            ReconfigureSSI(ui32SPEED);
            scan_triangle(ui16X_BEG, ui16X_END, ui16X_RES, ui16Y_BEG, ui16Y_END,
                         ui16Y_RES);
        }
}

// Receive new data and echo. Called by RxHandler.
static uint32_t EchoNewDataToHost(tUSBDBulkDevice *psDevice, uint8_t *pui8Data,
                                  uint32_t ui32NumBytes)
{
        uint32_t ui32Loop, ui32Space, ui32Count;
        uint32_t ui32ReadIndex;
        uint32_t ui32WriteIndex;
        tUSBRingBufObject sTxRing;

        USBBufferInfoGet(&g_sTxBuffer, &sTxRing);

        ui32Space = USBBufferSpaceAvailable(&g_sTxBuffer);
        ui32Loop = (ui32Space < ui32NumBytes) ? ui32Space : ui32NumBytes;
        ui32Count = ui32Loop;

        // Update receive counter.
        g_ui32RxCount += ui32NumBytes;

        // Set up to process the characters by directly accessing the USB buffers.
        ui32ReadIndex = (uint32_t)(pui8Data - g_pui8USBRxBuffer);
        ui32WriteIndex = sTxRing.ui32WriteIndex;
        while(ui32Loop) {

                //g_pui8USBTxBuffer[ui32WriteIndex] = g_pui8USBRxBuffer[ui32ReadIndex];
                // Unpack the scan elements here
                // Move to the next character taking care to adjust the pointer for
                // the buffer wrap if necessary.
                ui32WriteIndex++;
                ui32WriteIndex = (ui32WriteIndex == BULK_BUFFER_SIZE) ? 0 :
                ui32WriteIndex;

                ui32ReadIndex++;
                ui32ReadIndex = (ui32ReadIndex == BULK_BUFFER_SIZE) ? 0 : ui32ReadIndex;

                ui32Loop--;
        }
        // Send it back.
        USBBufferDataWritten(&g_sTxBuffer, ui32Count);
        return (ui32Count);
}

uint32_t TxHandler(void *pvCBData, uint32_t ui32Event, uint32_t ui32MsgValue, void
                   *pvMsgData)
{
        if(ui32Event == USB_EVENT_TX_COMPLETE) {
                g_ui32TxCount += ui32MsgValue;
        }
        return (0);
}

uint32_t RxHandler(void *pvCBData, uint32_t ui32Event, uint32_t ui32MsgValue, void
                   *pvMsgData)
{
        switch(ui32Event)
        {
                case USB_EVENT_CONNECTED:
                {
                        g_bUSBConfigured = true;

                        USBBufferFlush(&g_sTxBuffer);
                        USBBufferFlush(&g_sRxBuffer);
                        break;
                }

                case USB_EVENT_DISCONNECTED:
                {
                        g_bUSBConfigured = false;
                        break;
                }
```

```
            case USB_EVENT_RX_AVAILABLE:
            {
                  tUSBDBulkDevice *psDevice;
                  psDevice = (tUSBDBulkDevice *)pvCBData;
                  ParseData(psDevice, pvMsgData, ui32MsgValue);
                  //ParseData(psDevice, pvMsgData, ui32MsgValue);
                  //return(EchoNewDataToHost(psDevice, pvMsgData, ui32MsgValue));
            }

            case USB_EVENT_SUSPEND:
            case USB_EVENT_RESUME:
            {
                  break;
            }

            default:
            {
                  break;
            }
      }
      return (0);
}

void ConfigureUART(void)
{
    // Enable the GPIO Peripheral used by the UART.
    SysCtlPeripheralEnable(SYSCTL_PERIPH_GPIOA);

    // Enable UART0
    SysCtlPeripheralEnable(SYSCTL_PERIPH_UART0);

    // Configure GPIO Pins for UART mode.
    GPIOPinConfigure(GPIO_PA0_U0RX);
    GPIOPinConfigure(GPIO_PA1_U0TX);
    GPIOPinTypeUART(GPIO_PORTA_BASE, GPIO_PIN_0 | GPIO_PIN_1);

    // Use the internal 16MHz oscillator as the UART clock source.
    UARTClockSourceSet(UART0_BASE, UART_CLOCK_PIOSC);

    // Initialize the UART for console I/O.
    UARTStdioConfig(0, 115200, 16000000);
}

void ConfigureUSB(void)
{
      // Not configured initially.
      g_bUSBConfigured = false;
```

```
        // Initialize the transmit and receive buffers.
        USBBufferInit(&g_sTxBuffer);
        USBBufferInit(&g_sRxBuffer);

        // Set the USB stack mode to Device mode with VBUS monitoring.
        USBStackModeSet(0, eUSBModeForceDevice, 0);

        // Pass our device information to the USB library and place the device on the
bus.
        USBDBulkInit(0, &g_sBulkDevice);
}
```

5.7. usb_functions.h

```
/*
 * KUNRL Confocal Project:
 *                                              usb_functions.h of CDAQ Firmware
 *
 * Author & Date:
 *                                              Baran Yalcin, 14.10.2014
 *
 */

#ifndef USB_FUNCTIONS_H_
#define USB_FUNCTIONS_H_

#define COMMAND_PACKET_RECEIVED 0x00000001
#define COMMAND_STATUS_UPDATE    0x00000002


extern volatile uint32_t g_ui32TxCount;
extern volatile uint32_t g_ui32RxCount;

extern volatile uint32_t g_ui32Flags;
extern volatile bool g_bUSBConfigured;

extern uint32_t ParseData(tUSBDBulkDevice *psDevice, uint8_t *pui8Data, uint32_t
                          ui32NumBytes);
extern uint32_t EchoNewDataToHost(tUSBDBulkDevice *psDevice, uint8_t *pui8Data,
                                  uint32_t ui32NumBytes);
extern uint32_t TxHandler(void *pvCBData, uint32_t ui32Event, uint32_t ui32MsgValue,
                          void *pvMsgData);
extern uint32_t RxHandler(void *pvCBData, uint32_t ui32Event, uint32_t ui32MsgValue,
                          void *pvMsgData);
extern void ConfigureUART(void);
extern void ConfigureUSB(void);

#endif /* USB_FUNCTIONS_H_ */
```

6. Konfokal Source Code

6.1. ui.py

```
from __future__ import division
import os
import platform
import sys

from PyQt4.QtCore import *
from PyQt4.QtGui import *
from matplotlib.backends.backend_qt4agg import FigureCanvasQTAgg as FigureCanvas
from matplotlib.backends.backend_qt4agg import NavigationToolbar2QTAgg as
NavigationToolbar

import matplotlib.pyplot as plt
import matplotlib.image as mpimg
import numpy as np

import ui.qrc_resources as qrc_resources
import ui.newsessiondlg as newsessiondlg
import ui.newscanwzd as newscanwzd
import core, utils, dev

__version__ = "1.0.0"
__author__ = "Baran Yalcin"

class MainWindow(QMainWindow):

    def __init__(self, parent=None):
        super(MainWindow, self).__init__(parent)

        # sessionDict contains Sessions and their children(Scans)
        self.sessionDict = {}
        # Set windowtitle, icon and statusbar
        self.setWindowTitle("Konfokal")
        status = self.statusBar().showMessage("Ready")

        # Holds unsaved changes
        self.dirty = False
        self.filename = None
        self.directory = None

        # Selected item list
        self.selecteditem = []

        # Last scan: for quick scan
        self.lastscan = None

        self.currentImage = None
        self.currentPlot = None
        # Window properties/widgets
        navigatorDockWidget = QDockWidget()
        navigatorDockWidget.setAllowedAreas(Qt.LeftDockWidgetArea |
                                            Qt.RightDockWidgetArea)

        # Left pane navigation treeWidget
        self.navigatorWidget = QTreeWidget()
        self.navigatorWidget.setHeaderLabel("Navigator")
        navigatorDockWidget.setWidget(self.navigatorWidget)
        self.addDockWidget(Qt.LeftDockWidgetArea, navigatorDockWidget)

        # Create a tab widget for displaying plot/image
        # Set as central widget
        self.plot = plt.figure()
        self.plotCanvas = FigureCanvas(self.plot)
        self.image = plt.figure()
        self.imageCanvas = FigureCanvas(self.image)

        # Add plot toolbars
        self.plotToolbar = NavigationToolbar(self.plotCanvas, self)
        self.imageToolbar = NavigationToolbar(self.imageCanvas, self)


        vbox_img = QVBoxLayout()
        vbox_img.addWidget(self.imageCanvas)
        vbox_img.addWidget(self.imageToolbar)

        vbox_plt = QVBoxLayout()
        vbox_plt.addWidget(self.plotCanvas)
        vbox_plt.addWidget(self.plotToolbar)


        imageWidget = QWidget()
        imageWidget.setLayout(vbox_img)

        plotWidget = QWidget()
        plotWidget.setLayout(vbox_plt)


        self.tabWidget = QTabWidget()
        self.tabWidget.addTab(plotWidget, "Plot")
        self.tabWidget.addTab(imageWidget, "Image")



        #ax = self.plot.add_subplot(111)
        #ax.plot([1,2,3])
        #self.plotCanvas.print_figure('test')
        #self.plotCanvas.print_figure('plot30.png')

        #self.displayImage('plot30.png')
        self.setCentralWidget(self.tabWidget)


        # Create the menuBar, add menus
        self.fileMenu = self.menuBar().addMenu("&File")
        self.editMenu = self.menuBar().addMenu("&Edit")
        self.scanMenu = self.menuBar().addMenu("&Scan")
        self.helpMenu = self.menuBar().addMenu("&Help")

        # Create File Menu actions and add them
        fileNewSessionAction = self.createAction("&New Session",
                                                 self.fileNewSession,
                                                 QKeySequence.New,
                                                 "newsession",
                                                 "Create a new session")

        fileLoadSessionAction = self.createAction("&Load Session",
                                                   self.fileLoadSession,
                                                   "Ctrl+L",
                                                   "loadsession",
                                                   "Load a session")

        fileSaveSessionAction = self.createAction("&Save Session",
                                                   self.fileSaveSession,
                                                   QKeySequence.Save,
                                                   "savesession",
                                                   "Save the session")

        fileSaveAsSessionAction = self.createAction("&Save As...",
                                                     self.fileSaveAsSession,
                                                     "Ctrl+Shift+S",
                                                     "saveassession",
                                                     "Save as...")

        fileExportAction = self.createAction("&Export", self.fileExport,
                                             "Ctrl+E", "exportsession",
                                             "Export report/data/image")

        fileSettingsAction = self.createAction("&Settings", self.fileSettings,
                                               "Ctrl+O", "settings",
                                               "Konfokal settings")

        fileQuitAction = self.createAction("&Quit", self.close,
                                           "Ctrl+Q", "exit",
                                           "Close the application")

        self.fileMenuActions = (fileNewSessionAction, fileLoadSessionAction,
                                fileSaveSessionAction, fileSaveAsSessionAction,
                                fileExportAction, fileSettingsAction,
                                fileQuitAction)

        self.connect(self.fileMenu, SIGNAL("aboutToShow()"),
            lambda args=(self.fileMenu,self.fileMenuActions):
```

```
            self.updateMenu(args))

        # Create File Toolbar
        fileToolbar = self.addToolBar("File")
        self.addActions(fileToolbar, (fileNewSessionAction,
                                      fileLoadSessionAction,
                                      fileSaveSessionAction,
                                      fileExportAction,
                                      fileSettingsAction,
                                      fileQuitAction))

        # Create Edit actions and add them
        editUndoAction = self.createAction("&Undo", self.editUndo,
                                           "Ctrl+Z", "undo", "Undo")

        editRedoAction = self.createAction("&Redo", self.editRedo,
                                           "Ctrl+Y", "redo", "Redo")

        editManipulateAction = self.createAction("&Manipulate",
                                                 self.editManipulate,
                                                 "Ctrl+M", "manipulate",
                                                 "Manipulate")

        self.editMenuActions = (editUndoAction, editRedoAction,
                                editManipulateAction)

        self.connect(self.editMenu, SIGNAL("aboutToShow()"),
            lambda args=(self.editMenu,self.editMenuActions):
            self.updateMenu(args))

        # Create Edit Toolbar
        editToolbar = self.addToolBar("Edit")
        self.addActions(editToolbar, (editUndoAction, editRedoAction,
                                      editManipulateAction))


        # Create Scan actions and add them
        scanQuickScanAction = self.createAction("&Quick Scan", self.quickScan,
                                                "F5", "quickscan",
                                                "Perform Quick Scan")

        scanNewScanAction = self.createAction("&New Scan", self.newScan,
                                              "F6", "newscan",
                                              "Perform New Scan")

        self.scanMenuActions = (scanQuickScanAction, scanNewScanAction)

        self.connect(self.scanMenu, SIGNAL("aboutToShow()"),
            lambda args=(self.scanMenu,self.scanMenuActions):
            self.updateMenu(args))

        # Create Scan toolbar
        scanToolbar = self.addToolBar("Scan")
        self.addActions(scanToolbar, (scanNewScanAction, scanQuickScanAction))

        # Add help menu options
        helpKonfokalAboutAction = self.createAction("&About Konfokal",
                                              self.helpKonfokalAboutAction)
        helpKonfokalAction = self.createAction("&Help", self.helpKonfokalAction)
        self.helpMenuActions = (helpKonfokalAboutAction, helpKonfokalAction)

        self.connect(self.helpMenu, SIGNAL("aboutToShow()"),
            lambda args=(self.helpMenu,self.helpMenuActions):
            self.updateMenu(args))

        # Navigator signals go here
        self.connect(self.navigatorWidget, SIGNAL("itemSelectionChanged()"),
            self.navigatorSlot)


    def navigatorSlot(self):

        selectedscan = self.navigatorWidget.selectedItems()[-1]
        selectedtext = selectedscan.text(0)

        try:
            # Try if selected item has parent, if so add it to list
            selectedparent = selectedscan.parent()

            # Set last items font to normal first
            if self.selecteditem:
                lastitem = self.selecteditem.pop()
                normalfont = QFont(lastitem.text(0))
                normalfont.setBold(False)
                lastitem.setFont(0, normalfont)

            # Set its text to bold, if selected
            boldfont = QFont(selectedparent.text(0))
            boldfont.setBold(True)
            selectedparent.setFont(0, boldfont)
            # Set others' text to default
            self.selecteditem.append(selectedparent)

            selectedParentText = str(selectedparent.text(0))

            self.refresh(int(selectedParentText.split()[-1])-1,
                str(selectedtext))

        except AttributeError, e:
            print e

        return

    def refresh(self, sessionno, scantitle):
        "Refreshes image and plot in MainWidget."
        for scans in self.sessionDict[sessionno].scanlist:
            if scantitle == scans.data['TITLE']:
                # Check if image is available
                if scans.data['IMAGE'] is not core.ScanImage:

                    # Check if datatypes are constructed for X
                    if type(scans.data['X']) is not core.ScanData and  \
                        scans.data['X'] != 'N/A':
                        # If not, parse the data from the XML and
                        scans.data['X'] = core.ScanData('X',
                            np.fromstring(scans.data['X'], sep=','))

                    # Check if datatypes are constructed for Y
                    if type(scans.data['Y']) is not core.ScanData and  \
                        scans.data['Y'] != 'N/A':
                        # If not, parse the data from the XML and
                        scans.data['Y'] = core.ScanData('Y',
                            np.fromstring(scans.data['Y'], sep=','))

                    # Check if datatypes are constructed for SENSOR
                    if type(scans.data['SENSOR']) is not core.ScanData and  \
                        scans.data['SENSOR'] != 'N/A':
                        # If not, parse the data from the XML and
                        scans.data['SENSOR'] = core.ScanData('SENSOR',
                            np.fromstring(scans.data['SENSOR'], sep=','))

                self.currentImage = scans.formImage()
                self.displayImagePlot(self.currentImage, scans.data['X'],
                    scans.data['Y'], scans.data['SENSOR'])









    def helpKonfokalAction(self):
        pass

    def helpKonfokalAboutAction(self):
        pass

    # Action creation, addition, menu update functions
    def createAction(self, text, slot=None, shortcut=None, icon=None,
            tip=None, checkable=False, signal="triggered()"):
        action = QAction(text, self)
        if icon is not None:
            action.setIcon(QIcon(":/%s.png" % icon))
        if shortcut is not None:
            action.setShortcut(shortcut)
        if tip is not None:
            action.setToolTip(tip)
            action.setStatusTip(tip)
        if slot is not None:
            self.connect(action, SIGNAL(signal), slot)
        if checkable:
            action.setCheckable(True)
        return action

    def addActions(self, target, actions):
        for action in actions:
            if action is None:
                target.addSeparator()
            else:
                target.addAction(action)

    def updateMenu(self, args):
        args[0].clear()
        self.addActions(args[0], args[1])

    # Cleanliness check of program
    def okToContinue(self):
        if self.dirty:
            reply = QMessageBox.question(self,
                            "Konfokal - Unsaved Changes",
                            "Save unsaved changes?",
                            QMessageBox.Yes|QMessageBox.No|
                            QMessageBox.Cancel)
            if reply == QMessageBox.Cancel:
                return False
            elif reply == QMessageBox.Yes:
                self.fileSaveSession()
        return True

    # File Menu slots
    def fileNewSession(self):
        if not self.okToContinue():
            return
        dialog = newsessiondlg.NewSessionDlg(self)
        if dialog.exec_():
            user = dialog.userLineEdit.text()
            datetimestr = str(dialog.dateLabel.text()).split()
            datetimestr = [datetimestr[1], datetimestr[3]]
            notes = dialog.notesPlainTextEdit.toPlainText()
            self.sessionDict[len(self.sessionDict)] = core.Session(user,
                                    datetimestr, 'teknofil_cdaq', notes)
            self.updateNavigator()

    def fileLoadSession(self):
        if not self.okToContinue():
            return
        directory = os.path.dirname(self.filename) \
                    if self.filename is not None else "."
        formats = ["*.kff"]
        fname = str(QFileDialog.getOpenFileName(self,
                "Konfokal - Choose Konfokal File", directory,
                     "Konfokal File (%s)" % " ".join(formats)))

        if fname:
            self.sessionDict = utils.loadKFF(fname)
            # Set the working directory
            self.filename = fname
            self.directory = '/'.join(fname.split('/')[:-1])
            self.updateNavigator()

    def fileSaveSession(self):
        if self.filename:
            utils.saveKFF(str(self.filename), self.sessionDict)
        else:
            self.fileSaveAsSession()

    def fileSaveAsSession(self):
        fname = self.filename if self.filename is not None else "."
        formats = ["*.kff"]
        fname = str(QFileDialog.getSaveFileName(self,
                        "Konfokal - Save File", fname,
                        "Konfokal File (%s)" % " ".join(formats)))

        if fname:
            self.filename = fname
            utils.saveKFF(fname, self.sessionDict)


    def fileExport(self):
        pass

    def fileSettings(self):
        pass


    # Edit menu slots
    def editUndo(self):
        pass

    def editRedo(self):
        pass

    def editManipulate(self):
        pass

    # Scan menu slots
    def newScan(self):
        # set synthesize option
        synthesize = 1

        selecteditem = self.navigatorWidget.selectedItems()[0]
        if not selecteditem or selecteditem.parent():
            error = QErrorMessage()
            error.showMessage("Please select a session!")
            error.setWindowTitle("Konfokal")
            error.exec_()
            return
        wzd = newscanwzd.NewScanWizard(self)
        if wzd.exec_():
            # Get the title
            scantitle = wzd.titleLineEdit.text()
            notes = str(wzd.scanTextEdit.toPlainText())
            selecteditem.addChild(QTreeWidgetItem([scantitle]))

            currentSession = self.sessionDict[int(str(selecteditem.text(0)
                ).strip()[-1])-1]

            # Create Scan object
            scan = core.Scan(str(scantitle), currentSession.sessionparams['device'])


            # Parse waveform
            if wzd.triangularRadioButton.isChecked():
                wzdWave = ['T']
                fallrate = []
            else:
                wzdWave = ['S']
                fallrate = str(wzd.fallrateHorizontalSlider.value())
                fallrate = "%3s" %fallrate
                fallrate = fallrate.replace(' ', '0')
                fallrate = [fallrate]

            # Get scan limits
            wzdLimits = [str(wzd.xFromLineEdit.text()), str(wzd.xToLineEdit.text()),
                         str(wzd.xResolutionLineEdit.text()),
str(wzd.yFromLineEdit.text()),
                         str(wzd.yToLineEdit.text()),
str(wzd.yResolutionLineEdit.text())]

            for index in range(len(wzdLimits)):
                wzdLimits[index] = '%5s' %wzdLimits[index]
                wzdLimits[index] = wzdLimits[index].replace(' ', '0')

            # Get DAC speed
            print str(wzd.dacHorizontalSlider.value())
            wzdDac_speed = str(wzd.dacHorizontalSlider.value() * 1000)
            wzdDac_speed = "%7s" %wzdDac_speed
            wzdDac_speed = wzdDac_speed.replace(' ', '0')
            wzdDac_speed = [wzdDac_speed]

            self.lastscan = (wzdWave, wzdLimits, wzdDac_speed)

            received = dev.sendScanStr(wzdWave, wzdLimits, wzdDac_speed, fallrate)

            X,Y,PMT = core.ScanData('X'), core.ScanData('Y'), core.ScanData('SENSOR')
            for elem in range(len(received)):
                for lit in range(15):
                    parsed = int(received[elem][(lit*4):4+(lit*4)])

                    if lit%3 == 0:
                        PMT.dataset = np.append(PMT.dataset, parsed)
                    elif lit%3 == 1:
                        X.dataset = np.append(X.dataset, parsed)
                    elif lit%3 == 2:
                        Y.dataset = np.append(Y.dataset, parsed)

            if synthesize:
                X.dataset, Y.dataset = utils.generateTriangular(int(wzdLimits[0]),
                    int(wzdLimits[1]), int(wzdLimits[2]), int(wzdLimits[3]),
                    int(wzdLimits[4]), int(wzdLimits[5]))

            X.dataset -= min(X.dataset)
            Y.dataset -= min(Y.dataset)
            PMT.dataset -= min(PMT.dataset)

            Xresolution = (int(wzdLimits[1]) - int(wzdLimits[0]))/int(wzdLimits[2])
            Yresolution = (int(wzdLimits[4]) - int(wzdLimits[3]))/int(wzdLimits[5])



            X.dataset *= Xresolution / float(max(X.dataset))
            Y.dataset *= Yresolution / float(max(Y.dataset))
            PMT.dataset *= 255. / 4096


            scan.saveData(X,Y,PMT)

            # Append it to sessionDict
            currentSession.scanlist.append(scan)




    def quickScan(self):
        if self.lastscan is not None:
            dev.sendScanStr(self.lastscan[0], self.lastscan[1],
                self.lastscan[2])
        else:
            print "No scan to perform."


    def updateNavigator(self):
        self.navigatorWidget.clear()
        for (num, session) in sorted(self.sessionDict.items()):
            sessionItem = QTreeWidgetItem(["Session " + str(num+1)])
            for item in session.scanlist:
                scanItem = sessionItem.addChildren([
                                QTreeWidgetItem([item.data['TITLE']])])
                print item
            self.navigatorWidget.addTopLevelItem(sessionItem)

    # Tab widget functions
    def displayImagePlot(self, img, x, y, pmt):
        #img = mpimg.imread(imgPIL)
        ax = self.image.add_subplot(111)
        ax.imshow(img)

        axplt = self.plot.add_subplot(3,1,1)
        axplt.title("X, Y, PMT")
        axplt.plot(x)
        axplt = self.plot.add_subplot(3,1,2)
        axplt.plot(y)
        axplt = self.plot.add_subplot(3,1,3)
        axplt.plot(pmt)




def main():
    app = QApplication(sys.argv)
    app.setWindowIcon(QIcon(":/konfokal.png"))
    form = MainWindow()
    form.show()
    app.exec_()


main()
```

6.2. core.py

```
"""This module handles the image formation, storage and processing operations.
In order to create an image file, a Data() object should be provided.
"""
from __future__ import division

import PIL
import os
import matplotlib.pyplot as plt
import numpy as np
import datetime

class ScanData(object):
    """Contains the Data class for storing the scan data,
    manipulation functions which are used for handling: data
    generation, reformation, conversion, representation.
    """
    def __init__(self, datatype='N/A', dataset=np.array([])):
        """
        NAME
            Data()

        PARAMETERS
            Dataset: np.array
            Datatype: str
            Metadata: dict

        USAGEsa
            Creates a data class where datatype, dataset, datainverval and
            metadata(summary of data) is stored.
        """

        self.dataset = dataset
        self._datatype = datatype

    @property
    def datatype(self):
        return self._datatype

    @datatype.setter
    def datatype(self, value):
        self._datatype = value

    def synthesize(self, waveform, rep, wait, step, minimum=0, maximum=4096):
        """Generates data from [minimum, maximum) interval with step size of
        'step'. Used when creating synthetic position data.
        """

        length = wait*(step+1)
        self.dataset = np.zeros(rep*length)

        inc = int((maximum - minimum)/step)
        for i in range(rep):
            for j in range(step+1):
                self.dataset[j*wait+i*length:j*wait+wait+i*length] \
                = int(j*inc + minimum)

        # Construct the decreasing part of triangular wave
        if waveform == 'tri':
            for i in range(1,rep,2):
                for j in range(0,step+1):
                    self.dataset[j*wait+i*length:j*wait+wait+i*length] \
                    = int(self.dataset.max() - j*inc)

    def display(self):
        import matplotlib.pyplot as plt
        plt.plot(self.dataset)
        plt.show()

class Session(object):
    """Session class holds single/multiple scans, that are
    taken by a single user. Creates new or loads/saves
    sessions.
    """

    def __init__(self, user, date_time, device, notes=''):

        # Gather session parameters under a dict
        self.sessionparams = {'user':str(user),
        'date_time':str(date_time), 'device':str(device),
        'notes':str(notes)}

        # Collect associated scans with this session
        self.scanlist = []

class Scan(object):
    """Scan object holds the data that belong to a session.
    Contains ScanData objects for forming a ScanImage object.
    Generates a scan report which holds all the necessary
    information about speed, or any kind of problems.
    """

    def __init__(self, title, device, elapsed='N/A', xdata='N/A',
        ydata='N/A', sensordata='N/A', img='N/A', notes='N/A'):


        self.data = {
            'TITLE'   :  title,
            'DEVICE'  :  device,
            'ELAPSED' :  elapsed,
            'X'       :  xdata,
            'Y'       :  ydata,
            'SENSOR'  :  sensordata,
            'IMAGE'   :  img,
            'NOTES'    :  notes
        }


    def saveData(self, xScanData, yScanData, sensorScanData):
        self.data['X'] = xScanData
        self.data['Y'] = yScanData
        self.data['SENSOR'] = sensorScanData

    def formImage(self):
        self.data['IMAGE'] = ScanImage(self.data['X'], self.data['Y'],
                                       self.data['SENSOR'])
        return self.data['IMAGE']



class ScanImage(object):
    """A delegating wrapper, which extends the PIL.Image class'
    properties while generating a scan image.
    """

    def __init__(self, xpos, ypos, sensor):

        # X, Y and Sensor data
        self._x = xpos
        self._y = ypos
        self._sensor = sensor
        self._imagearray = np.zeros((max(ypos.dataset)+1, max(xpos.dataset)+1))

        for i in range(len(self._x.dataset)):
            self._imagearray[self._y.dataset[i]][self._x.dataset[i]] \
            = self._sensor.dataset[i]

        self._image = PIL.Image.fromarray(self._imagearray).convert("RGB")

    def __getattr__(self, key):
        if key == '_image':
            raise AttributeError()
        return getattr(self._image, key)

    def showTicks(self, saveticks=0):
        """Creates and pops up the image of the given Data() object in
        matplotlib. May be used for analysis.
        """
        pltobj = plt.imshow(self._imagearray)

        if saveticks:
            return pltobj
        plt.show()

    def saveTicks(self, filename):
        """Saves the image with ticks on it."""
        pltobj = self.showTicks(saveticks=1)
        plt.savefig(filename)

def testbench():

    xtxt = np.fromfile('x_data.txt')
    ytxt = np.fromfile('y_data.txt')
    pmttxt = np.fromfile('pmt_data.txt')
    xData = ScanData(dataset=xtxt)
    yData = ScanData(dataset=ytxt)
    pmtData = ScanData(dataset=pmttxt)
    session = Session('brn', '00:07', 'teknofil')
    session.createNewScan('nonotes', 'scan1')
    session.scanlist[0].saveData(xData, yData, pmtData)

    return session.scanlist[0].formImage()

#img = testbench()
```

6.3. dev.py

```
# Konfokal's dev file
import usb.util
import usb.core

def sendScanStr(waveform, ranges, dac_speed, damping=[]):
      """Sends scan string to TeknofilCDAQ.
            waveform: [WAVE=T,S or P] list,
            ranges: [XBEG, XEND, XRES, YBEG, YEND, YRES] list,
            dac_speed: SPEED
      """

      scanStr = waveform[0]

      # This is the string for performing scan
      for elem in ranges+dac_speed:
            if elem is not '':
                  scanStr += str(elem) + ' '

      if damping:
            scanStr += damping[0]
            print damping

      print scanStr

      # Find Teknofil CDAQ
      device = usb.core.find()

      # Found?
      if device is None:
          raise ValueError('Teknofil CDAQ not found')

      # Set active configuration. With no arguments, the first
      # configuration will be the active one.
      device.set_configuration()

      device.write(0x01, scanStr)
      received = []

      while True:
          try:
              reading = device.read(0x81, 64)
              received.append(reading.tostring())
          except Exception,e:
              break

      return received
```

6.4. utils.py

```
# Utilities for Konfokal. Functions for getting-things-done.

import xml.etree.ElementTree as xee
from core import *
import zipfile, glob, os
import numpy as np

def formXML(sessiondict, xmlfname="kff.xml"):

    # Create the 'root' container-tag for sessions and settings
    root = xee.Element('root')

    # Create 'sessions' tag for adding sessions
    sessions = xee.SubElement(root, 'sessions')

    # Iterate and add all sessions
    xmldict = dict()
    for (num,session) in enumerate(sessiondict.values()):
        sessionstr = 'session' + str(num)
        xmldict[sessionstr] = xee.SubElement(sessions, sessionstr,
                                        session.sessionparams)


        if session.scanlist:
           scandict = dict()
           session_scanlist = sessionstr + '_scans'
           xmldict[session_scanlist] = scandict
           for (snum,scan) in enumerate(session.scanlist):
               scanstr = 'scan' + str(snum)
               xmldict[session_scanlist][scanstr] = xee.SubElement(
                                xmldict[sessionstr], scanstr, scan.data)

    konfokalTree = xee.ElementTree(root)
    konfokalTree.write(xmlfname)

def parseXML(xmlfname="kff.xml"):
    "Parses given .kff XML file."
    xmlfile = xee.ElementTree(file=xmlfname)
    sessions = xmlfile.find('sessions')
    sessiondict = {}

    for (num,session) in enumerate(sessions.findall('*')):
        sessiondict[num] = Session(session.get('user'), \
          session.get('date_time'), session.get('device'), session.get('notes'))
        for scan in session.findall('*'):
            scanObj = Scan(scan.get('TITLE'), scan.get('DEVICE'),
             scan.get('ELAPSED'), scan.get('X'), scan.get('Y'),
             scan.get('SENSOR'), scan.get('IMAGE'), scan.get('NOTES'))
            sessiondict[num].scanlist.append(scanObj)

    return sessiondict
```

```
def saveKFF(fname, sessiondict, xmlfname="kff.xml"):
    "Saves given .xml file as fname.kff"
    formXML(sessiondict)
    kffFile = zipfile.ZipFile(fname, "w")
    kffFile.write(xmlfname, os.path.basename(xmlfname), zipfile.ZIP_DEFLATED)
    os.remove(xmlfname)
    kffFile.close()

def loadKFF(fname, xmlfname="kff.xml"):
    kffFile = zipfile.ZipFile(fname, "r")
    xmlstr = kffFile.read(xmlfname, os.path.basename(xmlfname))
    element = xee.fromstring(xmlstr)
    etree = xee.ElementTree(element)
    etree.write(xmlfname)
    sessiondict = parseXML(xmlfname)
    os.remove(xmlfname)
    return sessiondict

def generateTriangular(xbeg, xend, xstep, ybeg, yend, ystep):
    "Generate a triangular waveform, return as np array."

    # initial conditions
    xwave = np.array([])
    ywave = np.array([])

    xnow = xbeg
    ynow = ybeg
    counter = 0

    while ynow <= yend:
        if counter % 2:
            # going up
            while xnow < xend:
                xwave = np.append(xwave, xnow)
                ywave = np.append(ywave, ynow)
                xnow += xstep

        else:
            # going down
            while xbeg < xnow:
                xwave = np.append(xwave, xnow)
                ywave = np.append(ywave, ynow)
                xnow -= xstep
        ynow += ystep
        counter += 1
    return xwave, ywave

def testbench():

    ses1 = Session('brn', 'tarih1', 'cdaq', 'notlar1')
    ses1.createNewScan('scan1', 'degisikscan')
    ses1.createNewScan('lol2', 'sds')
    ses2 = Session('hodor', 'tarih2', 'teknofilCDAQ', 'not2')
    ses2.createNewScan('session2scan1', 'somenotes')

    sesdict = {0: ses1, 1:ses2}

    saveKFF('deneme', sesdict)

#testbench()
```

BIBLIOGRAPHY

[1] Davson, H. (1972). *The physiology of the eye* (3d ed.). New York: Academic Press.

[2] Rao, C. (2005). *Data mining and data visualization*. Amsterdam: Elsevier North Holland.

[3] Pedrotti, F., & Pedrotti, L. (1987). *Introduction to optics*. Englewood Cliffs, N.J.: Prentice-Hall.

[4] Minsky, M. (1988). Memoir on inventing the confocal scanning microscope. *Scanning,* 128-138.

[5] Webb, R. (1996). Confocal optical microscopy. *Rep. Prog. Phys. Reports on Progress in Physics,* 427-471.

[6] Xi, P., Rajwa, B., Jones, J., & Robinson, J. (n.d.). The design and construction of a cost-efficient confocal laser scanning microscope. *Am. J. Phys. American Journal of Physics,* 203-203.

[7] Cremer C., Cremer T. (1978). Considerations on a laser-scanning-microscope with high resolution and depth of field, *M1CROSCOPICA ACTA VOL. 81 NUMBER 1*, 31—44.

[8] (n.d.). from https://www.microscopyu.com/articles/confocal/images/pawley39stepsfigure1.jpg

[9] (n.d.). from https://en.wikipedia.org/wiki/Photomultiplier#/media/File:PhotoMultiplierTubeAndScintillator.jpg

[10] (n.d.). from
https://en.wikipedia.org/wiki/Photomultiplier#/media/File:PMT_Voltage_Divider.jpg

[11] (n.d.). from
http://www.camtech.com/index.php?option=com_content&view=article&id=92&Itemid=178

[12] (2012, January 24). Thor Labs Rev E PMM02 Amplified Photomultiplier User's Guide.

[13] Nisarga, B. (2015, September 28). PWM DAC Using MSP430 High-Resolution Timer. Retrieved from http://www.ti.com/lit/an/slaa497/slaa497.pdf

VITA

Baran Yalçın has received his B. Sc. degree in Electrical and Electronics Engineering from College of Engineering, Koç University, İstanbul, Turkey. He later joined the M. Sc. in Optoelectronics and Photonics Engineering program at Koç University, İstanbul, Turkey under supervisory of Prof. Dr. Alper Kiraz. For his M. Sc. thesis he worked on “Hardware/Software Design and Implementation of a Laser Scanning Confocal Microscope Controller Using Open Design Approach”.