WEARABLE HAND TREMOR ANALYSIS SYSTEM FOR CLINICAL APPLICATIONS

A Project report submitted in partial fulfillment of the requirements

for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

AKSHAYA KILLI (318126512062)

TAKASI KIRANMAYEE (318126512106)

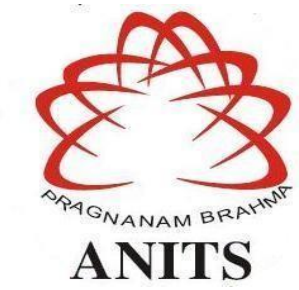
GOLLAPALLI PRISCILLA (318126512080)

UDIGALA MEGHANA (319126512L10)

Under guidance of

Dr.G.Prasanna

(Assistant Professor)



DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES

(*Permanently Affiliated to AU, Approved by AICTE and Accredited by NBA & NAAC with 'A' Grade*)

Sangivalasa, Bheemili Mandal, Visakhapatnam dist. (A.P) 2021-2022

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES

(Permanently Affiliated to AU, Approved by AICTE and Accredited by NBA & NAAC with 'A' Grade)

Sangivalasa, Bheemili Mandal, Visakhapatnam dist. (A.P)

PRAGNANAM BRAHMA

ANITS

CERTIFICATE

This is to certify that the project report entitled **"WEARABLE HAND TREMOR ANALYSIS SYSTEM FOR CLINICAL APPLICATION"** Submitted by in partial fulfillment of the requirements for the award of the degree of **Bachelor of Engineering Akshaya Killi (318126512062), TAKASI KIRANMAYEE (318126512106), GOLLAPALLI PRISCILLA (318126512080), UDIGALA MEGHANA(319126512L10)** in **Electronics & Communication Engineering** of Andhra University, Visakhapatnam is a record of bonafide work carried out under my guidance and supervision.

20/5/2022

Project Guide

Dr.G.Prasanna

Assistant professor

Department of E.C.E

ANITS

Assistant Professor
Department of E.C.E.
Anil Neerukonda
Institute of Technology & Sciences
Sangivalasa, Visakhapatnam-531 162

Head of the department

Dr. V. Rajya Lakshmi

Professor and HOD

Department of E.C.E

ANITS

Head of the Department
Department of E C E
Anil Neerukonda Institute of Technology & Sciences
Sangivalasa-531 162

ACKNOWLEDGEMENT

I would like to express my deep gratitude to my project guide **Dr.G.Prasanna,** Assistant Professor, Department of Electronics and Communication Engineering, ANITS, for their guidance with unsurpassed knowledge and immense encouragement. I am grateful to **Dr. V. Rajya Lakshmi**, Head of the Department, Electronics and Communication Engineering, for providing me with the required facilities for the completion of the project work.I am very much thankful to the **Principal and Management, ANITS, Sangivalasa,** for their encouragement and cooperation to carry out this work. I express my thanks to all **teaching faculty** of Department of ECE, whose suggestions during reviews helped me in accomplishment of my project. I would like to thank **all non-teaching staff** of the Department of ECE, ANITS for providing great assistance in accomplishment of our project. I would like to thank my parents, friends, and classmates for their encouragement throughout our project period. At last but not the least, I thank everyone for supporting me directly or indirectly in completing this project successfully.

PROJECT STUDENTS:

AKSHAYA KILLI(318126512062)

TAKASI KIRANMAYEE(318126512106)

G PRISCILLA (318126512080)

UDIGALA MEGHANA(319126512L10)

ABSTRACT

Tremors are involuntary rhythmic muscle contractions that can cause body parts to shake or tremble. Tremors in the hands are common in many diseases, they are used for differential diagnosis and monitoring. A prerequisite to diagnosis in a patient with tremor is its clinical classification as rest tremor, postural tremor, or intention tremor. In the existing systems one cannot detect and diagnose parallelly so we have developed a system interfacing a microcontroller (PSoC) and matlab using hardware components including ADXL- 335 sensor. Our wearable analysis system has sampling rate of 20Hz, power consumption of 5v@ 50mA, size/ volume of 5 cm cube and weight of 50mg which makes it feasible due to low cost, low weight and energy efficient.

CONTENTS

1. INTRODUCTION 1

1.1 Causes /conditions of hand tremor 1-8

1.2 Importance of this analysis 9

1.3 Existing systems for hand tremor analysis 9-10

1.4 Literature survey 10-11

1.5 Vibration 11-12

2.MOTIVATION 13-14

3. DESIGN OF PSoC BASED WEARABLE SYSTEM 15

3.1 Overview of the design 15-16

3.2 ADXL -335 accelerometer/motion sensor 17

3.3 Front end electronics 17-20

3.4 Schematic diagram of design 23

3.5 Sigma_Delta ADC 24-26

3.6 USB_UART 27-29

4.PSoC PROGRAM 30

4.1 PSoc program flow chart 31

4.2 PSoC KEIL-C Program 34-40

5.GUI BASED MATLAB – INTERFACE 41

5.1 matlab Program flow chart 41

5.2 MATLAB program 44-45

5.3 MATLAB program commands description 46-53

6.RESULT AND CONCLUSION 51-52

Result

6.1 Specifications of the model 53

6.2 Limitations 53

6.3 Future scope 53

6.4 Conclusion 54

REFERENCES 56-57

LIST OF FIGURES

Fig1 : Block diagram of design of hand glove

Fig2 : Functional diagram of ADXL-335 Accelerometer sensor

Fig3 : Pin configuration of ADXL-335 sensor

Fig4 : PSoC board

Fig5 : Architectural Overview of PSoC

Fig6 : Schematic diagram of PSoC chip connections

Fig7 : Block diagram of ADC del_sig

Fig8 : Wearable hand tremor system

Fig9 : Output of hand tremor after data acquisition analysis

LIST OF TABLES

Table1 : Pin description of ADXL sensor

Table2 : Bandwidth VS Capacitor of ADXL-335

Table3 : Programming interface command description

Table4 : USB _UART functional commands description

Table5 : USB DC specification

Table6 : USB AC specification

Table7 : Parameters of USBUART _ Start command

1 .INTRODUCTION

Tremors are involuntary rhythmic muscle contractions that can cause body parts to shake or tremble. Tremors in the hands are common. In some cases, they can be completely normal while in others they may signal an underlying medical condition. For some people, shaky hands may be a minor inconvenience. For others, the symptom may lead to difficulty using the hands for everyday tasks. Everyone has a slight tremor when moving or maintaining a particular posture. This is called physiologic tremor. Physiologic tremors are often so small that a person does not see or notice them. Hand tremors may be more noticeable when a person holds their hands out straight in front of the body or when they are stressed or anxious.

There are more than 20 types of tremors. However, most fall into two categories:

- **Resting tremors:** These occur when the muscles are relaxed, including when the hands are resting on the lap.
- **Action tremors:** These occur when the muscles are contracted due to voluntary movement. The majority of tremors are action tremor

1.1 CAUSES / CONDITIONS OF HAND SHAKE:

1.1.1 ESSENTIAL TREMORS:

Essential tremor is a nervous system (neurological) disorder that causes involuntary and rhythmic shaking. It can affect almost any part of the body, but the trembling occurs most often in hands — especially when you do simple tasks, such as drinking from a glass or tying shoelaces. Essential tremor is usually not a dangerous condition, but it typically worsens over time and can be severe in some people. About half of essential tremor cases appear to result from a genetic mutation. This form is referred to as familial tremor. It isn't clear what causes essential tremor in people without a known genetic mutation.

Risk factor

Genetic mutation. The inherited variety of essential tremor (familial tremor) is an autosomal dominant disorder. A defective gene from just one parent is needed to pass on the condition.If you have a parent with a genetic mutation for essential tremor, you have a 50 percent chance of developing the disorder yourself.

Age. Essential tremor is more common in people age 40 and older.

Complications

Essential tremor isn't life-threatening, but symptoms often worsen over time. If the tremors become severe, you might find it difficult to

- Hold a cup or glass without spilling
- Eat normally
- Put on makeup or shave
- Talk, if your voice box or tongue is affected
- Write legibly

1.1.2PARKINSON'SDISEASE

Parkinson disease (PD) is the most common neurodegenerative movement disorder . Its cardinal motor symptoms are tremor, rigidity ,bradykinesia/akinesia (Bradykinesia means slowness of movement and is one of the cardinal manifestations of Parkinson's disease. Weakness, tremor and rigidity may contribute to but do not fully explain bradykinesia) and postural instability, but the clinical picture includes other motor and non-motor symptoms (NMS). The diagnosis is principally clinical, although specific investigations can help the differential diagnosis from other forms of Parkinsonism.

SYMPTONS

Tremor: A tremor, or shaking, usually begins in a limb, often your hand or fingers. Patient's may rub thumb and forefinger back and forth, known as a pill-rolling tremor. Hand may tremble when it's at rest.

Slowed movement (bradykinesia): Over time, Parkinson's disease may slow patient's movement, making simple tasks difficult and time-consuming. Steps may become shorter when patient walk. It may be difficult to get out of a chair. Patient may drag his/her feet as they try to walk.

Rigid muscles: Muscle stiffness may occur in any part of the body. The stiff muscles can be painful and limit range of motion.

Impaired posture and balance: Posture may become stooped, or balance problems as a result of Parkinson's disease.

Loss of automatic movements: Decreased ability to perform unconscious movements, including blinking, smiling or swinging arms while walking.

Speech changes: Speak softly, quickly, slur or hesitate before talking. Speech may be more of a monotone rather than have the usual inflections.

RISK FACTORS

Risk factors for Parkinson's disease include:

Age: Young adults rarely experience Parkinson's disease. It ordinarily begins in middle or late life, and the risk increases with age. People usually develop the disease around age 60 or older.

Heredity: Having a close relative with Parkinson's disease increases the chances that you'll develop the disease. However, risks are still small unless relatives in patient's family with Parkinson's disease.

Exposure to toxins: Ongoing exposure to herbicides and pesticides may slightly increase risk of Parkinson's disease.

COMPLICATIONS

- Thinking difficulties
- Depression and emotional changes
- Swallowing problems
- Chewing and eating problems
- Sleep problems and sleep disorders
- Blood pressure changes
- Bladder problems
- Smell dysfunction
- Fatigue

CAUSES

Genes: Researchers have identified specific genetic mutations that can cause Parkinson's disease. But these are uncommon except in rare cases with many family members affected by Parkinson's disease.

Environmental triggers: Exposure to certain toxins or environmental factors may increase the risk of later Parkinson's disease, but the risk is relatively small.

The presence of Lewy bodies: Clumps of specific substances within brain cells are microscopic markers of Parkinson's disease. These are called Lewy bodies, and researchers believe these Lewy bodies hold an important clue to the cause of Parkinson's disease.

1.1.3 MULTIPLE SCLEROSIS (MS)

Multiple sclerosis (MS) is a potentially disabling disease of the brain and spinal cord (central nervous system). Signs and symptoms of MS vary widely and depend on the amount of nerve damage and which nerves are affected. Some people with severe MS may lose the ability to walk independently or at all, while others may experience long periods of remission without any new symptoms

Symptoms

Multiple sclerosis signs and symptoms may differ greatly from person to person and over the course of the disease depending on the location of affected nerve fibers. Symptoms often affect movement, such as:

- Numbness or weakness in one or more limbs that typically occurs on one side of your body at a time, or your legs and trunk
- Electric-shock sensations that occur with certain neck movements, especially bending the neck forward (Lhermitte sign)

- Tremor, lack of coordination or unsteady gait

Complications

People with multiple sclerosis may also develop:

- Muscle stiffness or spasms
- Paralysis, typically in the legs
- Problems with bladder, bowel or sexual function
- Mental changes, such as forgetfulness or mood swings
- Depression
- Epilepsy

1.1.4 ALCHOL WITHDRAWAL

Alcohol shakes, also called tremors, often occur when a person who has regularly consumed heavy amounts of alcohol suddenly stops drinking. Tremors are uncontrollable shaking, usually in the hands, and are often a side effect of alcohol withdrawal.Tremors may begin 5-10 hours after the last drink. That's why some who regularly drink large amounts of alcohol wake up with the shakes and need a drink to "feel steady." Tremors typically peak 24 to 78 hours after the last drink, but may last for several weeks or longer.

1.1.5 ENHANCED PHYSIOLOGIC TREMOR

Enhanced physiologic tremor (EPT) is a more noticeable form of physiologic tremor. It usually affects the hands and fingers on both sides of the body.

The following may cause EPT in some people:

- stress
- anxiety
- fatigue
- lack of sleep
- excessive caffeine intake
- vigorous exercise
- overactive thyroid

Enhanced physiological tremor does not require medical treatment, except when a person needs to rely on fine muscle coordination for their work or other activities.

1.1.6 STROKE:

Stroke occurs due to a decrease or blockage in the brain's blood supply. A person experiencing a stroke needs immediate emergency treatment.After a stroke , a person can show a variety of tremors depending on the area affected. Damage to the basal ganglia causes a person to have resting tremors, while damage to the cerebellum causes intention tremors.

1.1.7 MEDICATIONS

Certain drugs can also cause hand tremors. Examples include:

- some asthma medications
- Drugs for psychiatric conditions, such as certain antidepressants and mood stabilizers
- seizure medication, such as valproate (Depakene) and valproic acid (Depakote)
- anti-arrhythmic drugs, such as procainamide
- cancer medications ,such as thalidomide
- Medications that suppress the immune system, such as cyclosporine
- corticosteroids
- certain antiviral drugs
- specific antibiotics
- amphetamine
- caffeine

1.1.8 DYSTONIA:

Dystonia is a movement disorder in which involuntary muscle contractions cause repetitive, involuntary movements and postures. The condition is due to improper functioning of the basal ganglia in the brain. A study stated that dystonia and tremor are closely linked. Tremors occurring in people with dystonia are either jerky and irregular, regular and wave-like, or mixed. Mixed types commonly affect the hands.

1.1.9 CEREBELLAR DISORDER

Classic cerebellar tremor is often termed as intention tremor. The tremor is typically of low frequency below 5 Hz. It is characteristically kinetic in nature and has an added volitional component and particularly affects the head and the upper half of the body. Postural tremor may be present, but rest tremor is usually absent. When kinetic tremor occurs or worsens as the target is reached, it is referred to as terminal tremor. In rare occasions, cerebellar tremor also has a rest component in which case it would

be described as Holmes' tremor. In cerebellar tremor, the oscillations are of variable amplitude and are perpendicular to the direction of movement. It is usually best elicited during the finger-nose-finger or heel-shin-heel tests. Furthermore, cerebellar tremor is often associated with dysmetria, dyssynergia, and hypotonia. Titubation is another tremor that is probably a result of abnormality of the cerebellum or its afferent/efferent pathways and is a slow frequency oscillation depending on postural innervation. Its rhythmicity is, at times, the only sign distinguishing it from ataxia of the trunk. Multiple sclerosis is a common cause of cerebellar tremor. Other causes include Friedreich's ataxia, spinocerebellar degeneration, and cerebellar infarction. Although no clear correlations between cerebellar lesions and tremor have been established, lesions in the superior cerebellar peduncle and dentate nucleus are the most common reported sites resulting in intention tremor.

Unfortunately, there is no established pharmacological treatment for cerebellar tremor. The results of medical treatment are often less than satisfactory. However, as multiple neurotransmitters as well as feedback pathways, including brain stem, thalamus and cortical neurons, seem to be involved, several drugs have been tried, but with variable success, including odansetron (5-HT_3 antagonist), isoniazid, physostigmine, carbamazepine, and clonazepam. In refractory cases, chronic DBS of the VIM (or less commonly nucleus ventralisoralis posterior and zona incerta) may provide an alternative. Although few studies used highly standardized quantitative outcome measures, and follow up periods were generally one year or less, the data suggested that chronic DBS of the VIM produced improved tremor control in multiple sclerosis. However, complete cessation of tremor is generally not achieved. There were some reported cases in which tremor control decreased over time, and frequent reprogramming became necessary.

1.2 IMPORTANCE OF THIS ANALYSIS

Essential Tremor (ET) and Cerebellar Disorders (CD) are some of the examples of tremulous diseases. ET does not require emergent treatment while CD is often caused by emergent condition such as stroke . However, it is sometimes difficult to diagnose these diseases accurately by only physical examination in outpatient setting. In addition, there is no indicator for quantitatively evaluating the tremor. Therefore, many doctors have risks to misdiagnose these diseases.

Wearable hand tremor analysis system is quantitative, accurate, less time consumption, low cost and light weight system for the tremor analysis which gives the clinical supports for the above mentioned causes or conditions of the hand shaking or the hand tremor and helps to give the right treatment without misjudging the condition or cause of the hand tremors to the patient. The quantitative analysis indicates the seriousness of the disease or cause of the hand tremors

1. 3 EXISTING SYSTEMS FOR HAND TREMOR ANALYSIS

1.3.1 ELECTROMYOGRAPHY

Electromyography (EMG) is a diagnostic procedure to assess the health of muscles and the nerve cells that control them (motor neurons). EMG results can reveal nerve dysfunction, muscle dysfunction or problems with nerve-to-muscle signal transmission which causes tremors.Motor neurons transmit electrical signals that cause muscles to contract. An EMG uses tiny devices called electrodes to translate these signals into graphs, sounds or numerical values that are then interpreted by a specialist.During a needle EMG, a needle electrode inserted directly into a muscle records the electrical activity in that

muscle .A nerve conduction study, another part of an EMG, uses electrode stickers applied to the skin(surface electrodes) to measure the speed and strength of signals traveling between two or more points.

EMG results are often necessary to help diagnose or rule out a number of conditions such as:

- Muscle disorders, such as muscular dystrophy or polymyositis
- Diseases affecting the connection between the nerve and the muscle, such as myasthenia gravis
- Disorders of nerves outside the spinal cord (peripheral nerves), such as carpal tunnel syndrome or peripheral neuropathies
- Disorders that affect the motor neurons in the brain or spinal cord, such as amyotrophic lateral sclerosis or polio
- Disorders that affect the nerve root, such as a herniated disk in the spine

RISKS

EMG is a low-risk procedure, and complications are rare. There's a small risk of bleeding, infection and nerve injury where a needle electrode is inserted.When muscles along the chest wall are examined with a needle electrode, there's a very small risk that it could cause air to leak into the area between the lungs and chest wall, causing a lung to collapse (pneumothorax).

1.3.2 DOPAMINE TRANSPOTER SCAN (Da T Scan)

DaT Scan (DaT scan or Dopamine Transporter Scan) commonly refers to a diagnostic method to investigate if there is a loss of dopaminergic neurons in striatum. The term may also refer to a brand name of Ioflupane (123I) which is used for the study. The scan principle is based on use of the radiopharmaceutical Ioflupane (123I) which binds to dopamine transporters (DaT). The signal from them is then detected by the use of single-photon emission computed tomography (SPECT) which uses special gamma-cameras to create a pictographic representation of the distribution of dopamine transporters in the brain.DaTSCAN is indicated in cases of tremor in patients, when we are not sure about its origin. Although this method can distinguish essential tremor from Parkinson's syndrome, it is unable to show us if the problem is Parkinson’s disease, Multiple system atrophy .

At the beginning a patient should take two iodine tablets and wait for one hour. These pills are highly important, because they prevent the accumulation of radioactive substances in thyroid gland. After one hour, the patient gets an injection to shoulder, which contain the radiopharmaceutical and then he has to wait for 4 hours. The concentration of the substance increases and then it is scanned by gamma-camera, which is located around his head. Whole examination lasts about 30–45 minutes and it is non-invasive.If a patient use some of the medication listed below, it is necessary to stop the using few days or weeks before the DaTSCAN, but just after a consultation with his doctor.The examination takes just a few hours, so patients needn´t to stay in a hospital overnight, but they have to drink much more than they are used to and go to the toilet more often. It is important for a fast elimination of the radioactive substances from the body.

1.4 LITERATURE SURVEY

1.4.1 Using Wearable Sensor Systems for Objective Assessment of Parkinson's Disease

Title:-Using Wearable Sensor Systems for Objective Assessment of Parkinson's Disease

Abstract of the paper:

This paper presents a novel wearable sensor system based on the integration of miniaturised IMUs for fine hand movement analysis. The system, named SensHand V1, is composed of full 9-axis inertial sensors, placed on the fingers and wrist, which are managed by a cortex-M3 microcontroller. The acquired data are sent to a data logger through the use of Bluetooth communication. In this paper, the system is used for the objective diagnosis of Parkinson's disease, which is commonly assessed by neurologists through visual examination of motor tasks and semi-quantitative rating scales. Here, these motor tasks are also assessed using the SensHand V1, and then compared with the subjective metrics. Results demonstrate that the system is adequate to support neurologists in diagnostic procedures and allows for an objective evaluation of the disease.

The SensHand V1 wearable device was developed using inertial sensors integrated into four INEMO-M1 boards based on MEMS sensors and equipped with dedicated STM32F103RE microcontrollers .

The inertial data acquired with the SensHand V1 were stored on the PC and processed offline using Matlab®. All of the measured parameters were obtained from the acceleration and angular rate data supplied by the accelerometers and gyroscopes. A fourth-order low-pass digital Butterworth filter was applied with a 5 Hz cut-off frequency (fc) in order to eliminate high-frequency noise and tremor frequency bands. In the analysis conducted in the spatio-temporal domain, the appropriate algorithms of segmentation were implemented in order to identify the characteristic times of the typical phases for each exercise. Angular rates were integrated using the trapezoidal rule, with sub-intervals of integration equal to the inverse of the sensor-sampling rate (Δt=100 ms), in order to calculate the movement amplitude. For finger tapping exercises, it was hypothesized that movement occurred only at the metacarpo-

patient, overall mean and standard deviation of peak frequencies were obtained for this study group. The results show a mean of 5.05 ± 2.03 Hz for peak frequencies for the hand tremor patient sample in Sri Lanka. By the gender 4.62 ± 1.78 Hz and 5.69 Hz ± 2.38 Hz are the mean of peak frequencies for male and female sample respectively. This research will also be useful in disease diagnosing in clinical studies and developing hand tremor assessment tools.

The authors propose an IMU based soft glove with four IMUs attached to capture the hand tremor: two on the index finger and two on the middle finger. Middle, ring and small fingers are considered as one unit, as these three fingers tend to move together. Hence, the middle finger is used as a representative for those 3 fingers. Sensors are to be placed on top of the proximal (between the Metacarpophalangeal(MCP) joint and Proximal interphalangeal (PIP) joint) and middle phalanges (between the Proximal interphalangeal (PIP) joint and Distal interphalangeal (DIP) joint) of each finger. A 6-axis IMU (MPU-6050) which consists of an accelerometer and a gyroscope is selected and used in measuring the tremor in this proposed soft glove. The IMUs are attached to a soft cotton glove for easy donning and doffing and are attached using silicon sealant.

1.5 VIBRATION

1.5.1 TREMOR

A tremor is an unintentional and uncontrollable rhythmic movement of one part or one limb of your body. A tremor can occur in any part of the body and at any time. It's usually the result of a problem in the part of your brain that controls muscular movement.

1.5.2 GAIT

Gait is a person's pattern of walking. Walking involves balance and coordination of muscles so that the body is propelled forward in a rhythm, called the stride. There are numerous possibilities that may cause an abnormal gait. Some common causes are:

- A degenerative disease (such as arthritis)
- An inner ear disorder
- Stroke
- Foot conditions
- A neurologic condition
- Something as simple as ill-fitting shoes

1.5.2.1 TYPES OF GIAT DISORDERS

The following gait disorders are so distinctive as to earn names:

Propulsive gait. This type of gait is seen in patients with parkinsonism. It is characterized by a stooping, rigid posture, and the head and neck are bent forward. Steps tend to become faster and shorter.

Scissors gait. This type of gait gets its name because the knees and thighs hit or cross in a scissors-like pattern when walking. The legs, hips, and pelvis become flexed, making the person appear as though he or she is crouching. The steps are slow and small. This type of gait occurs often in patients with spastic cerebral palsy.

Steppage gait. A "high stepping" type of gait in which the leg is lifted high, the foot drops (appearing floppy), and the toes points downward, scraping

Spastic gait. Common to patients with cerebral palsy or multiple sclerosis, spastic gait is a way of walking in which one leg is stiff and drags in a semicircular motion on the side most affected by long-term muscle contraction.

the ground, when walking. Peroneal muscle atrophy or peroneal nerve injury, as with a spinal problem (such as spinal stenosis or herniated disc), can cause this type of gait.

Waddling gait. Movement of the trunk is exaggerated to produce a waddling, duck-like walk. Progressive muscular dystrophy or hip dislocation present from birth can produce a waddling gait.

2 .MOTIVATION

Neuromuscular diseases are currently increasing in prevalence over many age groups worldwide. These diseases occur due to different complications in the peripheral and central nervous system. Several symptoms can be seen in these diseases, of which, tremors being the most common. Tremors are involuntary contractions of muscle groups in the body due to neuromuscular diseases . Common body parts where tremors are observed are the limbs and head. Tremors in the hands can cause difficulties for patients especially when performing activities of daily living (ADL) i.e., feeding themselves or holding a glass of water or writing.

Essential Tremor (ET) and Cerebellar Disorders (CD) are some of the examples of tremulous diseases. ET does not require emergent treatment while CD is often caused by emergent condition such as stroke . However, it is sometimes difficult to diagnose these diseases accurately by only physical examination in outpatient setting. In addition, there is no indicator for quantitatively evaluating the tremor. Therefore, many doctors have risks to misdiagnose these diseases.

Tremor is an impairing symptom associated with several neurological diseases. Some of such diseases are neurodegenerative and tremor characterization may be of help in differential diagnosis. To date, electromyography (EMG) is the gold standard for the analysis and diagnosis of tremors. In the last decade, however, several studies have been conducted for the validation of different techniques and new, non-invasive, portable, or even wearable devices have been recently proposed as complementary tools to EMG for a better characterization of tremors. Such devices have proven to be useful for monitoring the efficacy of therapies or even aiding in differential diagnosis. The aim of this review is to present systematically such new solutions, trying to highlight their potentialities and limitations, with a hint to future developments.

Surface EMG is the gold standard technique for the diagnosis, characterization, and monitoring of tremor . Unfortunately, it suffers from uncertainty and errors due to bad positioning of electrodes, changes in skin conductance, and cross-talking from other muscles. To avoid such inconveniences, EMG is the most reliable technique for a precise characterization of tremor features, but it is invasive and costly. Generally, EMG is unsuitable for continuous monitoring or frequent assessment of tremor characteristics.

In the last decade, the large diffusion of mobile devices has fostered the development of several portable and wearable solutions for health monitoring or even for disease diagnosis. Most of such devices are based on inertial sensors (accelerometers and gyroscopes), while others use a combination of inertial and electrophysiological information. Many of them can be interfaced with smart phones or tablets through wireless communication protocols (Bluetooth, Wi-Fi, etc.). Smart phones, smart watches, and tablets have sufficient computing resources for performing complex calculations, such as digital signal processing and artificial intelligence (AI).

Mobile devices, together with the advent of the Internet of Things (IoT), have dramatically changed people&;s lifestyles and have found newer and newer areas of application, allowing for continuous monitoring of disease symptoms and vital signs. However, signal processing techniques and sensing technologies need to be properly selected in order to provide data in agreement with the clinical-functional assessment of tremor .

Wearable sensors have undergone important developments in the last decade in an increasing number of areas of application. Healthcare is one of the most promising sectors, where new technologies are being used for sensing, acquiring, analyzing, and sharing data. Several wearable solutions have been implemented, either using commercially available devices or developing custom systems, for aiding clinical evaluation and diagnosis. In this short review, we have focused on devices and solutions for the assessment, continuous monitoring, and diagnosis of tremor in neurological diseases. As a first consideration, up to date, most wearable applications are mainly focused on tremor assessment and quantification of tremor severity. A minor number of solutions are dedicated to home monitoring of tremor symptoms in order to fully characterize their occurrence and severity during daily life tasks and to optimize therapies. The use of wearable technologies for differential diagnosis between tremulous disorders is very promising. In the next future, more efforts will be devoted to this field. Another consideration regards sensing technologies. Inertial sensing based on Micro Electro-Mechanical Systems (MEMS) is still the most used technology for wearable devices measuring tremor. This is mainly due to their physical properties: tremor is a rhythmic movement, and these transducers sense motion. Moreover, they are nearly ubiquitous, as they are embedded in all mobile communication and entertainment devices, in smart watches and smart bands used for sports and fitness. Last, they can be easily embedded in any wearable solution thanks to their small dimensions and low power requirements. However, in diagnostic applications, the accuracy that can be achieved using MEMS sensors is still lower than that of solutions that include EMG and tremor pattern analysis.

So for easy ,accurate , quantitative analysis ,light weight, less power consumption and low cost go for the glove based wearable hand tremor analysis system for clinical application which is affordable and low man power is required when compared to all longitudinal diagnosis in most of them are costly also.

3.DESIGN OF PSOC BASED WEARABLE SYSTEM

Design of wearable hand tremor analysis system for clinical application for quantitative and accurate analysis of hand tremor based on PSoC. In this system the motion sensor called accelerometer ADXL-335 is used for the motion detection and the output of the 3-axis accelerometer is given to the front-end electronics for analog processing through analog channel. Here the front-end electronic used is PSoC chip for ADC (analog to d conversion) and the digital data is given to GUI based Matlab interface through digital communication link USB-UART. The data acquisition is completed after a getting a plot of the hand tremor in Matlab.

3.1 OVERVIEW OF THE DESIGN :

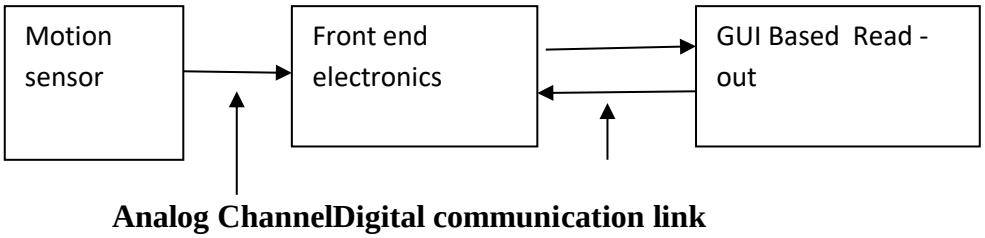


Fig 1 : Block diagram of design of hand glove

3.1.1 MOTION / ACCELEROMETER SENSOR :

For designing of wearable hand tremor analysis for clinical application requiring motion sensor . Selecting a specified motion sensor with accurate function , low cost and light weight .The main characters to be consider while selecting an accelerometer/ motion sensor are number of axes ,output type (analog or digital), output range, sensitivity, dynamic range, bandwidth, amplitude stability, and mass.

Some are 3-axis (meaning that it can measure acceleration in all 3 directions), 6-axis (meaning a 3-axis accelerometer and a 3-axis gyroscope) or 9 axis (meaning a 3-axis accelerometer, 3-axis gyroscope and 3-axis magnetometer); the resolution is typically specified as bits; the range of forces can vary from ±1g

up to ±250g (the smaller the range, the more sensitive the readings); all of these features are included within a single IC package.

The commercial motion /accelerometer sensor available are

i. **MMA8451QR1 by NXP:**A 3-axis accelerometer, 14-bit, I2C interface, XYZ, ±2g/±4g/±8g, 16-pin QFN, Reel packaging.

ii. **BMI055 by Bosch:** A 6-axis inertial sensor, 12-bit acceleration sensor, ±2000°/s gyroscope, ultra-small footprint: 3 x 4.5 mm², LGA-16 package.

iii. **ADXL345BCCZ-RL by Analog Devices**: A 3-axis accelerometer, 13-bit, ±2/±4/±8/±16g, SPI and I2C digital interfaces, LGA-14 package.

iv. **F1101 by Fairchild Semiconductor**: A 3-axis gyroscope and 3-axis accelerometer Inertial Measurement with Motion Processing, LGA-16 package.

v. **LSM303DLHC by STMicroelectronics:** A 3D digital linear acceleration sensor, I2C interface, ±2/±4/±8/±16g, available LGA-14 package.

vi. **MPU-6000 by InvenSense**:3-axis gyroscope, 3-axis accelerometer in a 4×4 package, I2C or SPI interface, ±2/±4/±8/±16g, QFN-24 package.

vii. **BMI160 by Bosch**:6-axis inertial MEMS sensor, 16-bit acceleration sensor and 16-bit gyroscope, ±2/±4/±8/±16g, LGA-14 package.

viii. **BNO055 by Bosch**: 9-axis motion sensor, 14-bit accelerometer, ±2000°/s 16 bit gyroscope, 5.2 x 3.8 x 1.1 mm³, LGA-28 package.

ix. **MPU-6050 by InvenSense**: A 3-axis gyroscope and a 3-axis accelerometer, ±2/±4/±8/±16g, 4 x 4 x 0.9 mm³ QFN-24 package.

x. **MPU-9250 by InvenSense:** A 9-axis Motion Tracking device, Gyroscope/Accelerometer/Magnetometer Sensor; 3 x 3 x 1mm³ QFN-24 package.

xi. **ADXL – 335 :** A 3 – axis accelerometer and 3 –axis gyroscope, small ,low power,±3 *g*

All of above listed motion /accelerometer sensor ADXL -335 selected because most of them are costly or not having the simple interface. We have chosen a simple low –cost sensor ADXL – 335 accelerometer sensor .

3.2 ADXL -335 ACCELEROMETER /MOTION SENSOR:

The ADXL335 is a small, thin, low power, complete 3-axis accelerometer with signal conditioned voltage outputs. The product measures acceleration with a minimum full-scale range of ±3 g. It can measure the static acceleration of gravity in tiltsensing applications, as well as dynamic acceleration resulting from motion, shock, or vibration.

The user selects the bandwidth of the accelerometer using the C_X, C_Y, and C_Z capacitors at the X_{OUT}, Y_{OUT}, and Z_{OUT} pins. Bandwidths can be selected to suit the application, with a range of 0.5 Hz to 1600 Hz for X and Y axes, and a range of 0.5 Hz to 550 Hz for the Z axis.

The ADXL335 is available in a small, low profile, 4 mm × 4 mm × 1.45 mm, 16-lead, plastic lead frame chip scale package (LFCSP_LQ).

Feature and benefits of ADXL – 335:

- Small, low-profile package
- 4 mm × 4 mm × 1.45 mm LFCSP
- Low power - 350 μA (typical)
- Single-supply operation 1.8 V to 3.6 V
- 3-axis sensing
- 10,000 *g* shock survival
- Excellent temperature stability
- BW adjustment with a single capacitor per axis
- RoHS/WEEE lead-free compliant

3.2.1 FUNCTIONAL DIAGRAM OF ADXL-335 ACCELEROMETER SENSOR

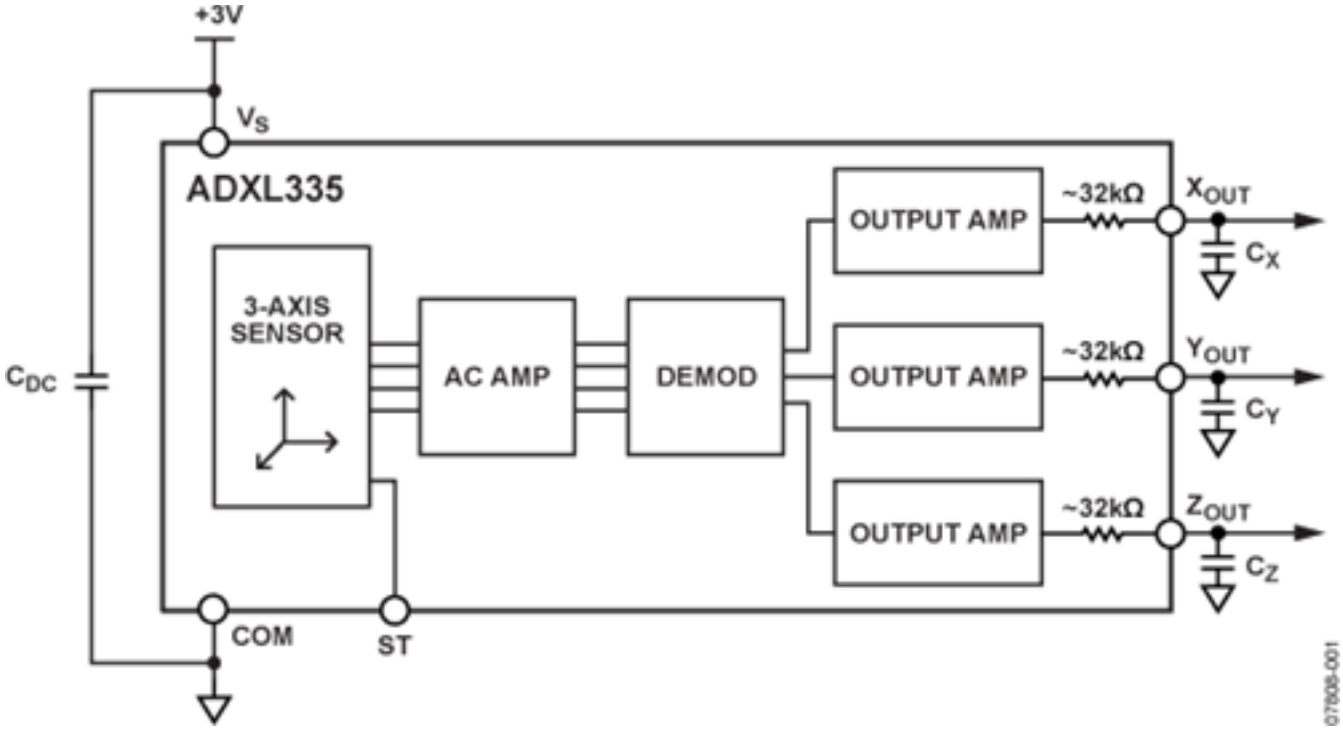


Fig 2 : Functional Diagram of ADLX -335 accelerometer sensor

3.2.2 PIN CONFIGURATION AMND DESCRIPTION

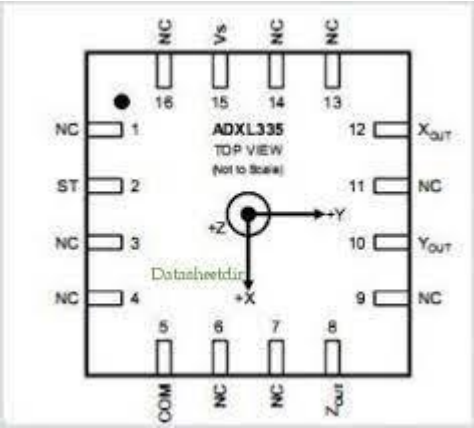


fig 3 : pin configuration of ADXL -335 sensor

Pin no.	**Mnemonic**	***Description***
1	NC	*NO CONNECT*
2	ST	*SELF-TEST*
3	COM	*COMMON*
4	NC	*NO CONNECT*
5	COM	*COMMON*
6	COM	*COMMON*
7	COM	*COMMON*
8	ZOUT	*Z CHANNEL OUTPUT*
9	NC	*NO CONNECT*
10	YOUT	*Y CHANNEL OUTPUT*
11	NC	*NO CONN*
12	XOUT	*X CHANNEL OUTPUT*
13	NC	*NO CONNECT*
14	VS	*SUPPLY VOLTAGE(1.8V-3.6V)*
15	VS	*SUPPLY VOLTAGE(1.8V-3.6V)*
16	NC	*NO CONNECT*
EP	EXPOSED PAD	*NOT INTERNALLY CONNECTED. SOLDER FOR MECHANICAL INTEGRITY*

Table 1:pin description of ADXL -335 Sensor

3.2.3 WORKING PRINCIPLE OF ADXL-335 SENSOR

The ADXL335 is a complete 3-axis acceleration measurement system. The ADXL335 has a measurement range of ±3 g minimum. It contains a polysilicon surface-micromachined sensor and signal conditioning circuitry to implement an open-loop acceleration measurement architecture. The output signals are analog voltages that are proportional to acceleration. The accelerometer can measure the static acceleration of gravity in tilt-sensing applications as well as dynamic acceleration resulting from motion, shock, or vibration. The sensor is a polysilicon surface-micromachined structure built on top of a silicon wafer. Polysilicon springs suspend the structure over the surface of the wafer and provide a resistance against acceleration forces. Deflection of the structure is measured using a differential capacitor that consists of independent fixed plates and plates attached to the moving mass. The fixed plates are driven by 180° out-of-phase square waves. Acceleration deflects the moving mass and unbalances the differential capacitor resulting in a sensor output whose amplitude is proportional to acceleration. Phase-sensitive demodulation techniques are then used to determine the magnitude and direction of the acceleration. The demodulator output is amplified and brought off-chip through a 32 kΩ resistor. The user then sets the signal bandwidth of the device by adding a capacitor. This filtering improves measurement resolution and helps prevent aliasing.

3.2.4 BANDWIDTH

The ADXL335 has provisions for band limiting the XOUT, YOUT, and ZOUT pins. Capacitors must be added at these pins to implement low-pass filtering for antialiasing and noise reduction. The equation for the 3 dB bandwidth is

$$F_{-3\ dB} = 1/(2\pi(32\ k\Omega) \times C(X, Y, Z))$$

or more simply

$$F_{-3\ dB} = 5\ \mu F/C(X, Y, Z)$$

The tolerance of the internal resistor (RFILT) typically varies as much as ±15% of its nominal value (32 kΩ), and the bandwidth varies accordingly. A minimum capacitance of 0.0047 μF for CX, CY, and CZ is recommended in all cases.

BANDWIDTH(HZ)	CAPACITOR (μF)
1	4.7
10	0.47
50	0.1
100	0.05
200	0.027
500	0.01

Table 2 :Bandwidth Vs Capacitor of ADXL -335

3.3 FRONT-END ELECTRONICS

PSoC devices integrate configurable analog and digital circuits. These are controlled by an on-chip microcontroller, thus, providing both enhanced design revision capability and component count savings. These devices occupy minimum board space, consume less power, provide high efficiency and reduce system cost. They can provide up to 100 peripherals. Peripherals allow interaction with the physical world. They also utilize fewer components to perform a task.

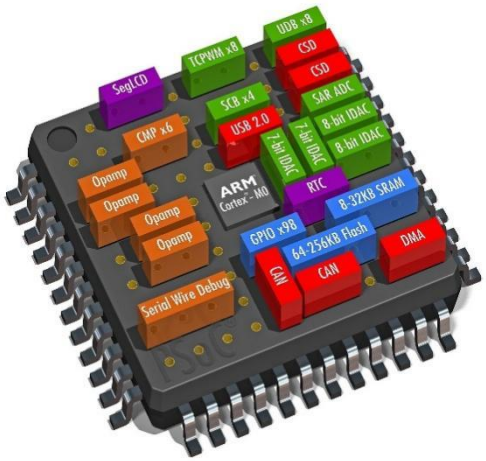


Fig 4 :PSoC board

3.3.1 ARCHITECTUARAL OVERVIEW

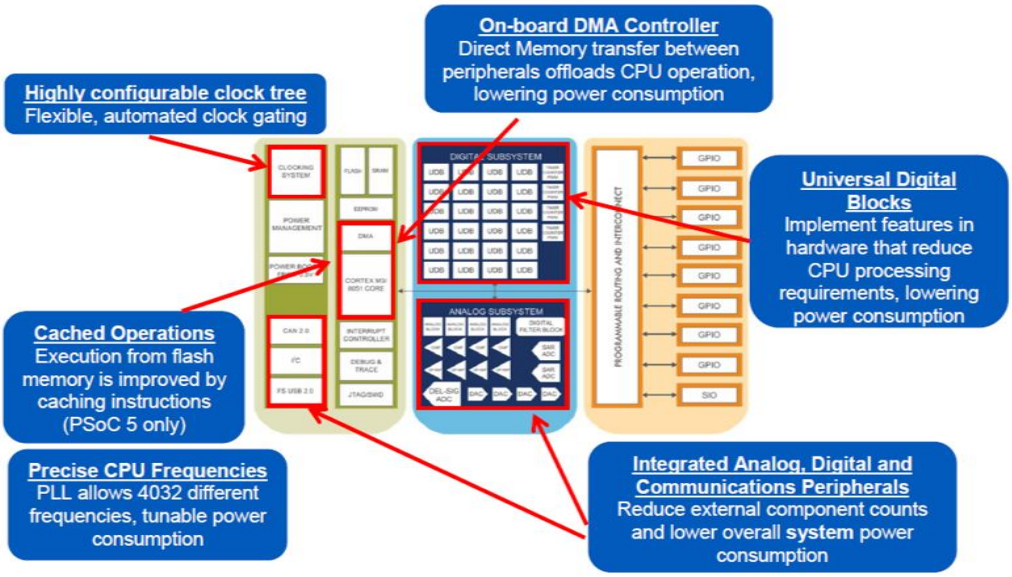


Fig 5 :Architectural overview of PSoC

A PSoC integrated circuit is composed of a core, configurable analog and digital blocks, and programmable routing and interconnect. The configurable blocks in a PSoC are the biggest difference from other microcontrollers.

PSoC has three separate memory spaces: paged SRAM for data, Flash memory for instructions and fixed data, and I/O registers for controlling and accessing the configurable logic blocks and functions. The device is created using SONOS technology.

PSoC resembles an ASIC: blocks can be assigned a wide range of functions and interconnected on-chip. Unlike an ASIC, there is no special manufacturing process required to create the custom configuration — only startup code that is created by Cypress' *PSoC Designer* (for PSoC 1) or *PSoC Creator* (for PSoC 3 / 4 / 5) IDE.

PSoC resembles an FPGA in that at power up it must be configured, but this configuration occurs by loading instructions from the built-in Flash memory.

PSoC most closely resembles a microcontroller combined with a PLD and programmable analog. Code is executed to interact with the user-specified peripheral functions (called "Components"), using automatically generated APIs and interrupt routines. *PSoC Designer* or *PSoC Creator* generate the startup configuration code. Both integrate APIs that initialize the user selected components upon the users needs in a Visual-Studio-like GUI.

3.3.2CONFIGURABLE ANALOG AND DIGITAL BLOCKS

Using configurable analog and digital blocks, designers can create and change mixed-signal embedded applications. The digital blocks are state machines that are configured using the blocks registers. There are two types of digital blocks, Digital Building Blocks (DBBxx) and Digital Communication Blocks (DCBxx). Only the communication blocks can contain serial I/O user modules, such as SPI, UART, etc.

Each digital block is considered an 8-bit resource that designers can configure using pre-built digital functions or user modules (UM), or, by combining blocks, turn them into 16-, 24-, or 32-bit resources. Concatenating UMs together is how 16-bit PWMs and timers are created.

There are two types of analog blocks. The continuous time (CT) blocks are composed of an op-amp circuit and designated as ACBxx where xx is 00–03. The other type is the switch cap (SC) blocks, which

allow complex analog signal flows and are designated by ASCxy where x is the row and y is the column of the analog block. Designers can modify and personalize each module to any design.

3.3.3 PROGRAMMABLE ROUTING AND INTERCONNECT

PSoC mixed-signal arrays' flexible routing allows designers to route signals to and from I/O pins more freely than with many competing microcontrollers. Global buses allow for signal multiplexing and for performing logic operations. Cypress suggests that this allows designers to configure a design and make improvements more easily and faster and with fewer PCB redesigns than a digital logic gate approach or competing microcontrollers with more fixed function pins.

3.4 SCHEMATIC DIAGRAM OF THE DESIGN

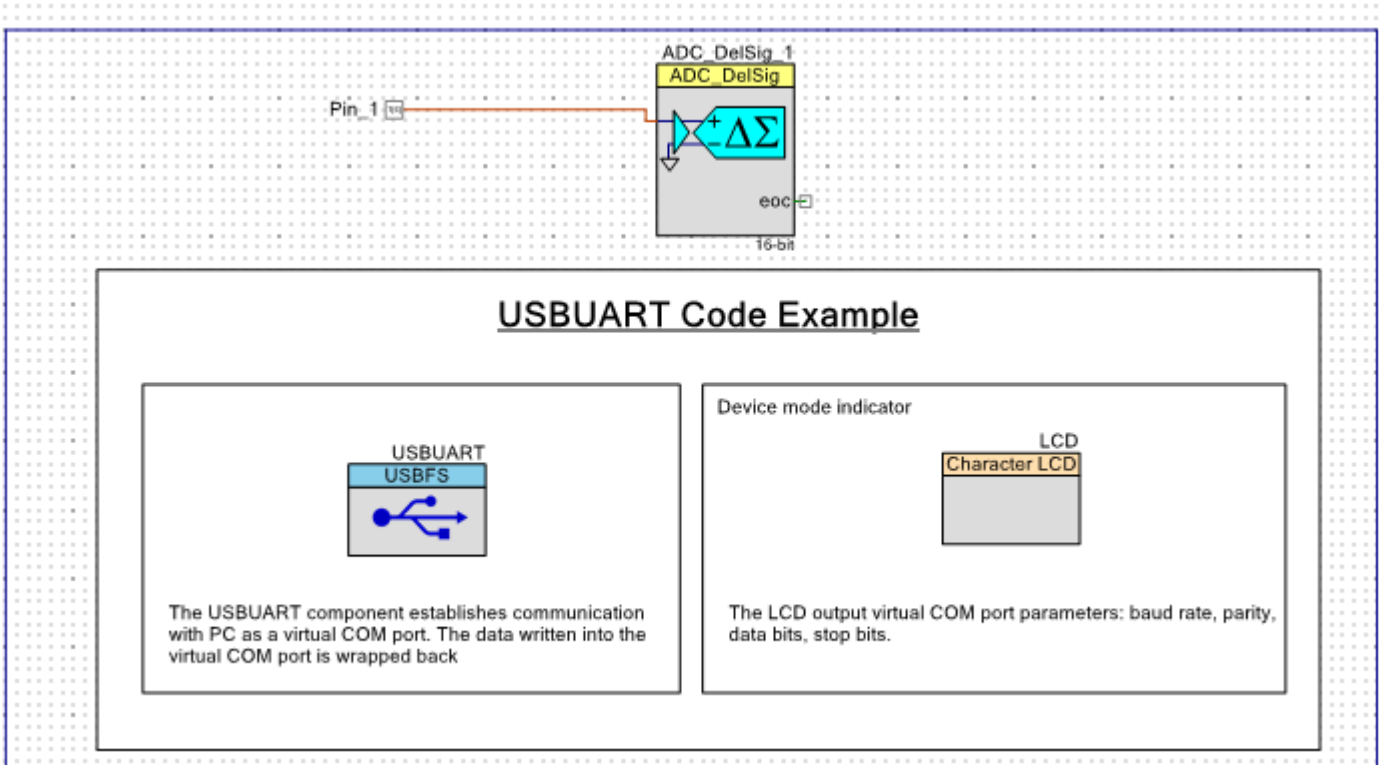


Fig 6 : schematic diagram of the PSoc chip connections

3.5 SIGMA _DELTA ADC (ANALOD TO DIGITAL CONVERSION)

3.5.1 WORKING PRINCIPLE OF SIGMA-DELTA ADC

Sigma Delta ADC consists of two main blocks, Sigma Delta modulator and digital decimeter. Sigma Delta modulator consist of difference amplifier, integrator, comparator and 1-bit DAC. Digital decimeter is used for digital filtering and down sampling.

Analog signal which is to be digitized is applied to the non inverting terminal of difference amplifier. Inverting terminal of difference amplifier is connected with either +Vref or –Vref depends on output of 1-bit DAC. Output of difference amplifier is integrated using integrator as shown in diagram. Output of integrator is applied to the non inverting terminal of comparator. In this case comparator works as 1-bit ADC and produces output as 1 or 0.
Output of comparator is connected to the 1-bit DAC. DAC is used to connect either +Vref or –Vref to the inverting terminal of difference amplifier.
DACs output is them again subtracted from analog input. This process is continuous in closed loop. After each loop 1-bit ADC produces 1 or 0. Density of '1' depends on analog voltage supplied. If analog voltage is high then density will be high and if analog signal is low density of '1' will be low. Output of 1-bit ADC is also connected to the digital decimeter. It is used for digital filtering and down sampling. It produces n-bit digital output in binary format.

3.5.2Advantages of Sigma Delta ADC

- Sigma Delta ADC is inexpensive since all circuitry within the converter is digital.
- The output of sigma delta ADC is inherently linear but it has little differential non linearity.
- It do not require sample and hold circuit. It is because due to high sampling rate and low precision.

3.5.3 Disadvantages

- It is limited to high resolution and very low frequency applications.
- It takes quite long time for producing first digital output because of digital filtering and down sampling.
- It is not possible to use Sigma Delta ADC for multiplexed ac input signals.

3.5.4 DELTA SIGMA ANALOG TO DIGITAL CONVERTER(ADC Del_Sig)

The Delta Sigma Analog to Digital Converter (ADC_DelSig) provides a low-power, low-noise front end for precision measurement applications. You can use it in a wide range of applications, depending on resolution, sample rate, and operating mode. It can produce 16-bit audio; high speed and low resolution for communications processing; and high-precision 20-bit low-speed conversions for sensors such as strain gauges, thermocouples, and other high-precision sensors. When processing audio information, the

ADC_DelSig is used in a continuous operation mode. When used for scanning multiple sensors, the ADC_DelSig is used in one of the multi-sample modes. When used for single-point high-resolution measurements, the ADC_DelSig is used in single-sample mode. Delta-sigma converters use oversampling to spread the quantization noise across a wider frequency spectrum. This noise is shaped to move most of it outside the input signal's bandwidth. An internal low-pass filter is used to filter out the noise outside the desired input signal bandwidth. This makes delta-sigma converters good for both high-speed medium-resolution (8 to 16 bits) applications, and low-speed high-resolution (16 to 20 bits) applications. The sample rate can be adjusted between 10 and 384000 samples per second, depending on mode and resolution. Choices of conversion modes simplify interfacing to single streaming signals such as audio, or multiplexing between multiple signal sources. The ADC_DelSig is composed of three blocks: an input amplifier, a third-order delta-sigma modulator, and a decimator (see Figure 1). The input amplifier provides a high-impedance input and a user-selectable input gain. The decimator block contains a four-stage CIC decimation filter and a post-processing unit. The CIC filter operates on the data sample directly from the modulator. The post-processing unit optionally performs gain, offset, and simple filter functions on the output of the CIC decimator filter.

ADC_ DELSIG BLOCK DIGRAM

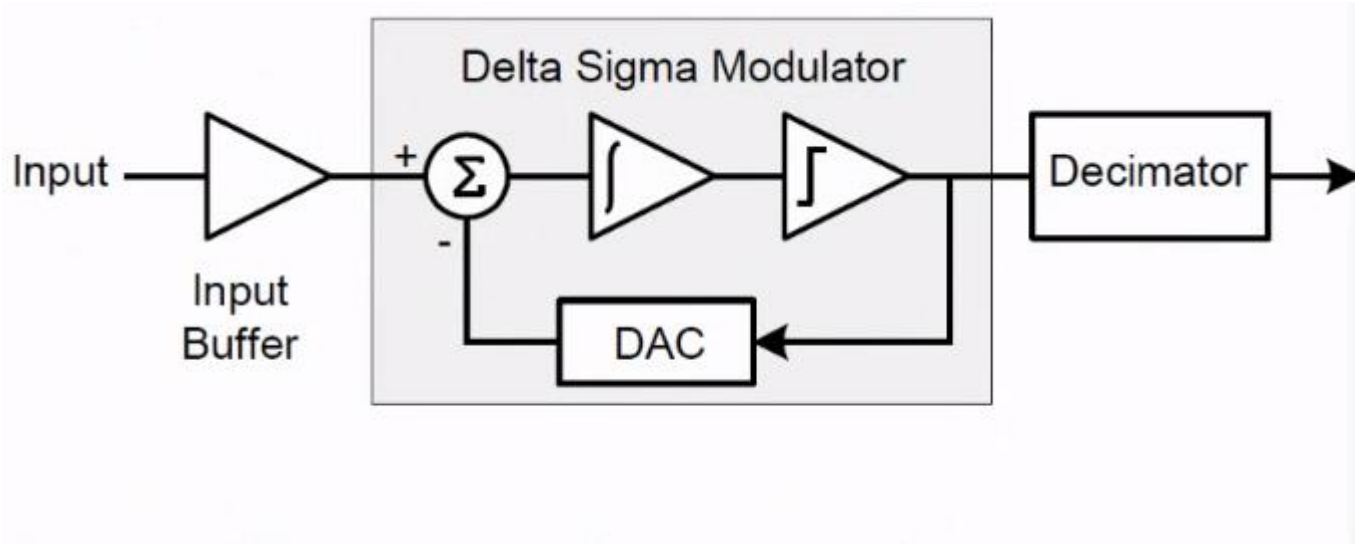


Fig 7: block diagram of ADC_del sig

3.5.5 FEATURES

- Selectable resolutions, 8 to 20 bits.
- Eleven input ranges for each resolution

- Sample rate 8 sps to 384 ksps
- Operational Modes
    - Single sample
    - Multi-sample
    - Continuous mode
    - Multi-sample (Turbo)

- High input impedance input buffer
    - Selectable input buffer gain (1, 2, 4, 8) or input buffer bypass
- Multiple internal or external reference options
- Up to four run-time ADC configurations
- Automatic power configuration

3.5.6 APPLICATION PROGRAMMING INTERFACE

By default, PSoC Creator assigns the instance name "ADC_DelSig_1" to the first instance of a Component in a given design. You can rename the instance to any unique value that follows the syntactic rules for identifiers. The instance name becomes the prefix of every global function name, variable, and constant symbol. For readability, the instance name used in the following table is "ADC".

Function	Description
ADC_Start()	Sets the initVar variable, calls the ADC_Init() function, and then calls the ADC_Enable() function.
ADC_Stop()	Stops ADC conversions and powers down
ADC_SetBufferGain()	Selects input buffer gain (1,2,4,8)
ADC_StartConvert()	Starts conversion
ADC_StopConvert()	Stops conversions
ADC_IRQ_Enable()	Enables interrupts at the end of conversion
ADC_IRQ_Disable()	Disables interrupts
ADC_IsEndConversion()	Returns a nonzero value if conversion is complete
ADC_GetResult8()	Returns an 8-bit conversion result
ADC_GetResult16()	Returns a 16-bit conversion result
ADC_GetResult32()	Returns a 32-bit conversion result
ADC_Read8()	Starts ADC conversions, waits for the conversion to be complete, stops ADC conversion and returns the signed 8-bit value of result.
ADC_Read16()	Starts ADC conversions, waits for the conversion to be complete, stops ADC conversion and returns the signed 16-bit value of result.
ADC_Read32()	Starts ADC conversions, waits for the conversion to be complete, stops ADC conversion and returns the signed 32-bit value of result.
ADC_SetOffset()	Sets the offset used by the ADC_CountsTo_mVolts(), ADC_CountsTo_uVolts(), and ADC_CountsTo_Volts() functions.
ADC_SelectConfiguration()	Sets one of up to four ADC configurations
ADC_SetGain()	Sets the gain used by the ADC_CountsTo_mVolts(), ADC_CountsTo_uVolts(), and ADC_CountsTo_Volts() functions.

ADC_CountsTo_mVolts()	Converts ADC counts to millivolts
ADC_CountsTo_uVolts()	Converts ADC counts to microvolts
ADC_CountsTo_Volts()	Converts ADC counts to floating point volts
ADC_Sleep()	Stops ADC operation and saves the user configuration
ADC_Wakeup()	Restores and enables the user configuration
ADC_Init()	Initializes or restores the ADC using the Configure dialog settings
ADC_Enable()	Enables the ADC
ADC_SaveConfig()	Saves the current configuration
ADC_RestoreConfig()	Restores the configuration
ADC_SetCoherency()	Sets the coherency register
ADC_SetGCOR()	Calculates a new GCOR value and sets the GCOR registers with this new value
ADC_ReadGCOR()	Returns the normalized GCOR register values

Table 3 : programming interface command description

3.6 USB – UART

3.6.1 USB :

Short for universal serial bus, USB (pronounced yoo-es-bee) is a plug and play interface that allows a computer to communicate with peripheral and other devices. USB-connected devices cover a broad range; anything from keyboards and mice to music players and flash drives. USB may also send power to certain devices, such as powering smartphones and tablets and charging their batteries.

3.6.2 UART :

A universal asynchronous receiver-transmitter (UART) is a computer hardware device for asynchronous serial communication in which the data format and transmission speeds are configurable. It sends data bits one by one, from the least significant to the most significant, framed by start and stop bits so that precise timing is handled by the communication channel. The electric signaling levels are handled by a driver circuit external to the UART. Two common signal levels are RS-232, a 12-volt system, and RS-485, a 5-volt system. Early teletypewriters used current loops. It was one of the earliest computer communication devices, used to attach teletypewriters for an operator console. It was also an early hardware system for the Internet. A UART is usually an individual (or part of an) integrated circuit (IC) used for serial communications over a computer or peripheral device serial port. One or more UART peripherals are commonly integrated in microcontroller chips. Specialized UARTs are used for automobiles, smart cards and SIMs.

3.6.3 USBUART (CDC)

The PsoC Creator Component Catalog contains a Schematic Macro implementation of a communications device class (CDC) interface (also known as USBUART). This is a USBFS Component with the descriptors configured to implement a CDC interface. This allows you to use a CDC-enabled USBFS Component with minimal configuration required.

To start a USBUART-based project, drag the USBUART Schematic Macro 26print26d 'USBUART (CDC Interface)' from the Component Catalog onto your design. This macro has already been configured to function as a CDC device. See the Component Parameters section of this datasheet for information about modifying the parameters of this interface, such as the VID, PID, and String Descriptors.
The CDC device requires drivers to be installed. The drivers can be found in the generated sources folder of your project:
<PROJECT_NAME>.cydsn\Generated_Source\<PSOC_NAME>\<INSTANCE_NAME>_cdc.inf
See the USBFS_UART code example for detailed steps on how to install a driver.

To add and configure communications and data interface descriptors for the USBUART, open the Configure USBFS dialog and click the **CDC Descriptor** tab.

3.6.4 FEATURES

- USB Full Speed device interface driver
- Support for interrupt, control, bulk, and isochronous transfer types
- Run time support for description set selectio
- USB string descriptions
- USB HID class support
- Bootloader support
- Audio class support (See the USBFS Audio section)
- MIDI devices support (See the USBFS MIDI section)
- Communications device class (CDC) support (See the USBUART (CDC) section)
- Mass storage device class (MSC) support (See the USBFS MSC section)

3.6.5 Enable CDC API

This option enables generation of CDC APIs. The list of generated APIs is provided in the section USBUART (CDC) .

CDC Descriptors List

This area allows you to add CDC descriptors. Detailed information on the CDC descriptors is available in the Universal Serial Bus Class Definitions for Communication Devices.

Item Value

This window allows you select a value that is appropriate for the currently selected CDC Descriptor item. The parameters in these windows are context based and will vary depending upon the item value selected in the CDC Descriptors List window.

To Add CDC Descriptors

- Select the CDC Descriptors root item in the tree on the left.
- Under the CDC Descriptors List on the right, select either the Communications or Data interface.
- Click Add to add the descriptor to the tree on the left.
- You can rename the CDC Interface x title by selecting a node and clicking on it again or by using of Rename context menu item.

To Add Functional Descriptors

- Select the appropriate Communications Alternate Settings x item in the tree on the left.
- Under the CDC Descriptors List on the right, select one of the items under Functional Descriptors as appropriate.
- Under Item Value, enter the appropriate values under Specific.
- Click Add to add the descriptor to the tree on the left.

To Add Endpoint Descriptors

- Select the appropriate Communications Alternate Settings x or Data Alternate Settings x item in the tree on the left.
- Under the CDC Descriptors List on the right, select the Endpoint Descriptor item.
- Under Item Value, enter the appropriate values under Specific.
- Click Add to add the descriptor to the tree on the left.

To Add the Configured CDC Interface Descriptor to the Device Descriptor Tree

- Go to the Device Descriptor tab.
- Select the Configuration Descriptor to which a new interface will belong.
- Click the Add Interface tool button, choose CDC, and select the appropriate item to add.

CDC interfaces are disabled in the Device Descriptor tab list because they can only be edited on the CDC Descriptor tab.

3.6.6 USBUART (CDC) FUNCTIONS

The following high-level APIs are available when the **Enable CDC API** option in the **CDC Descriptor** tab is selected. These APIs do not support DMA with Automatic Memory Management.

Function	Description
USBUART_CDC_Init()	Initializes the CDC interface to be ready for the receive data from the PC
USBUART_PutData()	Sends a specified number of bytes from the location specified by a pointer to the PC

USBUART_PutString()	Sends a null terminated string to the PC
USBUART_PutChar()	Writes a single character to the PC
USBUART_PutCRLF()	Sends a carriage return (0x0D) and line feed (0x0A) to the PC
USBUART_GetCount()	Returns the number of bytes that were received from the PC
USBUART_CDCIsReady()	Returns a nonzero value if the Component is ready to send more data to the PC
USBUART_DataIsReady()	Returns a nonzero value if the Component received data or received a zero-length packet
USBUART_GetData()	Gets a specified number of bytes from the input buffer and places them in a data array specified by the passed pointer
USBUART_GetAll()	Gets all bytes of received data from the input buffer and places them into a specified data array
USBUART_GetChar()	Reads one byte of received data from the buffer
USBUART_IsLineChanged()	Returns the clear-on-read status of the line
USBUART_GetDTERate()	Returns the data terminal rate set for this port in bits per second
USBUART_GetCharFormat()	Returns the number of stop bits
USBUART_GetParityType()	Returns the parity type for the CDC port
USBUART_GetDataBits()	Returns the number of data bits for the CDC port
USBUART_GetLineControl()	Returns the line control bitmap
USBUART_SendSerialState()	Sends the serial state notification to the host using the interrupt endpoint
USBUART_GetSerialState()	Returns the current serial state value
USBUART_SetComPort()	If multiple COM ports are instantiated, this function selects which COM port user wish to address.
USBUART_GetComPort()	If multiple COM ports are instantiated, this function returns the current selected COM port that the user is addressing.
USBUART_NotificationIsReady()	Returns a nonzero value if the Component is ready to send more notification data to the host

Table 4 : USB _UART Functional commands description

3.6.7 DC AND AC ELECTRICAL CHARACTERISTICS

Specifications are valid for –40 °C ≤ TA ≤ 85 °C and TJ ≤ 100 °C, except where noted. Specifications are valid for 1.71 V to 5.5 V, except where noted.

USB DC SPECIFICATIONS

Parameter	Description	Conditions	Min	Typ	Max	Units
VUSB_5 ■	Device supply for USB operation	USB configured, USB regulator enabled PsoC 3/5LP	4.35	–	5.25	V
		USB configured,	4.5	–	5.5	V

		USB regulator enabled PsoC4200L				
VUSB_3.3 ■		USB configured, USB regulator bypassed	3.15	–	3.6	V
VUSB_3		USB configured, USB regulator bypassed	2.85	–	3.6	V
IUSB_Configured	Device supply current in device active mode, bus clock and IMO = 24 MHz	VDDD = 5 V	–	10	–	mA
		VDDD = 3.3 V	–	8	–	mA
IUSB_Suspended	Device supply current in device sleep mode	VDDD = 5 V, connected to USB host, PICU configured to wake on USB resume signal	–	0.5	–	mA
		VDDD = 5 V, disconnected from USB host	–	0.3	–	mA
		VDDD = 3.3 V, connected to USB host, PICU configured to wake on USB resume signal	–	0.5	–	mA
		VDDD = 3.3 V, disconnected from USB host	–	0.3	–	mA

Table 5: USB DC specifications

USB DRIVER AC SPECIFICATIONS

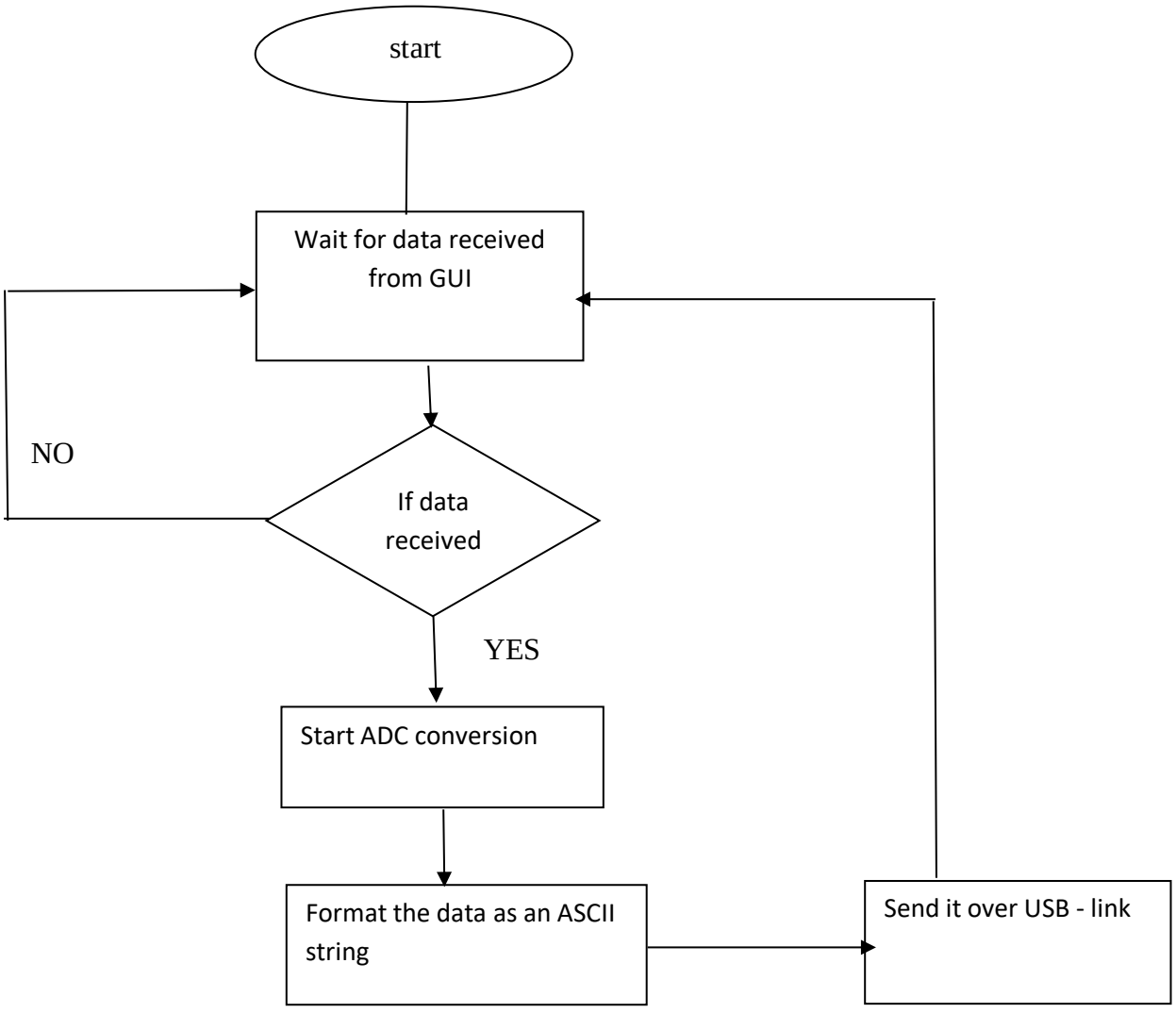
Parameter	Description	Conditions	Min	Typ	Max	Units
Tr	Transition rise time	-	4	-	20	ns
Tf	Transition fall time	-	4	-	20	Ns
TR	Rise/fall time matching	-	90%	-	111%	TR
VCRS	Output signal crossover	-	1.3	-	V2	v

	voltage					

Table 6 : USB AC Specifications

4 .PsoC PROGRAM

4.1 PSOC PROGRAM – FLOW CHART



4.2 PSOC KEIL-C PROGRAM

```
#include<project.h>
#include "stdio.h"

#if defined (__GNUC__)
/* Add an explicit reference to the floating point printf library */
/* to allow usage of the floating point conversion specifiers. */
/* This is not linked in by default with the newlib-nano library. */
asm (".global _printf_float");
#endif

#define USBFS_DEVICE    (0u)

/* The buffer size is equal to the maximum packet size of the IN and OUT bulk
* endpoints.
*/
#define USBUART_BUFFER_SIZE (64u)
#define LINE_STR_LENGTH   (20u)

char8* parity[] = {"None", "Odd", "Even", "Mark", "Space"};
char8* stop[]   = {"1", "1.5", "2"};
char * Accelval[6];
uint16 sensorout;


/*******************************************************************************
* Function Name: main
********************************************************************************
*
* Summary:
*  The main function performs the following actions:
*   1. Waits until VBUS becomes valid and starts the USBFS component which is
*      enumerated as virtual Com port.
*   2. Waits until the device is enumerated by the host.
*   3. Waits for data coming from the hyper terminal and sends it back.
*   4. PsoC3/PsoC5LP: the LCD shows the line settings.
*
* Parameters:
*  None.
*
* Return:
*  None.
*
*******************************************************************************/
int main()
{
uint16 count;
uint8 buffer[USBUART_BUFFER_SIZE];
//uint8 flag=0;

#if (CY_PSOC3 || CY_PSOC5LP)
```

```
uint8 state;
char8 lineStr[LINE_STR_LENGTH];

//LCD_Start();
    ADC_DelSig_1_Start();
#endif/* (CY_PSOC3 || CY_PSOC5LP) */

CyGlobalIntEnable;

/* Start USBFS operation with 5-V operation. */
USBUART_Start(USBFS_DEVICE, USBUART_5V_OPERATION);

for(;;)
    {



/* Host can send double SET_INTERFACE request. */
if (0u != USBUART_IsConfigurationChanged())
        {
/* Initialize IN endpoints when device is configured. */
if (0u != USBUART_GetConfiguration())
            {
/* Enumeration is done, enable OUT endpoint to receive data
                 * from host. */
USBUART_CDC_Init();
            }
        }

/* Service USB CDC when device is configured. */
if (0u != USBUART_GetConfiguration())
        {
/* Check for input data from host. */
if (0u != USBUART_DataIsReady())
            {
/* Read received data and re-enable OUT endpoint. */
                count = USBUART_GetAll(buffer);
                ADC_DelSig_1_StartConvert();
if(ADC_DelSig_1_IsEndConversion(ADC_DelSig_1_WAIT_FOR_RESULT))
                {
sensorout=ADC_DelSig_1_GetResult32();//read the numerical analog value
32print((char *)Accelval,"%u",sensorout);//prepare an ascii string to send over usb
strcat(Accelval,"\n");
                }

if (0u != count)
                {
/* Wait until component is ready to send data to host. */
while (0u == USBUART_CDCIsReady())
                    {
                    }
USBUART_PutString(Accelval);

/* Send data back to host. */
//USBUART_PutData(buffer, count);

//                    if(flag==0)
//                    {
//                        USBUART_PutString("1000\n");
//                        CyDelay(50);
//                        flag=1;
```

```
//                    }
//                    else
//                    {
//                        USBUART_PutString("500\n");
//                        CyDelay(50);
//                        flag=0;
//                    }



/* If the last sent packet is exactly the maximum packet
                    *  size, it is followed by a zero-length packet to assure
                    *  that the end of the segment is properly identified by
                    *  the terminal.
                    */
if (USBUART_BUFFER_SIZE == count)
                    {
/* Wait until component is ready to send data to PC. */
while (0u == USBUART_CDCIsReady())
                        {
                        }

/* Send zero-length packet to PC. */
USBUART_PutData(NULL, 0u);
                    }
                }
            }



//        #if (CY_PSOC3 || CY_PSOC5LP)
//            /* Check for Line settings change. */
//            state = USBUART_IsLineChanged();
//            if (0u != state)
//            {
//                /* Output on LCD Line Coding settings. */
//                if (0u != (state & USBUART_LINE_CODING_CHANGED))
//                {
//                    /* Get string to output. */
//                    33print(lineStr,"BR:%4ld %d%c%s", USBUART_GetDTERate(),\
//                                    (uint16) USBUART_GetDataBits(),\
//                                    parity[(uint16) USBUART_GetParityType()][0],\
//                                    stop[(uint16) USBUART_GetCharFormat()]);
//
//                    /* Clear LCD line. */
//                    LCD_Position(0u, 0u);
//                    LCD_PrintString("                    ");
//
//                    /* Output string on LCD. */
//                    LCD_Position(0u, 0u);
//                    LCD_PrintString(lineStr);
//                }
//
//                /* Output on LCD Line Control settings. */
//                if (0u != (state & USBUART_LINE_CONTROL_CHANGED))
//                {
//                    /* Get string to output. */
//                    state = USBUART_GetLineControl();
```

```
//                                34print(lineStr,"DTR:%s,RTS:%s",    (0u  !=  (state  &
USBUART_LINE_CONTROL_DTR)) ? "ON" : "OFF",
//                                                                     (0u  !=  (state  &
USBUART_LINE_CONTROL_RTS)) ? "ON" : "OFF");
//
//                    /* Clear LCD line. */
//                    LCD_Position(1u, 0u);
//                    LCD_PrintString("                    ");
//
//                    /* Output string on LCD. */
//                    LCD_Position(1u, 0u);
//                    LCD_PrintString(lineStr);
//                }
//            }
//        #endif /* (CY_PSOC3 || CY_PSOC5LP) */
        }
    }
}


/* [] END OF FILE */
```

4.2.1 PSOC PROGRAM COMMANDS DESCRIPTION

i. USBUART_PutString

Description: Sends a null terminated (RAM) string to the PC. Refer to Sending Packets of Length 64 Bytes and More when sending large packets.

C Prototype:

void USBUART_PutString(BYTE * pStr)

Assembler:

mov A,>pStr ; Load MSB part of pointer to RAM based null

; terminated string mov X,

Mov X, <pStr ; Load LSB part of pointer to RAM based null

; terminated string

lcallUSBUART_PutString ; Call function to send string out

Parameters:

pStr: Pointer to the string to be sent to PC. The MSB is passed in the Accumulator and the LSB is passed in the X register. The maximum string length is 64 bytes including the terminating null character.

Return Value:

None

Side Effects:

The A and X registers may be modified by this or future implementations of this function. The same is true for all RAM page pointer registers in the large memory model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently only the IDX_PP and the CUR_PP page pointer registers are modified.

ii. Unit 8 USBUART_GetConfiguration(void)

Description:
This function gets the current configuration of the USB device

Return Value:
uint8: Returns the currently assigned configuration. Returns 0 if the device is not configured.

iii. uint8 USBUART_IsConfigurationChanged(void)

Description:
This function returns the clear-on-read configuration state. It is useful when the host sends double SET_CONFIGURATION requests with the same configuration number or changes alternate settings of the interface.
After configuration has been changed the OUT endpoints must be enabled and IN endpoint must be loaded with data to start communication with the host.

Return Value:
uint8: Returns a nonzero value when a new configuration has been changed; otherwise, it returns zero.

iv. void USBUART_CDC_Init(void)

Description:
This function initializes the CDC interface to be ready to receive data from the PC. The API set active communication port to 0 in the case of multiple communication port support. This API should be called after the device has been started and configured using USBUART_Start() API to initialize and start the USBFS Component operation. Then call the USBUART_GetConfiguration() API to wait until the host has enumerated and configured the device. For example:

```
USBUART_Start(…);
while(0 == USBUART_GetConfiguration())
{
}
USBUART_CDC_Init();
```

Parameters: None

Return Value: None

Side Effects: None

v. uint8 USBUART_DataIsReady(void)

Description:

This function returns a nonzero value if the Component received data or received a zero-length packet. The USBUART_GetAll() or USBUART_GetData() API should be called to read data from the buffer and reinitialize the OUT endpoint even when a zero-length packet is received. These APIs will return zero value when zero-length packet is received.

Parameters: None

Return Value: uint8: If the OUT packet is received, this function returns a nonzero value. Otherwise, it returns zero.

Side Effects: None

vi. uint8 USBUART_IsLineChanged(void)

Description: This function returns the clear-on-read status of the line. It returns not zero value when the host sends updated coding or control information to the device. The USBUART_GetDTERate(), USBUART_GetCharFormat() or USBUART_GetParityType() or USBUART_GetDataBits() API should be called to read data coding information. The USBUART_GetLineControl() API should be called to read line control information.

Parameters: None

Return Value: uint8: If SET_LINE_CODING or CDC_SET_CONTROL_LINE_STATE requests are received, it returns a nonzero value. Otherwise, it returns zero.

Return Value	Description
USBUART_LINE_CODING_CHANGED	Line coding changed
USBUART_LINE_CONTROL_CHANGED	Line control changed

Side Effects : None

vii. uint8 USBUART_GetParityType(void)

Description: This function returns the parity type for the CDC port.

Parameters: None

Return Value: unit 8

Return Value	Description
USBUART_PARITY_NONE	None
USBUART_PARITY_ODD	Odd
USBUART_PARITY_EVEN	Even
USBUART_PARITY_MARK	Mark
USBUART_PARITY_SPACE	Space

viii. uint8

USBUART_GetDataBits(void)

Description: This function returns the number of data bits for the CDC port

Parameters: None

Return Value: uint8: Returns the number of data bits. The number of data bits can be 5, 6, 7, 8, or 16.

Side Effects: None

viii. uint16 USBUART_GetLineControl(void)

Description: This function returns the line control bitmap that the host sends to the device.

Parameters: None.

Return Value: uint8:

Return Value	Notes
USBUART_LINE_CONTROL_DTR	Indicates that a DTR signal is present. This signal corresponds to V.24 signal 108/2 and RS232 signal DTR.
USBUART_LINE_CONTROL_RTS	Carrier control for half-duplex modems. This signal corresponds to V.24 signal 105 and RS232 signal RTS.
RESERVED	The rest of the bits are reserved.

Side Effects : None

ix. uint16 USBUART_GetAll(uint8* pData)

Description:
This function gets all bytes of received data from the input buffer and places them into a specified data array. The USBUART_DataIsReady() API should be called first, to be sure that data is received from the host.

Parameters: uint8* pData: Pointer to the data array where data will be placed.

Return Value: uint16: Number of bytes received. The maximum amount of the received at a time data is 64 bytes.

Side Effects: None

x. uint32 USBUART_GetDTERate(void)

Description : This function returns the data terminal rate set for this port in bits per second.

Parameters: None

Return Value: uint32: Returns a value of the data rate in bits per second

Side Effects: None

xi. uint8 USBUART_GetCharFormat(void)

Description: This function returns the number of stop bits.

Parameters: None

Return Value: uint8: Returns the number of stop bits.

Return Value	Description
USBUART_1_STOPBIT	1 stop bit
USBUART_1_5_STOPBITS	1,5 stop bits
USBUART_2_STOPBITS	2 stop bits

Side Effects: None

xii. USBUART_Start

Description :
Performs all required operations to start the USBUART Device User Module.

Parameters:
bVoltage is the operating voltage of the chip, passed in the Accumulator. This determines whether the voltage regulator is enabled for 5 V operation or pass through mode is used for 3.3 V operation. Symbolic names are given in C and assembly, and their associated values are listed in the following table.

Mask	Value	Description
USBUART_3V_OPERATION	0x02	Disable the voltage regulator and pass-through Vcc for pull up
USBUART_5V_OPERATION	0x03	Enable the voltage regulator and use the regulator for pull up

Table 7 : parameters of USBUART _Start command

Return Value: None

Side Effects:
This function or its future implementations can modify the A and X registers. The same is true for all RAM page pointer registers in the large memory model. When necessary, it is the calling function's responsibility to preserve the values across calls to fastcall16 functions. Currently only the IDX_PP and the CUR_PP page pointer registers are modified.

xiii. void LCD_Char_Position(uint8 row, uint8 column)

Description: Moves the cursor to the location specified by arguments **row** and **column.**

Parameters: uint8 row: The row number at which to position the cursor. Minimum value is zero.
Uint8 column: The column number at which to position the cursor. Minimum value is zero.

Return Value: None

Side Effects: None

xiv. void LCD_Char_PrintString(char8 const string[])

Description: Writes a null-terminated string of characters to the screen beginning at the current cursor location.

Parameters: char8 conststring[]: Null-terminated array of ASCII characters to be displayed on the LCD module's screen.

Return Value: None

Side Effects: None

xv. void LCD_Char_Start(void)

Description:
This function initializes the LCD hardware module as follows:

- • Enables 4-bit interface
- • Clears the display
- • Enables auto cursor increment
- • Resets the cursor to start position

It also loads a custom character set to LCD if it was defined in the customizer's GUI.

Parameters: None

Return Value: None

Side Effects: None

xvi. void ADC_Start(void)

Description:
Sets the initVar variable, calls the ADC_Init() function, and then calls the ADC_Enable() function.
This function configures and powers up the ADC, but does not start conversions. By default, the ADC is configured for Config1. Use the ADC_SelectConfiguration() function to select an alternate configuration afterward.
Side Effects:
Enables internal interrupts. Reading the result clears the interrupt. Refer to Interrupt Service Routine section for information how to remove internal interrupt.

xvii. void ADC_StartConvert(void)

Description:

Forces the ADC to initiate a conversion.In Single Sample mode, call this API to start a single conversion. When the conversion completes, use ADC_IsEndConversion() API to check or wait on this event, the ADC will halts. If the ADC_StartConvert() function is called while the conversion is in progress, the next conversion start is queued and a new conversion will start after finishing the current conversion. If you want to start a new conversion without waiting for the current conversion to finish, then stop the current conversion by calling ADC_StopConvert(). After stopping the conversion, restart the conversion by calling ADC_StartConvert().In Multi Sample, Continuous or Multi Sample (Turbo) modes, call this API to start continuous ADC conversions until either the ADC_StopConvert() or ADC_Stop() functions are executed.

xviii. uint8 ADC_IsEndConversion(uint8 retMode)

Description:

Checks for ADC end of conversion. This function provides the programmer with two options. In one mode this function immediately returns with the conversion status. In the other mode, the function does not return (blocking) until the conversion has completed.

Parameters:

uint8 retMode: Check conversion return mode. See the following table for options.

Options	Description
ADC_RETURN_STATUS	Immediately returns conversion result status.
ADC_WAIT_FOR_RESULT	Does not return until ADC conversion is complete.

Return Value:

uint8: If a nonzero value is returned, the last conversion has completed. If the returned value is zero, the ADC is still calculating the last result.

Side Effects:

When the EOC output is used to trigger the DMA to read the ADC result, the ADC_IsEndConversion() function should not be used. This is because, reading the output register with DMA clears the ADC's conversion complete flag and this API may not return when in the ADC_WAIT_FOR_RESULT mode.

xix. int32 ADC_GetResult32(void)

Description:

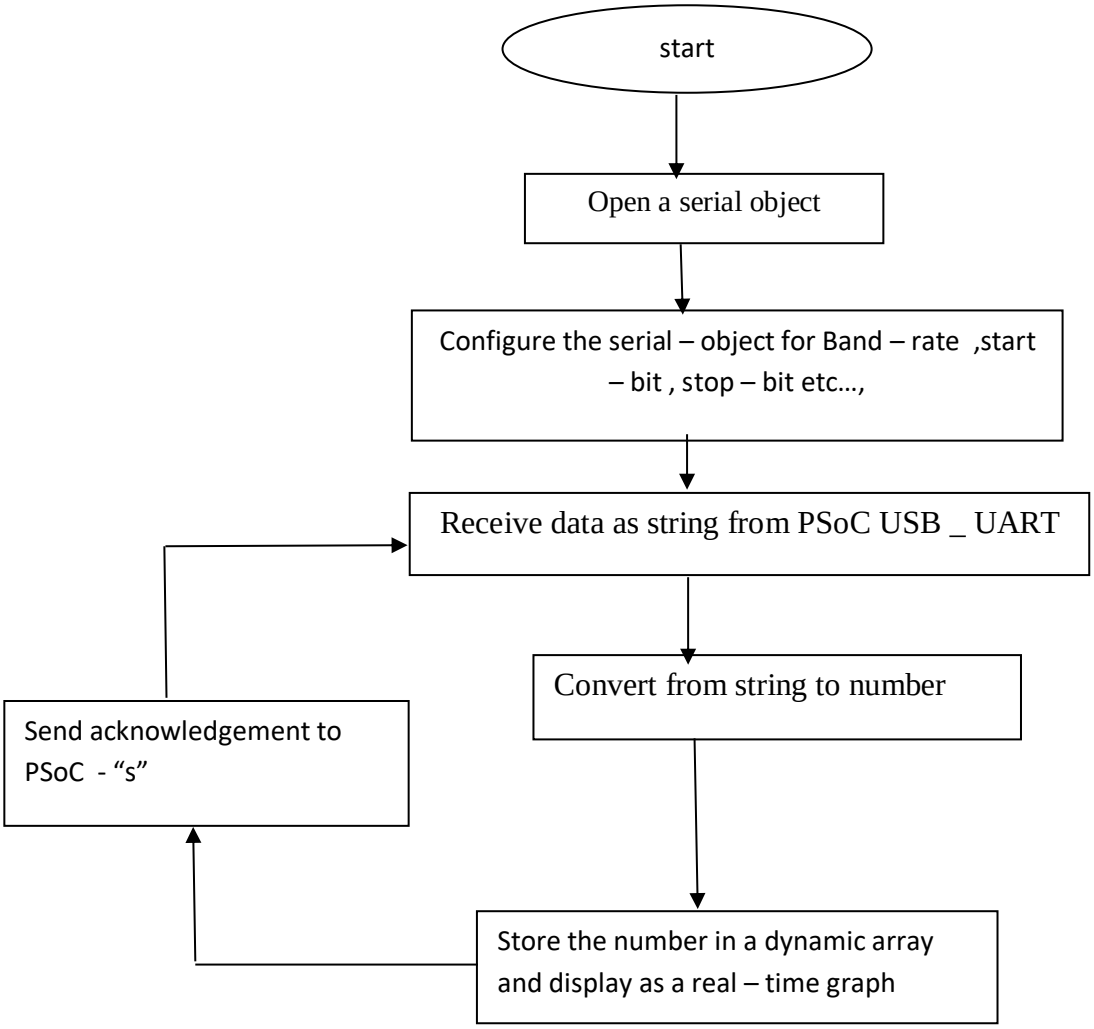
Returns a 32-bit result for a conversion that has a resolution of 8 to 20 bits.

Return Value:

int32: Result of the last ADC conversion.

5. GUI BASED MATLAB – INTERFACE

5.1 MATLAB PROGRAM – FLOW CHART



5.2 MATLAB PROGRAM

```
clc
clear all
% s = serial('COM8','baudrate',9600);
%s = serial('COM5','baudrate',57600);
s = serial('COM8','baudrate',9600);
set(s, 'InputBufferSize', 30);
set(s,'Terminator', 'LF');

s.BytesAvailableFcnMode='terminator';
%s.BytesAvailableFcn=@VentRead;
fopen(s);
pause on;

x = linspace(0,10,100);
k=length(x);
waveform=zeros(1,10000);
%y = sin(x);
y=zeros(1,k);
figure(1);
% h = plot(x,y);
h= plot(x,y,'-ro','LineWidth',2,...
'MarkerEdgeColor','r',...
'MarkerFaceColor','w',...
'MarkerSize',10);
set(h,'XdataSource','x');
set(h,'YdataSource','y');
ylim([0, 40000]);

spectra=fft(y);

figure(2);
j= plot(x,spectra,'-ko','LineWidth',2,...
'MarkerEdgeColor','k',...
'MarkerFaceColor','w',...
'MarkerSize',5);
set(j,'XdataSource','x');
set(j,'YdataSource','spectra');
ylim([-1000, 100000]);
xlim([0.25,5]);




%y = sin(x.^3);
fori=1:100
fprintf(s,'s');
%pause(0.1);
%b='4193';
 b=fgets(s);

 y(k)=str2double(b);

 waveform(i)=y(k);
```

```
refreshdata(1)
% pause(0.1);
y = circshift(y,[0,-1]);
spectra=fft(y);
refreshdata(2)

flushinput(s);
pause(0.01);

end
fclose(s);
```

5.3 MATLAB PROGRAM COMMANDS DESCRIPTION

5.3.1 SERIAL OBJECT

The MATLAB program is based on SERIAL – OBJECT concept

serial	- Construct serial port object.
instrcb	- Wrapper for serial object callback.
instrfind	- Find communication interface objects with specified property values.
instrfindall	- Find all communication interface objects with specified

serial is both a directory and a function.

serial Construct serial port object.

S = serial('PORT') constructs a serial port object associated withport, PORT. If PORT does not exist or is in use you will not be able to connect the serial port object to the device.

In order to communicate with the device, the object must be connectedto the serial port with the FOPEN function.

When the serial port object is constructed, the object's Status property is closed. Once the object is connected to the serial port with the FOPEN function, the Status property is configured to open. Only one serial port object may be connected to a serial port at a time.

S = serial('PORT','P1',V1,'P2',V2,...) constructs a serial port objectassociated with port, PORT, and with the specified property values. If an invalid property name or property value is specified the object will not be created.

Note that the property value pairs can be in any format supported by the SET function, i.e., param-value string pairs, structures, andparam-value cell array pairs.

Example:

```
% To construct a serial port object:
  s1 = serial('COM1');
  s2 = serial('COM2', 'BaudRate', 1200);

% To connect the serial port object to the serial port:
```

```
fopen(s1)
fopen(s2)

% To query the device.
fprintf(s1, '*IDN?');
idn = fscanf(s1);

% To disconnect the serial port object from the serial port.
fclose(s1);
fclose(s2);
```

5.3.2FOPEN

fopen Connect serial port object to device.

fopen(OBJ) connects the serial port object, OBJ, to the device. OBJ can be an array of serial port objects.

Only one serial port object with the same configuration can be connected to an instrument at a time. For example, only one serial port object can be connected to the COM2 port at a time. If OBJ was successfully connected to the device, OBJ's Status property is configured to open otherwise the Status property remains configured to closed.

When OBJ is opened, any data remaining in the input buffer and the output buffer is flushed and the BytesAvailable, BytesToOutput, Values Received and Values Sent properties are reset to 0.

Some property values can only be verified after the connection to the device has been made. Examples include BaudRate, Flow Control, and Parity. If any of these properties are set to a value not supported by the device, an error will be returned and the object will not be connected to the device.

Some properties are read-only while the serial port object is open (connected) and must be configured before using fopen. Examples include InputBufferSize and OutputBufferSize.

An error will be returned if fopen is called on a serial port objectthat has a Status property value of Open.

The byte order of the device can be specified with OBJ's ByteOrder property.

If OBJ is an array of serial port objects and one of the objects cannot be connected to the device, the remaining objects in the array will be connected to the device and a warning will be displayed.

Example:

```
s = serial('COM1');
fopen(s);
fprintf(s, '*IDN?');
idn = fscanf(s);
fclose(s);
```

FCLOSE

fclose Close file.

ST = fclose(FID) closes the file associated with file identifier FID,which is an integer value obtained from an earlier call to FOPEN. Fclose returns 0 if successful or -1 if not. If FID does not represent an open file, or if it is equal to 0 (standard input), 1 (standard output), or 2 (standard error), fclose throws an error.

ST = fclose('all') closes all open files, except 0, 1 and 2.

5.3.3 clc Clear command window.

Clc clears the command window and homes the cursor.

Clear Clear variables and functions from memory.

Clear removes all variables from the workspace.

Clear VARIABLES does the same thing.

Clear GLOBAL removes all global variables.

Clear FUNCTIONS removes all compiled MATLAB and MEX-functions.

Calling clear FUNCTIONS decreases code performance and is usually unnecessary.

For more information, see the clear Reference page.

Clear ALL removes all variables, globals, functions and MEX links.

Clear ALL at the command prompt also clears the base import list.

Calling clear ALL decreases code performance and is usually unnecessary.

For more information, see the clear Reference page.

5.3.4 linspace Linearly spaced vector.

Linspace(X1, X2) generates a row vector of 100 linearly

equally spaced points between X1 and X2.

Linspace(X1, X2, N) generates N points between X1 and X2.

For N = 1, linspace returns X2.

5.3.5 zeros Zeros array.

Zeros(N) is an N-by-N matrix of zeros.

Zeros(M,N) or zeros([M,N]) is an M-by-N matrix of zeros.

Zeros(M,N,P,...) or zeros([M N P ...]) is an M-by-N-by-P-by-... array ofzeros.

Zeros(SIZE(A)) is the same size as A and all zeros.

Zeros with no arguments is the scalar 0

zeros(..., CLASSNAME) is an array of zeros of class specified by thestring CLASSNAME.

Zeros(..., ‘like’, Y) is an array of zeros with the same data type, sparsity,

and complexity (real or complex) as the numeric variable Y.

Example:

x = zeros(2,3,’int8’);

5.3.6 figure Create figure window.

Figure, by itself, creates a new figure window, and returns its handle.

Figure(H) makes H the current figure, forces it to become visible, and raises it above all other figures on the screen. If Figure H does not exist, and H is an integer, a new figure is created with handle H.

GCF returns the handle to the current figure.

Execute GET(H) to see a list of figure properties and their current values. Execute SET(H) to see a list of figure

properties and their possible values.

5.3.7 PAUSE WAIT FOR USER RESPONSE.

Pause(n) pauses for n seconds before continuing, where n can also be a fraction. The resolution of the clock is platform specific. Fractional pauses of 0.01 seconds should be supported on most platforms.

Pause causes a procedure to stop and wait for the user to strike any key before continuing.

Pause OFF indicates that any subsequent pause or pause(n) commands should not actually pause. This allows normally interactive scripts to run unattended.

Pause ON indicates that subsequent pause commands should pause.

Pause QUERY returns the current state of pause, either ‘on’ or ‘off’.

STATE = pause(...) returns the state of pause previous to processing the input arguments.

The accuracy of pause is subject to the scheduling resolution of the operating system you are using and also to other system activity. It cannot be guaranteed with 100% confidence, but only with some expected error. For example, experiments have shown that choosing N with a resolution of .1 (such as N = 1.7) leads to actual pause values that are correct to roughly 10% in the relative error of the fractional part. Asking for finer resolutions (such as .01) shows higher relative error.

Examples:

% Pause for 5 seconds

pause(5)

% Temporarily disable pause

pause off

pause(5) % Note that this does not pause

pause on

% Alternatively, disable/restore the state

pstate = pause('off')

pause(5) % Note that this does not pause

pause(pstate);

5.3.8 flushinput Remove data from input buffer.

Flush input(OBJ) removes remaining data from the interface object's input buffer and sets the Bytes Available property to 0 for the interface object, OBJ.

If flush input is called during an asynchronous (nonblocking) read operation, the data currently stored in the input buffer is flushed and the read operation continues. You can read data asynchronously from the instrument using the READASYNC function.

The input buffer is automatically flushed when you connect an object to the instrument with the FOPEN function.

Example:

g = gpib('ni', 0, 2);

fopen(g);

fprintf(g, 'Curve?')

readasync(g, 512);

flushinput(g);

fclose(g);

5.3.9 fft Discrete Fourier transform.

Fft(X) is the discrete Fourier transform (DFT) of vector X. Formatrices, the fft operation is applied to each column. For N-D arrays, the fft operation operates on the first non-singleton dimension.

Fft(X,N) is the N-point fft, padded with zeros if X has lessthan N points and truncated if it has more

fft(X,[],DIM) or fft(X,N,DIM) applies the fft operation across thedimension DIM.

For length N input vector x, the DFT is a length N vector X, with elements

N

X(k) = sum x(n)*exp(-j*2*pi*(k-1)*(n-1)/N), 1 <= k <= N.

n=1

The inverse DFT (computed by IFFT) is given by

N

x(n) = (1/N) sum X(k)*exp(j*2*pi*(k-1)*(n-1)/N), 1 <= n <= N.

k=1

5.3.9 refreshdata Refresh data in plot

refreshdata evaluates any data source properties in theplots in the current figure and sets the corresponding data

properties of each plot.

Refresh data(FIG) refreshes the data in figure FIG.

Refresh data(H) for a vector of handles H refreshes the data of the objects specified in H or the children of those

objects. Therefore, H can contain figure, axes, or plot object handles.

Refresh data(H,WS) evaluates the data source properties in the work space WS. WS can be ‘caller’ or ‘base’. The default work space is ‘base’.

5.3.10 str2double Convert string to double precision value

X = str2double(S) converts S, text that represents a real or complex scalar value, to a double precision number. S may contain digits a comma (thousands separator), a decimal point, a leading + or – sign,an

‘e’ preceding a power of 10 scale factor, and an ‘i’ for a complex unit. S can be a character vector or a string scalar.

If S does not represent a valid scalar value, str2double(S) returns NaN.

X = str2double(STR) converts the elements in string array STR to double.

The matrix X returned will be the same size as STR. NaN is returned for any string element that does not represent a valid scalar value.

X = str2double© converts the elements in the cell array of character vectors C to double. The matrix X returned will be the same size as C. NaN is returned for any cell that does not represent a valid scalar value. NaN is returned for individual cells in C that are cell arrays.

Examples

str2double(‘123.45e7’)

str2double(‘123 + 45i’)

str2double(‘3.14159’)

str2double(‘2.7i – 3.14’)

str2double({‘2.71’ ‘3.1415’})

str2double(‘1,200.34’)

5.3.11 end Terminate

scope of FOR, WHILE, SWITCH, TRY, and IF statements.

Without end’s, FOR, WHILE, SWITCH, TRY, and IF wait for further input.

Each end is paired with the closest previous unpaired FOR, WHILE,

SWITCH, TRY or IF and serves to terminate its scope.

End also marks the termination of a function, although inmost cases it is optional. End statements are required only in MATLAB files that employ one or more nested functions. Within such file, every function (including primary, nested, private, and sub functions) must be terminated with an end statement. You can terminate any function type with end, but doing so is not required unless the file contains a nested function.

End can also serve as the last index in an indexing expression. Inthat context, end = SIZE(X,k) when used as part of the k-th index. Examples of this use are, X(3:end) and X(1,1:2:end-1). When using end to grow an array, as in X(end+1) = 5, make sure X exists first.

End(A,K,N) is called for indexing expressions involving the object A when end is part of the K-th index out of N indices. For example,the expression A(end-1,☺ calls A's end method with end(A,1,2).

5.3.12 circshift Shift positions of elements circularly.

Y = circshift(X,K) where K is an integer scalar circularly shifts the elements in the array X by K positions. If X is a vector and K ispositive, then the values of X are circularly shifted from the beginning to the end. If K is negative, they are shifted from the end to the beginning. If X is a matrix, circshift shifts along columns. If X is anN-D array, circshift shifts along the first nonsingleton dimension.

Y = circshift(X,K,DIM) circularly shifts along the dimension DIM.

Y = circshift(X,V) circularly shifts the values in the array X by V elements. V is a vector of integers where the N-th element specifies the shift amount along the N-th dimension of array X.

Examples:

A = [1 2 3; 4 5 6; 7 8 9];

B = circshift(A,1) % circularly shifts first dimension values down by 1.

B = 7 8 9

1 2 3

4 5 6

B = circshift(A,[1 -1]) % circularly shifts first dimension values

% down by 1 and second dimension left by 1.

B = 8 9 7

2 3 1

5 6 4

6. RESULT AND CONCLUSION

6.0 RESULT

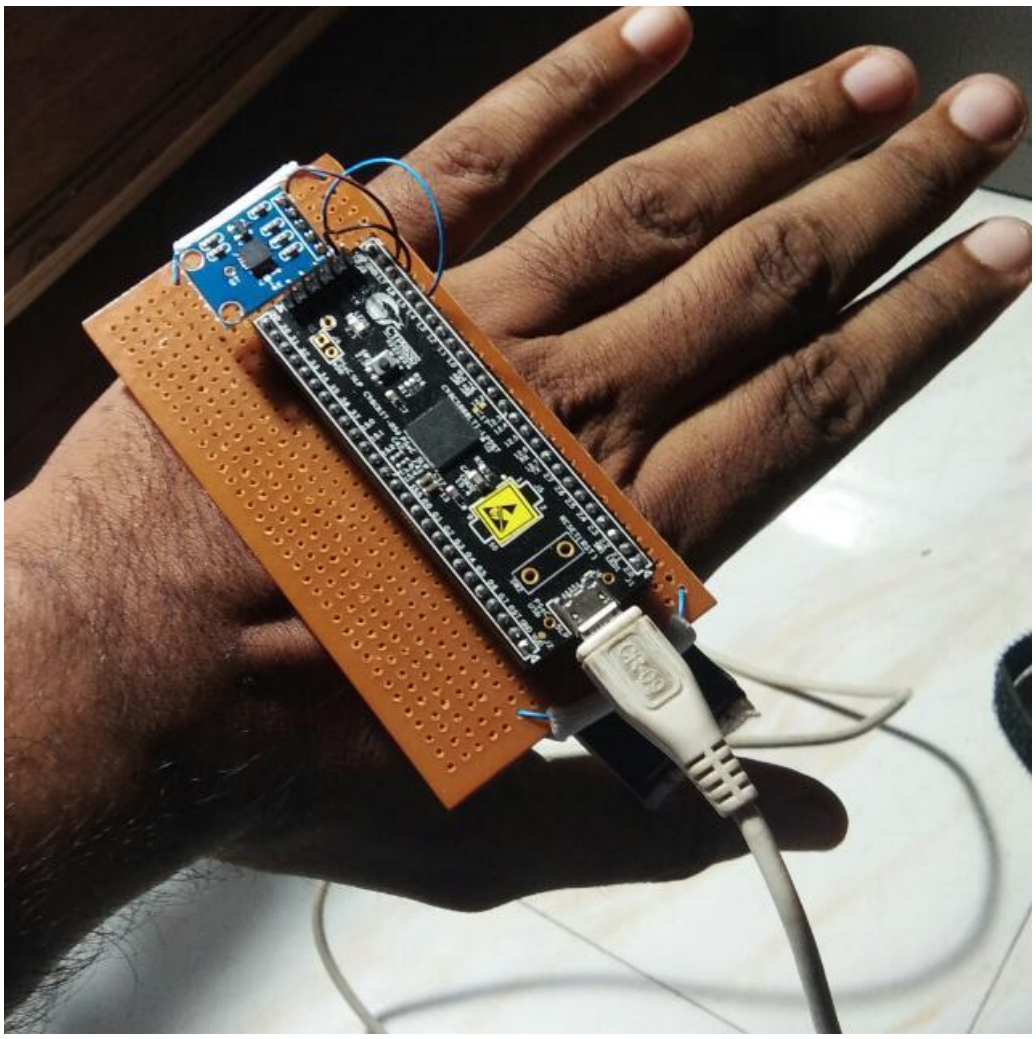


Fig 8 : Wearable hand tremor system

The ADXL -335 accelerometer sensor placed on back of the hand which measures acceleration in all three directions (x,y,z). The X-output ,Y-output and Z –output which are given to the PSoC chip programmed with the PSoc code. PSoC chip starts the Sigma Delta ADC conversion after receiving the data .Once the conversion is done USBUART transmits the data to the GUI based Matlab interface ,data acquisition is completed after which analysis is done .The discrete fourier transform of the data is obtained and plotted.

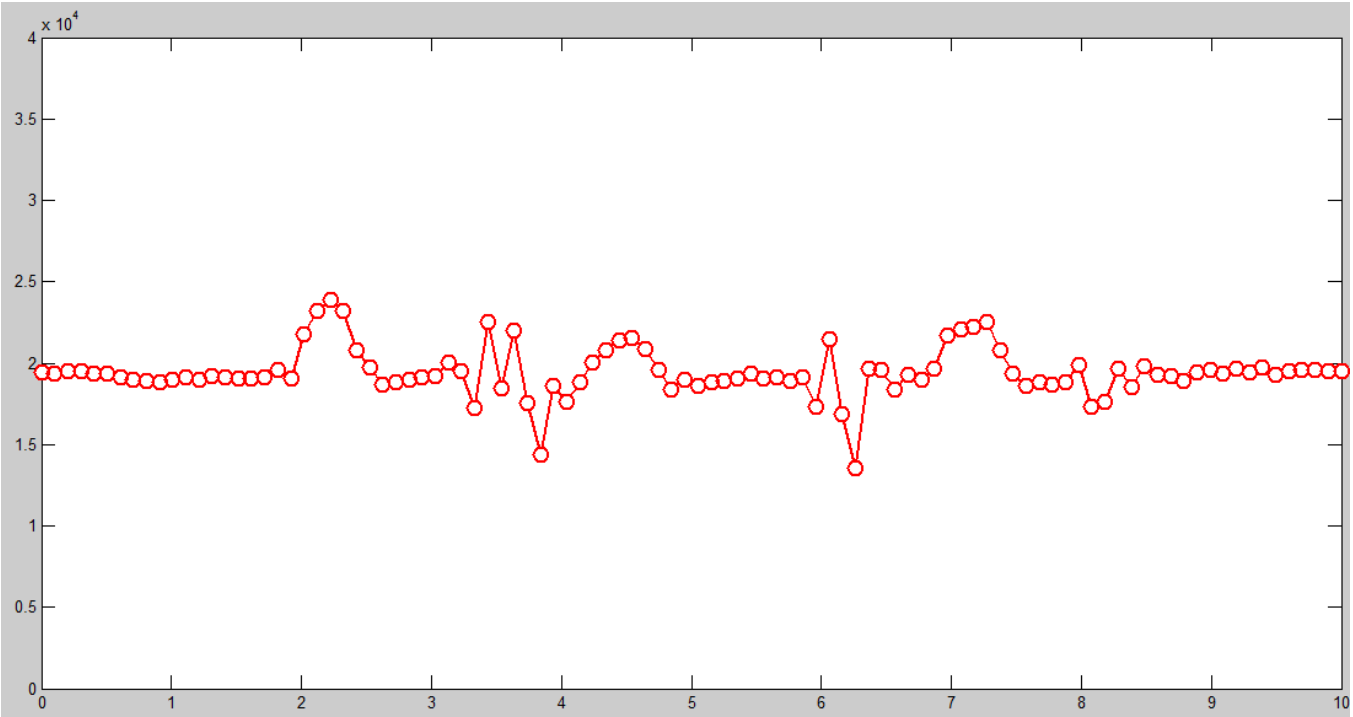


Fig 9 : output of hand tremor after data acquisition and analysis

The sampling frequency or sampling rate, fs, is the average number of samples obtained in one second (samples per second), thus fs = 1/T. Sampling a signal. To sample a signal in MATLAB, generate a time vector at the appropriate rate, and use this to generate the signal.

FAST FOURIER TRANSFORM(FFT)

Fast Fourier transform is a mathematical method for transforming a function of time into a function of frequency. It is described as transforming from the time domain to the frequency domain.

The Fast Fourier transform (FFT) is a development of the Discrete Fourier transform (DFT) which removes duplicated terms in the mathematical algorithm to reduce the number of mathematical operations performed. In this way, it is possible to use large numbers of samples without compromising the speed of the transformation. The FFT reduces computation by a factor of N/(log2(N)).

FFT computes the DFT and produces exactly the same result as evaluating the DFT; the most important difference is that an FFT is much faster!

Let x 0, ..., xN-1 be complex numbers. We have already seen that DFT is defined by the formula:

$$X_k = \sum_{n=0}^{N-1} x_n e^{-i 2\pi k \frac{n}{N}} \qquad k = 0, \ldots, N-1.$$

Evaluating this definition directly requires N2 operations: there are N outputs of Xk, and each output requires a sum of N terms. An FFT is any method to compute the same results in N log(N) operations. All known FFT algorithms require N log(N) operations.

To illustrate the savings of an FFT, consider the count of complex multiplications and additions. Evaluating the DFT's sums directly involves N2 complex multiplications and N(Nâˆ’1) complex additions. FFT algorithm can compute the same result with only (N/2)log2(N) complex multiplications and Nlog2(N) complex additions.

6.1 SPECIFICATIONS OF THE MODEL

SAMPLING RATE	20HZ
POWER CONSUMPTION	5V@50mA
SIZE AND VOLUME	5 cubic cm
WEIGHT	59GRAMS

6.2 LIMITATIONS

Despite the huge progress in the fabrication of soft and flexible sensors in the past decade, full system-level implementation is still a challenge. Realization of an array of actuators capable of providing localized haptic feedback is still in its infancy and issues such as crosstalk and power management are some of the technological hurdles to overcome

6.3 FUTURE SCOPE

Further, with the advances in the technologies for fabrication of biomimetic e-skins, we also envision the realization of smart skin-like gloves with all the human-like sensing capabilities, enabling users to use the gloves to explore surfaces and obtain richer information (e.g., softness, temperature, roughness) about the contact feature. Considering the fast-growing IoT, where sensors serve as data sources, enabling the harvesting of information from users, actuators will help to improve the quality of feedback that users receive from IoT devices. Furthermore, with the rapidly growing interest and research on tactile Internet, the potential of seamlessly integrating sensors and actuators cannot be overemphasized. In general, we envision e-skins with integrated sensing and actuation as an innovative opportunity for advancing HMIs by providing them with the capabilities for richer user interaction experience.

6.4 CONCLUSION

This work was designed to support clinical analysis for hand tremors caused due to neural disorders in the central nervous system .Standard procedures like EMG, DaT-Scan are expensive ,time consuming which leads to design of the wearable system for analysis of hand tremors. This system makes easy and cost efficient for clinical support. Data acquisition, analysis is done .

PUBLICATION DETAILS.

[1] G.Prassana, Akshaya Killi, Gollapalli Priscilla, Takasi Kiranmayee, Udigala Meghana, Wearable Hand Tremor Analysis System for Clinical Applications, International Conference On Advances In Steel, Power And Construction Technology.

REFERENCES

[1] Louis, E.D., 2001. Essential tremor. *New England Journal of Medicine*, *345*(12), pp.887-891.

[2] Davie, C.A., 2008. A review of Parkinson's disease. *British medical bulletin*, *86*(1), pp.109-127.

[3] Niazmand, K., Tonn, K., Kalaras, A., Fietzek, U.M., Mehrkens, J.H. and Lueth, T.C., 2011, June. Quantitative evaluation of Parkinson's disease using sensor based smart glove. In *2011 24th International Symposium on Computer-Based Medical Systems (CBMS)* (pp. 1-8). IEEE. Hagan, M. and Geman, O., 2016, November. A wearable system for tremor monitoring and analysis. In *Proc. Rom. Acad. Series A* (Vol. 17, No. 1, pp. 90-98).

[4] Rovini, E., Esposito, D., Maremmani, C., Bongioanni, P. and Cavallo, F., 2014, September. Using wearable sensor systems for objective assessment of Parkinson's disease. In *20th IMEKO TC4 international symposium and 18th international workshop on ADC modelling and testing* (pp. 862-867).

[5] Ma, C., Li, D., Pan, L., Li, X., Yin, C., Li, A., Zhang, Z. and Zong, R., 2022. Quantitative assessment of essential tremor based on machine learning methods using wearable device. *Biomedical Signal Processing and Control*, *71*, p.103244.

[6] Vescio, B., Quattrone, A., Nisticò, R., Crasà, M. and Quattrone, A., 2021. Wearable devices for assessment of tremor. *Frontiers in Neurology*, *12*.

[7] Powell, H.C., Hanson, M.A. and Lach, J., 2007, November. A wearable inertial sensing technology for clinical assessment of tremor. In *2007 IEEE Biomedical Circuits and Systems Conference* (pp. 9-12). IEEE.

[8] Iosa, M., Picerno, P., Paolucci, S. and Morone, G., 2016. Wearable inertial sensors for human movement analysis. *Expert review of medical devices*, *13*(7), pp.641-659.

[9] Locatelli, P. and Alimonti, D., 2017, June. Differentiating essential tremor and Parkinson's disease using a wearable sensor—A pilot study. In *2017 7th IEEE International Workshop on Advances in Sensors and Interfaces (IWASI)* (pp. 213-218). IEEE.

[10] Zhou, Y., Ibrahim, A., Hardy, K.G., Jenkins, M.E., Naish, M.D. and Trejos, A.L., 2021. Design and Preliminary Performance Assessment of a Wearable Tremor Suppression Glove. *IEEE Transactions on Biomedical Engineering*, *68*(9), pp.2846-2857.

[11] Zhang, J., Chu, F. and Mohammed, N., 2005, June. DSP controller based signal processing of physiological hand tremor. In *Proceedings of the 2005, American Control Conference, 2005.* (pp. 1569-1574). IEEE.

[12] Elble, R.J. and McNames, J., 2016. Using portable transducers to measure tremor severity. *Tremor and Other Hyperkinetic Movements*, *6*.

[13] Zhang, J. and Chu, F., 2005, March. Real-time modeling and prediction of physiological hand tremor. In *Proceedings.(ICASSP'05). IEEE International Conference on Acoustics, Speech, and Signal Processing, 2005.* (Vol. 5, pp. v-645). IEEE.

[14] Veluvolu, K.C. and Ang, W.T., 2011. Estimation of physiological tremor from accelerometers for real-time applications. *Sensors*, *11*(3), pp.3020-3036.

[15] Bain, P.G., 1998. Clinical measurement of tremor. *Movement disorders*, *13*(S3), pp.77-80.

[16] Kim, H.B., Lee, W.W., Kim, A., Lee, H.J., Park, H.Y., Jeon, H.S., Kim, S.K., Jeon, B. and Park, K.S., 2018. Wrist sensor-based tremor severity quantification in Parkinson's disease using convolutional neural network. *Computers in biology and medicine*, *95*, pp.140-146.
[17] Haubenberger, D., Abbruzzese, G., Bain, P.G., Bajaj, N., Benito-León, J., Bhatia, K.P., Deuschl, G., Forjaz, M.J., Hallett, M., Louis, E.D. and Lyons, K.E., 2016. Transducer-based evaluation of tremor. *Movement Disorders*, *31*(9), pp.1327-1336.