

**Designing a Wireless 8-channel EEG (WEEG) recording device for
Brain-Computer Interface Application**

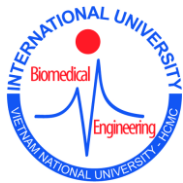
By

Vo Tri Thong

A thesis submitted to the Biomedical Engineering Department in partial
fulfillment of the requirements for the degree of Master of Engineering

Ho Chi Minh city, Vietnam

September – 2016



Vietnam National Universities – HCMC

International University

Biomedical Engineering Department

**Designing a Wireless 8-channel EEG (WEEG) recording device for Brain-
Computer Interface Application**

APPROVED BY:

ADVISOR

Prof. Võ Văn Tới, PhD.

THESIS COMMITTEE

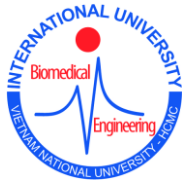
Lê Quốc Trung, PhD., Chair

Lê Chí Thông, PhD., Reviewer 1

Lê Thanh Hải, PhD, Reviewer 2

Prof. Đặng Thành Tín, PhD., Member

Phạm Thị Thu Hiền, PhD., Secretary



Acknowledgments

After a prolonged period of three years, I am finally writing this thanks note as a finishing touch for my Master thesis. It has been a challenging time for me to learn about academia, do research on frontier topics, and explore biomedical concepts. Besides gaining scientific knowledge, I also grow up personally by going through struggles upon completing this thesis. Without the guidance, support from generous individuals around me, I would not have success in this mission. I would like to express my appreciations to all of my benefactors throughout this course.

I would like first to thank my thesis advisor, Professor Vo Van Toi, Ph.D. He influents me to discover novel ideas and steer me through the Master program. His office door is always open whenever I need his support. The positive thinking and courageous of Prof. Toi toward challenges empowers me to confront issues confidently. Also, he let me explore original ideas which he does not endorse. I learn a lot from him not only as an academic advisor but also as a kind person. Likewise, I express my greatest gratitude to Mr. Nguyen Phuong Nam, MSc, lab manager, who is my mentor throughout the course. His practical experience and excellent knowledge in biomedical engineering are a treasure for me. He prefers letting me battle with issues for a while rather than giving me the solution without delay. This approach nurtures my problem-solving skills which are critical to my success as an engineer.

Other people who contribute to my success are notably mentioned below. Special thanks to Ms. Nguyen Thanh Chau who worked closely with me to complete the first prototype; Mr. Do Minh Thai, M.Sc, lab technician; Mr. Nguyen Quang, lab partner; Mr. Tran Ngoc Viet, research assistant and Mr. Le Y, experiment subject. Also, to Mr. Le Quoc Trung, Ph.D. who share his understanding and experience of data analysis. Last but not least, I would like to thanks my wonderful parents for their sympathize ears and pressure for me to complete this thesis on time. They assure that I always have a soothing place to come back after failures.

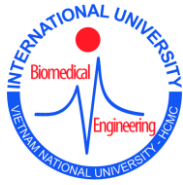
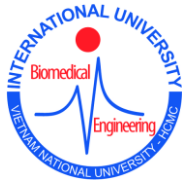


Table of Contents

Contents

1. Chapter 1.....	13
Introduction.....	13
1.1. Overview	13
1.2. Objectives and scope of project	14
1.3. Thesis outline	14
2. Chapter 2.....	16
Literature Review.....	16
2.1. Other mobile EEG recording systems:.....	16
2.2. Overall system architecture	18
3. Chapter 3.....	23
Methodology	23
3.1. Hardware development	23
3.1.1. Major hardware components.....	27
3.1.2. Hardware design	41
3.1.3. Communication.....	45
3.2. Firmware development.....	49
3.2.1. Current design	49
3.2.2. Data Format	51
3.3. Software development.....	54



Vietnam National Universities – HCMC

International University

Biomedical Engineering Department

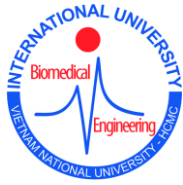
3.3.1.	Major software block	54
3.3.2.	Current design	55
3.3.3.	Filtering	56
3.3.4.	Ring Buffer	57
3.3.5.	FFT	59
3.3.6.	Spectrogram	60
4.	Chapter 4.....	63
	Evaluation and results	63
4.1.	Alpha wave.....	63
4.2.	Steady State Evoke Potential experiment.....	64
4.3.	Compare with a traditional EEG system, Biosemi Active two.	67
4.3.1.	General properties of an EEG system	67
4.3.2.	Compare Power Signal Densities (PSD) at alpha band	69
	The normalized cross-correlation between two signal is very high at 0.98. The calculation for this value is described as below	72
4.3.3.	Compare SSVEP PSD.....	73
5.	Chapter 5.....	75
	Conclusions and Future Work	75
5.1.	Conclusion.....	75
5.2.	Future work	75



Vietnam National Universities – HCMC

International University

Biomedical Engineering Department



List of Figures

Figure 1: Emotiv EPOC[6]	17
Figure 2: OPENBCI board[11]	17
Figure 3: Simplified overall system architecture	18
Figure 4: BCI system architecture	22
Figure 5: DIN Touch-proof Jack and PCB mount header [12].....	24
Figure 6: Prototype #3 with 0.1' header and Gold Cup electrodes	25
Figure 7: STM32F407 in LQFP100 package[14]	27
Figure 8: STM32F407 Block Diagram[15]	28
Figure 9: ADS1299 Block Diagram[13].....	30
Figure 10: ADS1299 Input multiplexer block for one channel[13].....	31
Figure 11: TPS63060 Simplified schematic	32
Figure 12 MCP73832 circuit diagram	32
Figure 13: TPS63060 functional block diagram [16]	33
Figure 14: TPS72325 Functional Block Diagram [17].....	33
Figure 15: ADS1299EEG-FE Kit	34
Figure 16: ADS1299 EEG-FE Daughter Card Block Diagram[18]	35
Figure 17: Low-pass filter for input channel	36
Figure 18: Mid Supply and Buffer Circuit.....	37
Figure 19: Channel Control Registers GUI Panel in the Software	38
Figure 20: Scope Tool Feature of the Software	38
Figure 21: ADS1299 Device ID Control Register from the Datasheet [13].....	40
Figure 22: Main power management block	41
Figure 23: Sub power management block	42
Figure 24: ADS1299 Input schematic.....	43
Figure 25: Sub Analog system.....	44

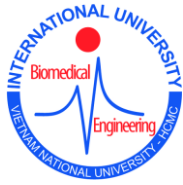


Figure 26: Digital system schematic (Prototype #3).....	45
Figure 27: Data Rate vs. Power Consumption [22]	47
Figure 28: HC-05 Bluetooth Module[23]	48
Figure 29: Initiate SPI parameters	49
Figure 30: SPI Timing characteristic of ADS1299	49
Figure 31: Firmware process flow	51
Figure 32: Ideal Output Code versus Input Signal.....	52
Figure 33: Communication Protocol and Data Type	53
Figure 34: Brainbay screenshot	54
Figure 35: Dataflow diagram	55
Figure 36: User Interface of Filter feature in the GUI	56
Figure 37: Demonstration the effect of filters.....	57
Figure 38: Structure of a Ring buffer.....	58
Figure 39: A sample of FFT plot during an SSVEP experiment	59
Figure 40: Padded vs. Unpadded FFT to find a peak frequency at 11 Hz	60
Figure 41: Sample of a spectrogram of an SSVEP Experiment	62
Figure 42: Sample of spectrogram to detect alpha waves.....	62
Figure 43: Protocol for detecting alpha waves	64
Figure 44: Spectrogram of alpha waves detection session	64
Figure 45: Illustration of a SSVEP test session	65
Figure 46: Spectrogram of an SSVEP session.....	65
Figure 47: Amplitude Spectrum during stimulation	66
Figure 48: Pictures of the five EEG acquisition systems[27]	67
Figure 49: Electrode placement for comparison with Biosemi's ActiveTwo	70
Figure 50: Alpha waves comparison protocol	70
Figure 51 Sample of alpha waves in time domain from both systems (left: Biosemi ActiveTwo; right WEEG)	71

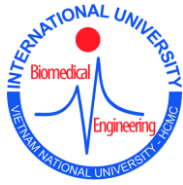


Vietnam National Universities – HCMC

International University

Biomedical Engineering Department

Figure 52: A comparison of power spectral densities (PSDs) for WEEG and Biosemi recordings for the “eyes closed.”	72
Figure 53: Coherence Estimation via Welch’s method.	74



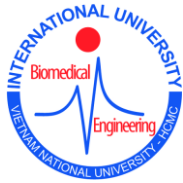
Vietnam National Universities – HCMC

International University

Biomedical Engineering Department

List of Tables

Table 1: Target project specification	23
Table 2: Hardware and Software Evolution.....	25
Table 3: Analog Supply Configurations[18].....	36
Table 4: Clock Jumper Options[18].....	36
Table 5: General specs comparision between systems [27].....	68
Table 6: General specs comparison between systems (continue)	69



Vietnam National Universities – HCMC

International University

Biomedical Engineering Department

List of Abbreviations and Symbols

ADC	Analog Digital Converter
BCI	Brain Computer Interface
BME	Biomedical Engineering
DFT	Discrete Fourier Transform
EEG	Electroencephalogram
FFT	Fast Fourier Transform
FIR	Finite Impulse Response
GUI	Graphical User Interface
HCI	Human Computer Interaction
IC	Integrated circuit
IIR	Infinite Impulse Response
LCD	Liquid Crystal Display
MCU	Microcontroller
PGA	Programmable Amplifier
PSD	Power Spectral Density
RF	Radio Frequency
SNR	Signal-to-Noise Ratio
SoC	System On Chip
SPS	Sample per second
SSVEP	Steady State Evoked Potential
TI	Texas Instruments
UART	Universal Asynchronous Receiver/Transmitter
WEEG	Wearable 8-channel EEG – The prototype of this project.



Vietnam National Universities – HCMC

International University

Biomedical Engineering Department

Abstract

Research-grade Electroencephalogram (EEG) instruments offer the brain's electrical activity recording with high accuracy and a wide range of functionalities. Due to rigorous component quality requirements, these devices are usually expensive and elaborate. Recent advancements in electronic enable mobile designs to achieve comparable measurement results with similar functions. This thesis presents the development and evaluation of WEEG – A Wearable 8-channel EEG recording device with integrated hardware and software interface. The wireless feature is a major improvement compared to a traditional system. The circuit board which has the size of a half of a credit card contains a power management system, an ADS1299 Analog to Digital Converter (ADC) to capture brainwaves and a microcontroller to handle partially digital processing tasks. It communicates with a host computer via Bluetooth/Serial. Extensive evaluations were performed to assess the performance of WEEG in practice. We compared the Signal quality of WEEG to that of the Biosemi Active Two - a conventional high-quality EEG recording system. The results came out comparable, indicating a high accurate recording of the proposed system. Also, Steady State Visually Evoked Potential (SSVEP) Brain-Computer Interface (BCI) game was devised. The high performance of users in the game suggests usage of our device in a wider range of Brain Computer Interface applications.



1. Chapter 1

Introduction

1.1. Overview

An EEG recording device gauges electrical activities at the scalp surface. The majority of these signals are produced by the summation of potentials from a large number of neurons within the brain. Brainwaves varies from $0.5\mu\text{V}$ to $100\mu\text{V}$ in amplitude. In the frequency domain, EEG signals have been categorized into four basic groups. Delta waves have low frequencies which are less than 4Hz. Delta waves dominate during dreamless sleep. Adjacent to Delta wave domain, Theta waves are in 4Hz-8Hz region and Alpha waves have frequencies from 8 to 12Hz. They dominate when eyes are closed, and the brain is in a relaxed state. The rest called Beta waves are signals with frequencies greater than 13Hz. Beta waves associated with alertness and active attention [1]. By identifying the frequency of current dominant waves, researchers were able to recognize the current arousal state [2].

EEG has a broad range of applications from clinic to game entertainment. Some critical clinical applications are monitoring alertness, coma and brain death; locating areas of damage following head injury, stroke; investigating epilepsy and locate seizure. Additionally, other interesting applications are evoked potentials and brain-computer interface (BCI). Evoked potentials are used in marketing research and detecting lies. Non-invasive brain activities inputs enhance human-computer interactions (HCI) with an effortless requirement from users. Brain states such as stress level, attention level, emotion, the event-related response could be extracted from EEG to augment HCI. Moreover, real-time EEG-enabled systems could be developed for medical applications, performance improvement, entertainment, or even neuromarketing[3]. Alongside those promising functions, costs for traditional EEG related system are quite high. A popular research-graded EEG recording device, i.e. Biosemi ActiveTwo, costs more than \$13,000 USD. The price impedes development and wide adoption of EEG related application.



Therefore, the motivation is to build a low-cost EEG recording device to substitute those expensive devices wherever possible. Even though the novel system is not to the bar of the traditional system, it offers satisfactory performance in some applications. Also, the WEEG is a wireless device which is a significant advantage compared to the traditional wired system.

1.2. Objectives and scope of project

Goals of this project are to produce a low-cost EEG recording device, the cost is less than \$200, which wirelessly transfer data to a host computer for signal processing. A Graphical user interface to visualize outputs and processed data are also presented. This system is capable of performing typical BCI applications such as alpha wave BCI, SSVEP based BCI game. Getting a comparable measuring result from the proposed system with a traditional ‘gold standard’ system is an audacious goal. Such following intentions are outside of the scope: to create an all-in-one physiology measurement device which is capable of capturing EMG or ECG, even though this device can measure these signals. Also, it is not an aim to create a medical device. Extensive evaluations and certification processes need to be done to create a medical device. These opportunities are open for next stage of development.

1.3. Thesis outline

Chapter 1 – This chapter goes through background information about EEG, its current application, the scope of this project and the thesis outline.

Chapter 2 – This section describes current state of mobile EEG platform development by reviewing other literature. Moreover, an overview of such system is also discussed.

Chapter 3 – The developments for every aspect of WEEG are presented in this chapter. Highlights are software development, hardware, and firmware design. This section also discloses hard decisions during the development process.



Vietnam National Universities – HCMC

International University

Biomedical Engineering Department

Chapter 4 – Various system evaluations on difference paradigms are in this part. The device performs well during a tested with a BCI game. Results from the comparison with Biosemi Activetwo are very promising.

Chapter 5 – Project conclusion and future works are presented in this chapter.



2. Chapter 2

Literature Review

2.1. Other mobile EEG recording systems:

Advancements in electronic have enabled smaller, better and lower cost EEG recording device in recent years. Some wearable EEG recording prototypes were successfully developed such as a wireless non-contact cardiac and neural monitoring system[3], a helmet to monitor physiological signal[4], a portable device for real-time drowsiness detection[5]. Few brain wave recording devices can commercialize as consumer products; moreover, they are adopted by researchers. One prominent example is the Emotiv EPOC® EEG gaming device which costs around \$800[6]. It is extensively evaluation by researchers in different paradigm such as auditory event-related potentials [7] or assistive technologies using facial expressions [8]. Other notable novel EEG systems are the ABM's B-Alert X10 and QUASAR's HMS.

The release of TI's ADS1299 chip to quantify brain waves spark interests to build EEG recording system in researcher's community. The small SoC packed with various EEG amplifiers and function blocks. Majority prototypes are built and verify with BCI application [9, 10]. However, authors of these systems do not publish any objective performance indicator of BCI paradigm. Also, the OPENBCI launched a massively successful campaign, collected \$168,829, to build a biosensing system based on the ADS1299. The team also provides open source headset for electrode placement.



Vietnam National Universities – HCMC

International University

Biomedical Engineering Department



Figure 1: Emotiv EPOC[6]

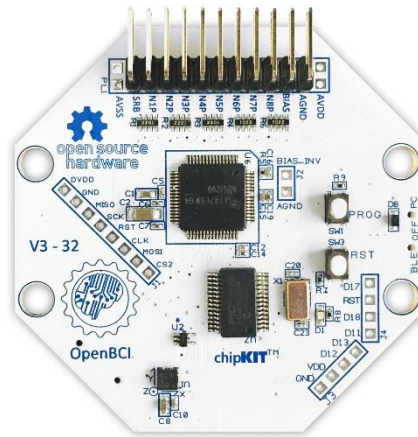


Figure 2: OPENBCI board[11]

All of the above systems are remarkable efforts to build EEG recording system. Major concerns for those systems are limited evaluations to verify the performances. As author's observation, existing applications for these platforms are limited to alpha waves BCI which are a binary output experiment. The systems determine whether a subject is closing their eyes or not. Also, the sampling rate is limited to 250 SPS which may not be enough for some event-related potential experiments. Also, they have not benchmark these platforms with traditional “gold standard” EEG device to weigh theirs' system performance.

2.2. Overall system architecture

A typical EEG recording system includes essential functional modules such as a signal acquisition block, a signal processing block, a power management system and an interface for users to interact with. Low noise, high common mode rejection ratio (CMRR) analog to digital converter (ADC) is a crucial component of the acquisition block. The Digital System, microcontroller, facilitate the communication between the Analog system and the Signal processing module. In addition, the signal processing block has robust algorithms to filter out noises, interpret incoming data and save them for offline analysis. The graphical user interface software doesn't face many technical challenges as the two previous modules, but it requires careful design to support smooth interactions with the hardware. The Power module convert a constant voltage from a battery to various voltage level for the whole system. In depths, details are discussed more in section 3.1.2.

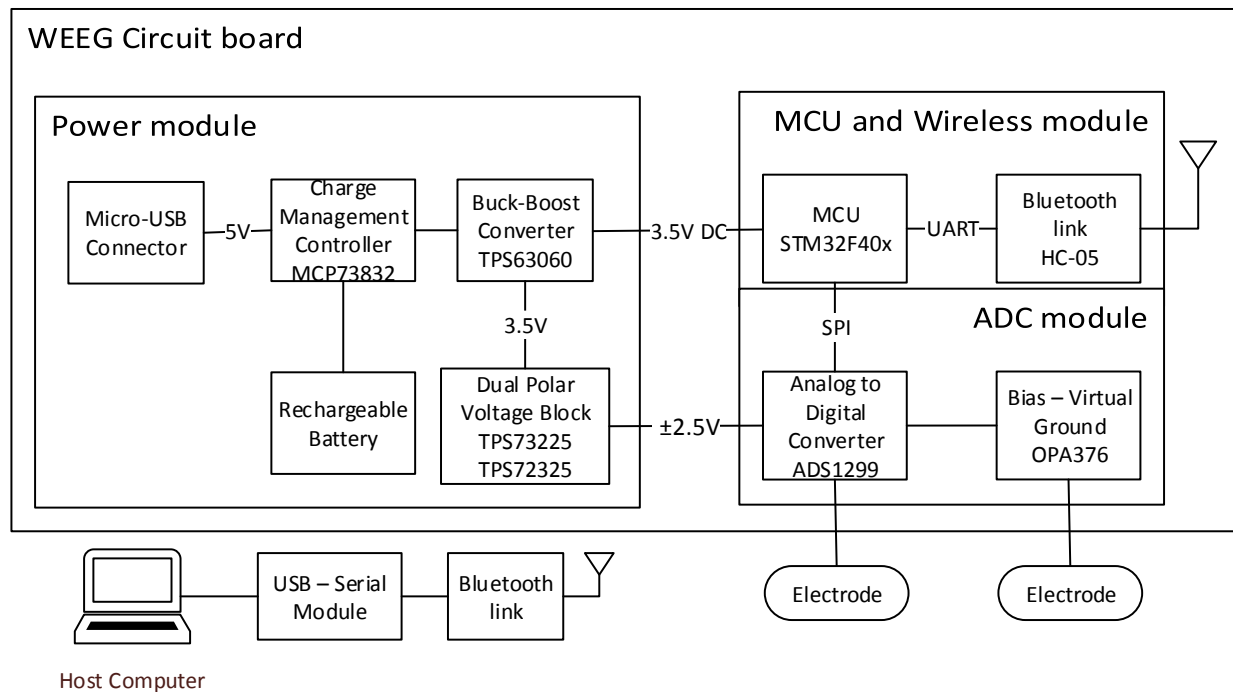


Figure 3: Simplified overall system architecture



All circuit components are packed into a circuit board which is smaller than a regular smartphone size. Users can comfortably carry the device in a pouch holster all day long for monitoring activities if required. It consists an ARM Cortex M4 STM32F4 as the main microcontroller and the ADS1299 for Analog to Digital Converter (ADC) tasks. A 3.7V/1000mAh lithium polymer battery powers the whole system which draws approximately 30mA at steady state. This battery will last for 24 hours after fully charged. A power management system is also included to charge the battery. Constant system voltage is maintained regardless input voltage from the battery by a buck/boost converter. The device communicates with a computer or a smartphone via Bluetooth or Serial interface. Real-time graphs of all channels are presented on the PC.

This system uses standard gold cup wet electrodes to acquire data. It has a distinct advantage in term of signal quality compares to dry electrode. This electrode terminal was made by a 0.1' female header compare to a standard DIN Touch Proof Connector to save space on the Printed Circuit Board (PCB).

The device takes continuous eight channels 24bit ADC EEG signal from non-invasive electrodes. The gain at the Programmable Gain Amplifier (PGA) is 24. It is the maximum PGA of the ADS1299. The ADC module used a bipolar power supply at -2.5V and +2.5V. This bias electrode (or right leg electrode) is the virtual ground of these bipolar supplies. The second option for this bias electrode is a signal from the internal bias amplifier. This bias op-amp sums input data from all channel and compares the sum with a reference voltage to generate the bias signal. The sample sampling rate is 500 sample/second. Even though the ADS1299 module is capable of sampling up to 16k samples/second, 500/s is sufficient for BCI application whereas brainwave frequency is below 50 Hz. It is well below the Nyquist frequency 250Hz. On the other hand, the data throughput of the system is limited by the serial port protocol. The reliability of data transfer drops when the baud rate is increased. Digital signals are sent from the ADS module to the microcontroller for preprocessing via SPI interface. Whenever a new data package is ready to transfer, the ADS1299 module will signal the microcontroller via the Data ready pin. Multiple ADS1299 devices may share one SPI line. They are synchronized by the START pin of every instrument.



Two electrode input options are provided in the current design. The first option is the single ended input which is the typical design of EEG recording device. All negative pins of eight channels are wired internally to a single pin SRB1 of the ADS1299. Therefore, these configurations are able changed by firmware commands. The second option is differential input. Every channel has independent positive and negative input. This configuration provides better noise reduction effect in BCI application based on our experience.

According to Figure 4, the peripheral controller of the STM32F4 takes recording data from the ADS via SPI interface. The data package consists of 3 status byte information and 24 EEG data bytes (details in Figure 33). The microcontroller converts these two complement values to decimal and repackages them. The counter is added to these packages as a rudimentary way to monitor data integrity. A checksum or other techniques may be implemented to improve this process.

After processing the data, the STM32F407 chip sends data in packages to the computer via serial/Bluetooth for real-time display and data recording. Users may access these data via Bluetooth on a smartphone or an RS-232 serial port. WEEG system is performing at 230400 baud rate, details in 3.1.3.1, with a reliable output.. After an investigation to use Bluetooth Low Energy (BLE) for the design, it is known that this BLE protocol data transfer rate is slow, and the package size is limited to 20 bytes or less. It opposed to high data transfer rate of EEG recording devices whereas multiple channel information is recorded simultaneously. In the next stage, data will be processed within the STM32F407 chip on the instrument. It will compute FFT to support Steady State Evoked Potential (SSVEP) BCI. It will be possible to use BLE to send status data to a smartphone or computer.

The interface for this project is developed on Matlab. Even though Matlab may not be the best tool to handle large real-time data, we found it is powerful with much ready-to-use signal pro-cessing toolboxes. This is a key factor for EEG data analysis. Data from the STM32F4 is transferred to PC serial buffer periodically. MATLAB put these data chunks to a Ring Buffer. This is a key component in this design to bal-ance the transfer rate for serial data and the processing time of



Vietnam National Universities – HCMC

International University

Biomedical Engineering Department

previous packages. In our previous design, we faced intermittent package drop because PC's serial buffer was full and MATLAB hasn't finished the current process to take more data. Increasing the buffer size alleviated the problem; however, a large buffer introduced a long delay between signal acquisition and presentation.

Whenever a new set of packages is available in the ring buffer, the program extracts them to channel level data and plot it on the interface. A rolling plot presents last 1000 sample (4 seconds) values. These values are filtered by a DC blocker to eliminate drift current and a low-pass filter at 45 Hz to eliminate electric hump and indifferent signal. Real-time FFT is also present for SSVEP BCI tasks. The program also checks the counter of every incoming package to monitor the data integrity. Data of every channel are saved for offline analysis after each recording session.

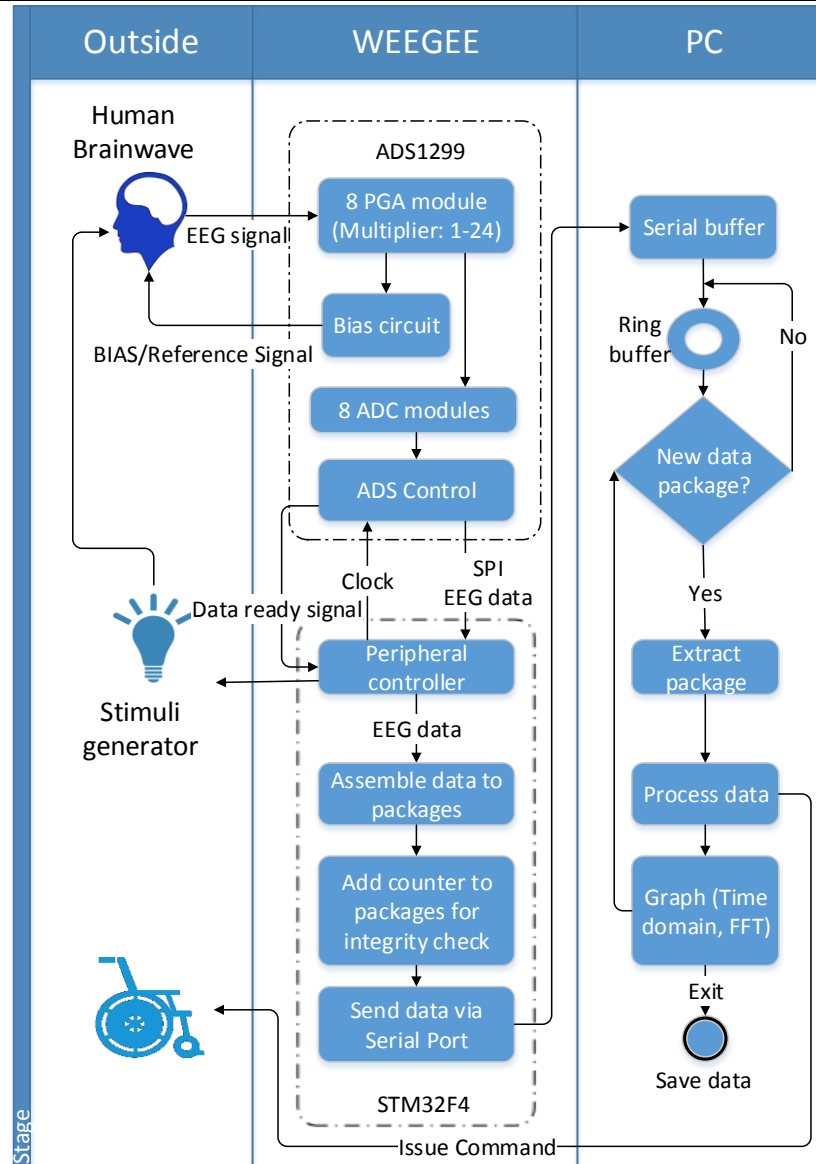


Figure 4: BCI system architecture



3. Chapter 3

Methodology

3.1. Hardware development

Table 1: Target project specification

ADC Precision	24-bit (ADS1299)
Power	Battery (24h)
EEG channel	8
Sampling rate	250SPS
Bandwidth	1-40Hz
Communication	Bluetooth
MCU	ARM-Cortex M4 (best performance)
Size	< 10cm x 10cm

Even though the end goal of this project is an integrated EEG recording module on a small circuit board, the team start off by using a **TI evaluation module**. It offers a quick way to assess the performance of ADS1299 without spending much time to design, layout and solder a circuit board. TI also provides a comprehensive software on the PC to configure the device. One big drawback of this module is that it does not provide real-time data for processing. This problem motivated the author to build a Prototype #1 quickly to interface the ADS1299 directly without using TI proprietary hardware and software. This approach provides the author full control of every data in and out of the system. The Prototype #1 consists of an Analog Front End ADS1299 daughter card, an STM32F4 Discovery board, and an electrode adapter. A complement MATLAB software was built to process data from Prototype #1. It can display only one channel at a time while data packages constantly drop during the transferring process due to system overloading.

With successful primary results from the 1st design, **Prototype #2** was built by putting every essential component for Prototype #1 on the same circuit board to achieve an integrated system. The digital system of this version works well as expected. It connects with the ADS1299 to get



Vietnam National Universities – HCMC

International University

Biomedical Engineering Department

data and transfer those data to the computer. It is also able to program or reading the value from the ADS1299. However, the analog system does not work well at the first. The problem surfaces out after extensive debugging. A polarized capacitor is soldered in wrong polar. This issue led to a decision to use all non-polar capacitor in the next prototype. Along these lines, the software in version two was improved with a better data stream buffer, a ring buffer. This upgrade boosted the performance significantly. It enables multiple channel processing in contrast with single channel handling in the previous version.

Moving on to **Prototype #3**, the circuit board is compressed to a credit card size board while the overall schematic was maintained. A battery power management system was added. The DIN touch proof connection terminal was swapped for a 0.1" 2x8-pin strip dual male header because the predecessor connectors took too much circuit board footprint. The new Gold Cup electrode was purchased from OPENBCI store. On the other hand, both digital system and analog system work well as designed. Extensive tests were deployed to verify the performance of the whole system. In details, variety signals were tested such as large DC signal, small DC signal, small AC signal, and the various signal from the function generator. On the software side, the signal processing block was vastly improved by adding a real-time filter and real-time FFT. These lasted features enable a Brain-computer interface system which is discussed on **Error! Reference source not found..** In summary, this version presents a full features EEG recording system with a BCI application. Finally, **Prototype #4** which is half a credit card size is the most compact design of WEEG. It includes a new USB power port for convenient power delivery. This version is under electrical validation.



Figure 5: DIN Touch-proof Jack and PCB mount header [12]



Vietnam National Universities – HCMC

International University

Biomedical Engineering Department

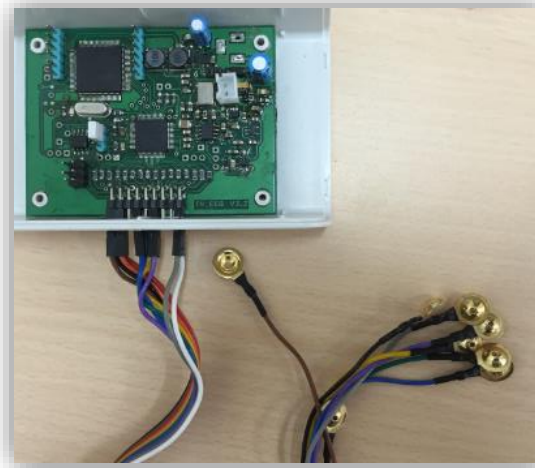
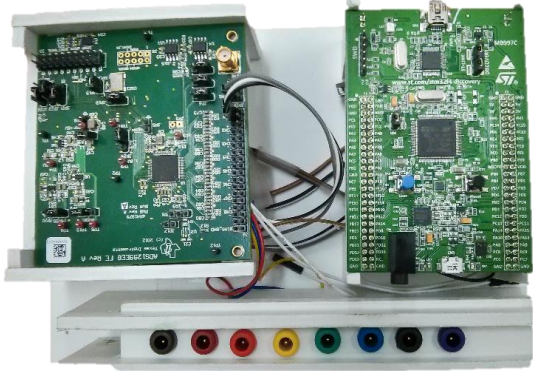
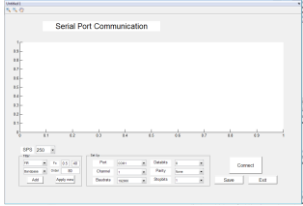





Figure 6: Prototype #3 with 0.1' header and Gold Cup electrodes

Table 2: Hardware and Software Evolution

Version	Hardware	Software
TI Evaluation Module	 <ul style="list-style-type: none"> • Motherboard MMB0 and Daughter Card ADS1299 [13] 	 <ul style="list-style-type: none"> • Software provided by TI

<p>Prototype 1</p>	 <ul style="list-style-type: none"> • Full control of the ADS1299 • DIN Touch Proof Electrode Adapter 	 <ul style="list-style-type: none"> • Rudimentary Matlab GUI to • Read serial value • Split Data Package to Channel value • Display 1 channel value
<p>Prototype 2</p>	 <ul style="list-style-type: none"> • One integrated circuit board • Digital communication with ADS1299 • Analog data acquisition fail 	<ul style="list-style-type: none"> • Software enhancement: • Improved data structure (ring buffer) – 0 package drop • Code optimization (reduce execution time) • Multiple channels plots
<p>Prototype 3</p>	 <ul style="list-style-type: none"> • Fully integrated battery power in credit card size • Successfully acquire digital and acquired analog data • Transmits data wirelessly at 500SPS 	<ul style="list-style-type: none"> • More feature • Real-time Filter • Real-time FFT • BCI game • Data Marker • Offline data



<p>Prototype 4</p>	 <ul style="list-style-type: none"> • Smaller board which reduces noise, better signal transmission • Improve power management block • Half the credit card size 	<ul style="list-style-type: none"> • N/A
---------------------------	--	---

3.1.1. Major hardware components

3.1.1.1. The STM32F4 Microprocessor and the Discovery kit

The STM32F407 is a high-performance ARM Cortex-M4 32-bit core and 1-Mbyte flash. It features a Floating Point Unit (FPU) which is suitable for demanding computing task such as FFT. These tasks are crucial to developing a standalone device without tethering to a computer in the future. For prototype 1 to 3, the 100 pin version is used. In prototype #4, the smaller 64 pin out package STM32F405 with similar functions is employs. Both of the STM32F407 and STM32F405 have extra I/O pins for this application which uses only ~20% I/O pin of the 100-pin chip. However, there is no such less I/O chip which packed same capability as the Cortex-M4. Also, these extra pins may be used for a future peripheral interface such as Liquid Crystal Display or SD Card.



Figure 7: STM32F407 in LQFP100 package[14]

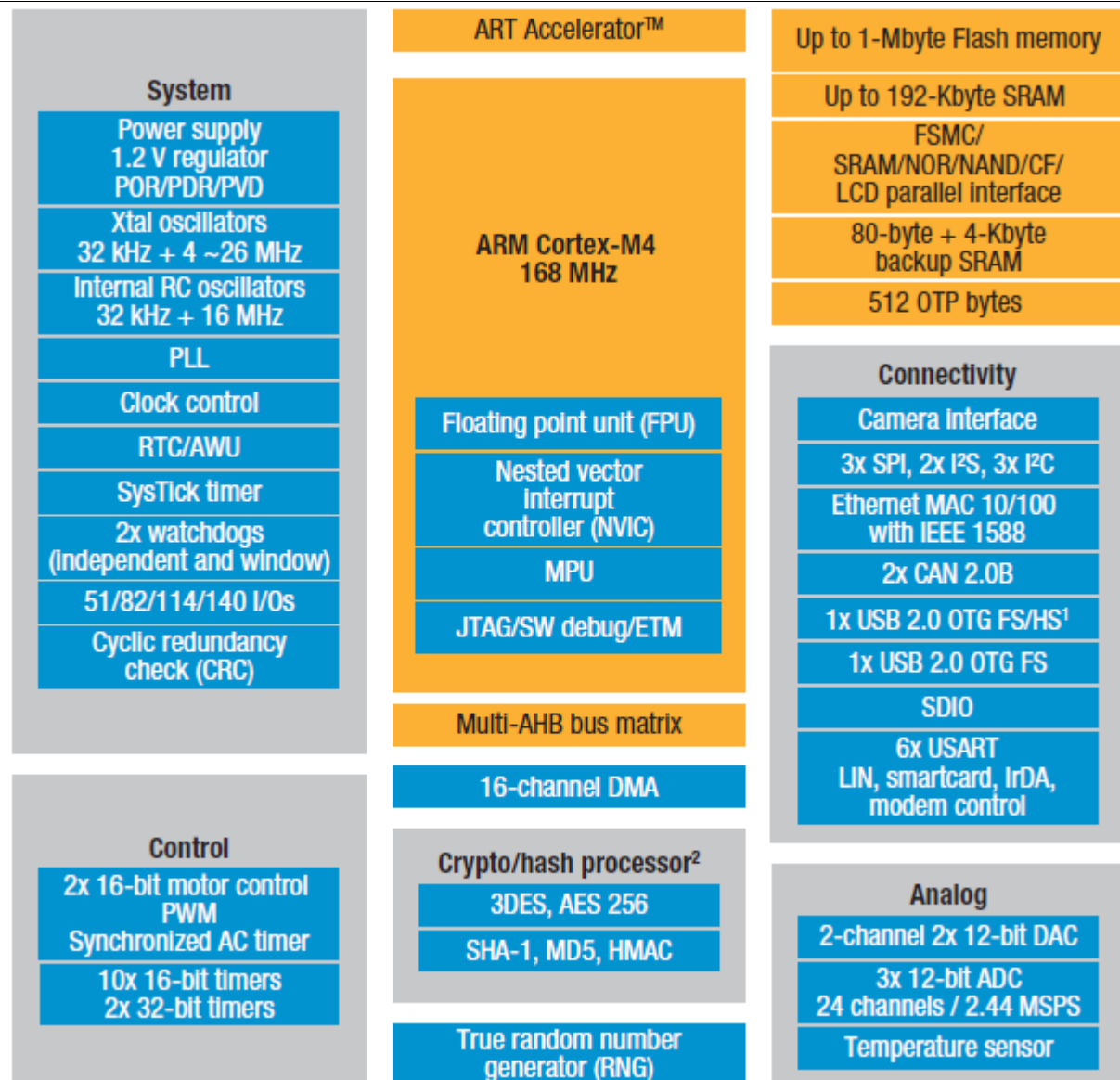


Figure 8: STM32F407 Block Diagram[15]

3.1.1.2. The ADS1299 Analog to Digital converter

The hearts of all hardware prototypes for this project are the superior TI ADC chip ADS1299. It is the most important module in the circuit board. Other components support this chip to work well as designed. The SoC is targeted for medical instrument application (EEG and ECG) with high-



Vietnam National Universities – HCMC

International University

Biomedical Engineering Department

precision (24-bit), simultaneous sampling and multichannel signal acquisition [13]. The data rate is configurable from 250 SPS to 16 kSPS for all eight channels. The Common Rejection Rate is as low as 110dB.

Two unique pins of the ADS1299 are SRB1 and SRB2. If the SRB1 bit is turn on, all negative input of 8 channels will connect to SRB1 pin inside the chip. This function is optimal for generic unipolar EEG measurement. The system reference electrode will connect to the SRB1 pin instead of connect to all negative inputs externally. Meanwhile, SRB2 is configurable to connect with any Positive input of all channels. It provides, even more, flexibility for a custom application.

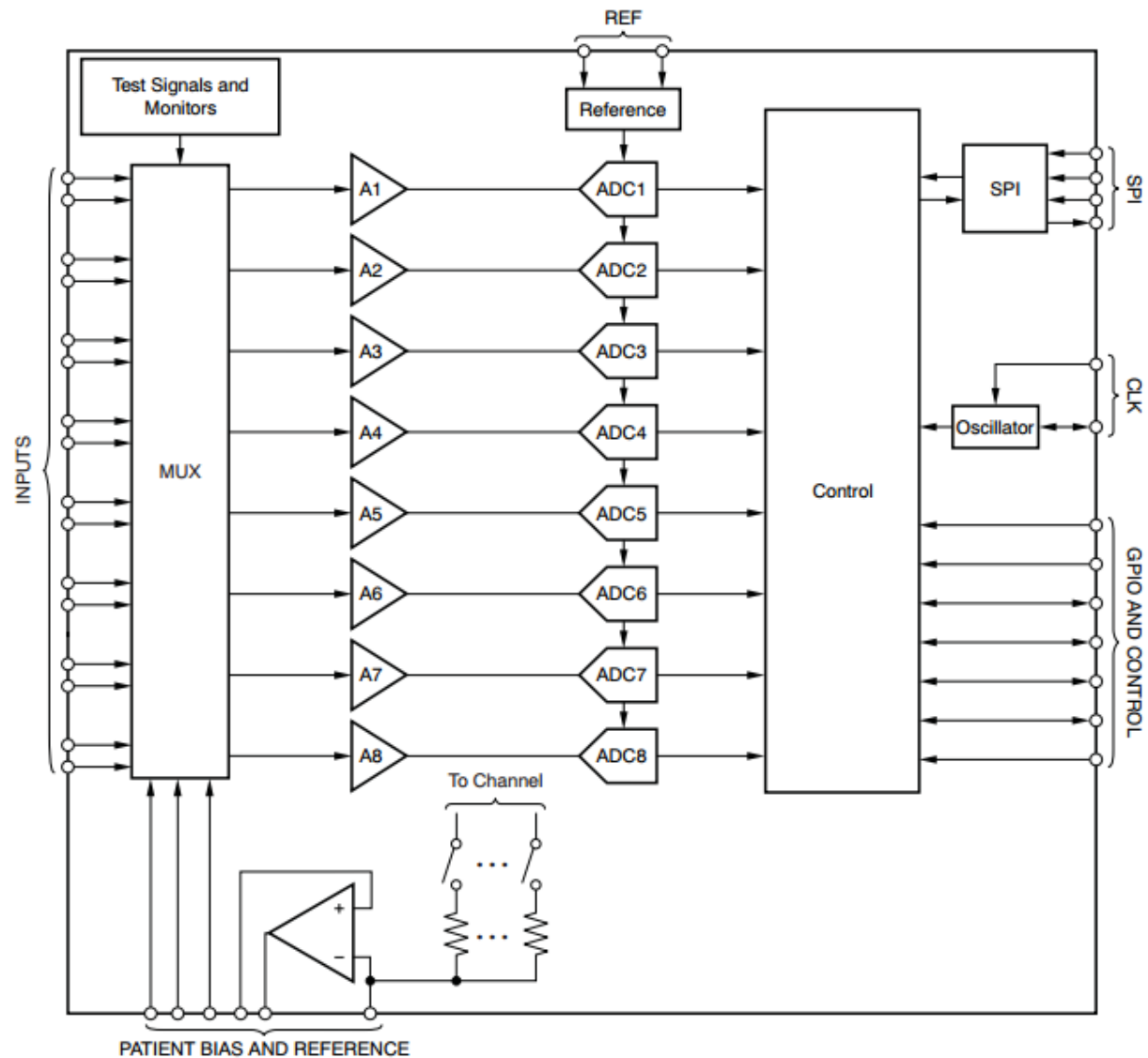
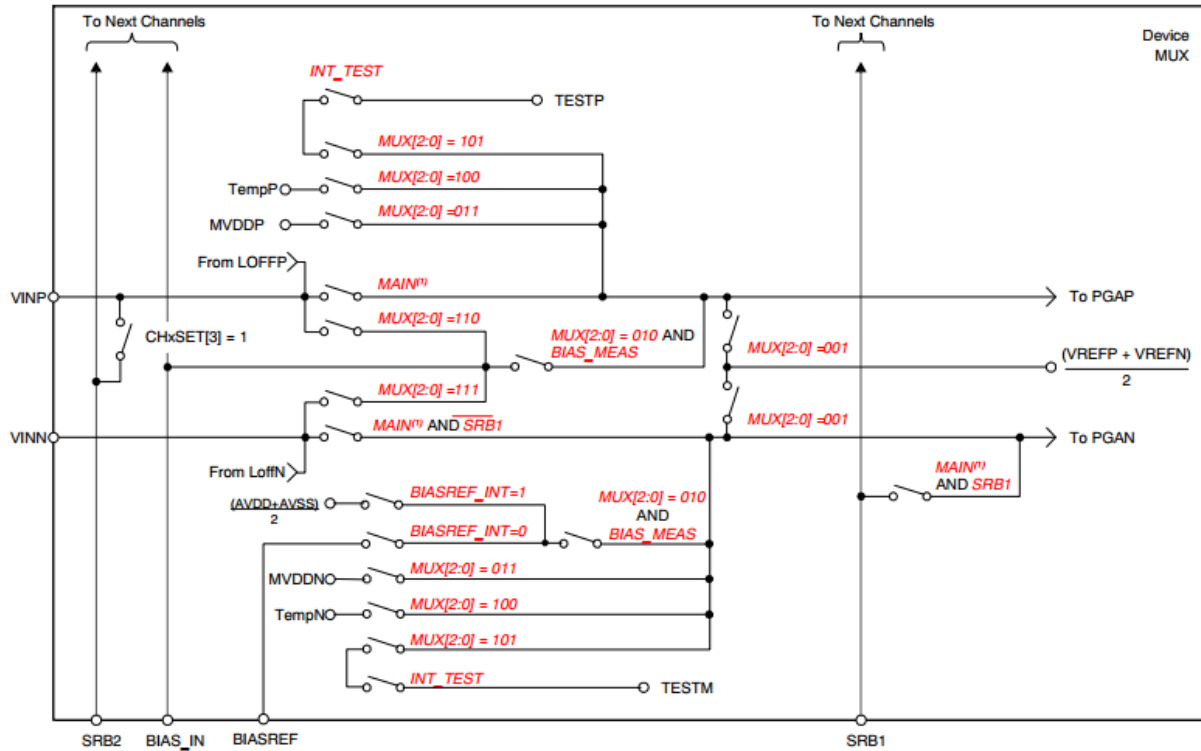


Figure 9: ADS1299 Block Diagram[13]



(1) MAIN is equal to either MUX[2:0] = 000, MUX[2:0] = 110, or MUX[2:0] = 111.

Figure 10: ADS1299 Input multiplexer block for one channel[13]

3.1.1.3. The power module

The power module consists of a MCP73832[16] charge management controller to manage battery charging activities. Follow up is the buck boost converter TPS63060[17] to maintain a 3.5V output for the digital system regardless the variable battery voltage level because the

battery voltage fluctuate from 4.2V to 3.2V depend upon on its capacity, whereas a 3.5V supply rail must be maintained.

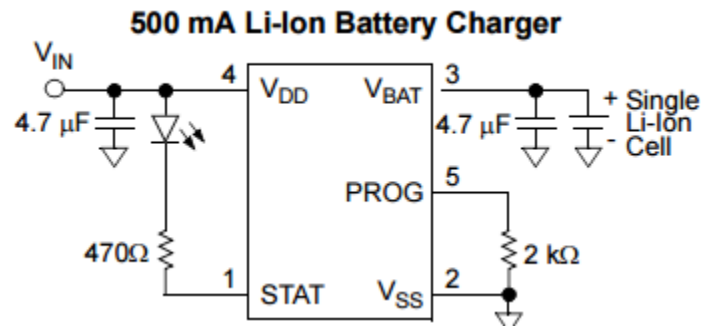


Figure 12 MCP73832 circuit diagram

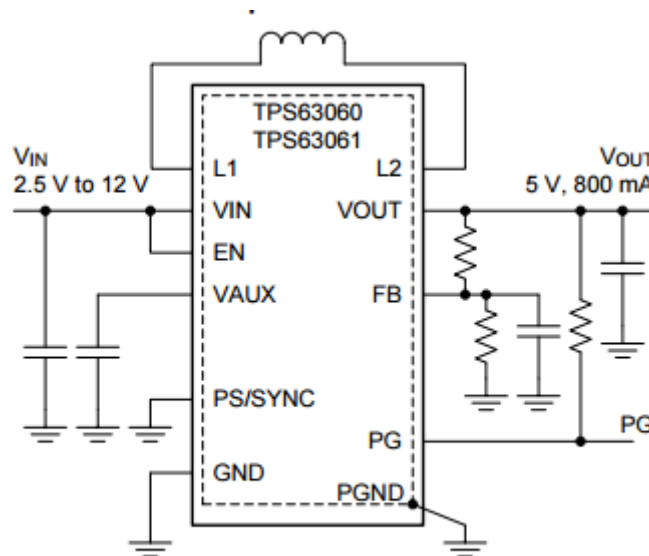


Figure 11: TPS63060 Simplified schematic

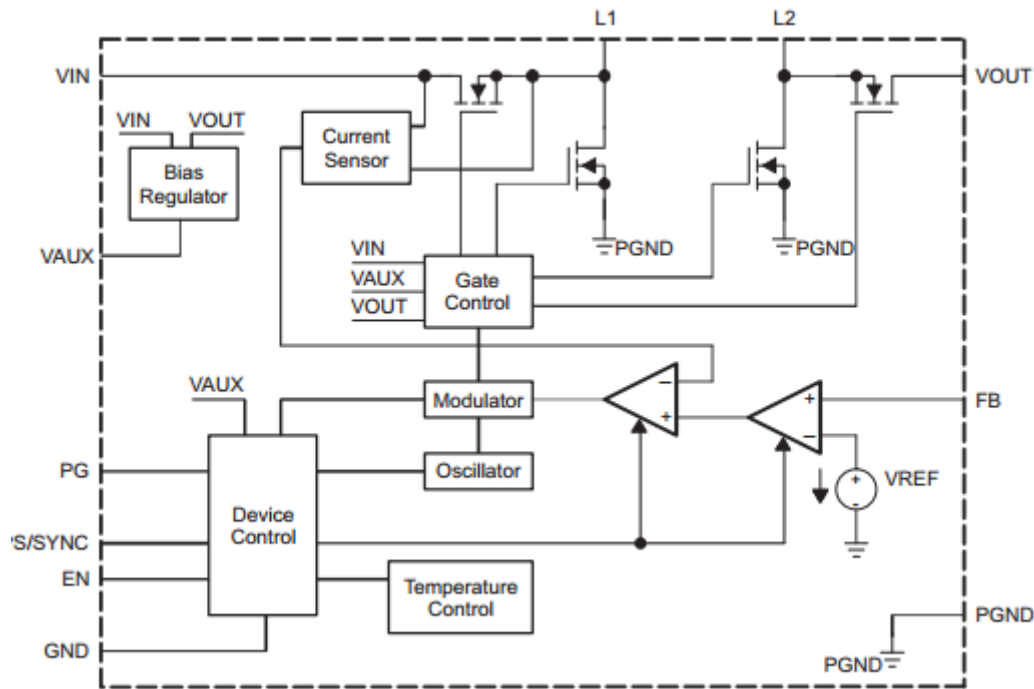


Figure 13: TPS63060 functional block diagram [17]

The circuit board also includes TPS72325 as a negative voltage regulator.

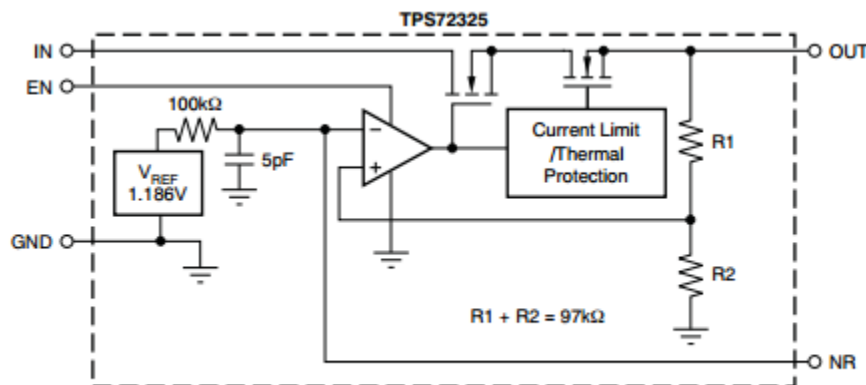


Figure 14: TPS72325 Funtional Block Diagram [18]

3.1.1.4. *TI ADS1299 Evaluation module*

The complete ADS1299EEG-FE Kit evaluation package comprises two hardware items, the ADS1299 Analog Front-End Module, and the motherboard MMB0. The AFE is stacked on top of the motherboard. The AFE converts data from analog to digital whereas the motherboard transfers these digital values to a computer via USB cable.



Figure 15: ADS1299EEG-FE Kit

The Daughter Card has comprehensive options to configure the module with more than 25 jumper block. Some distinctive functions are in the below hardware function list.

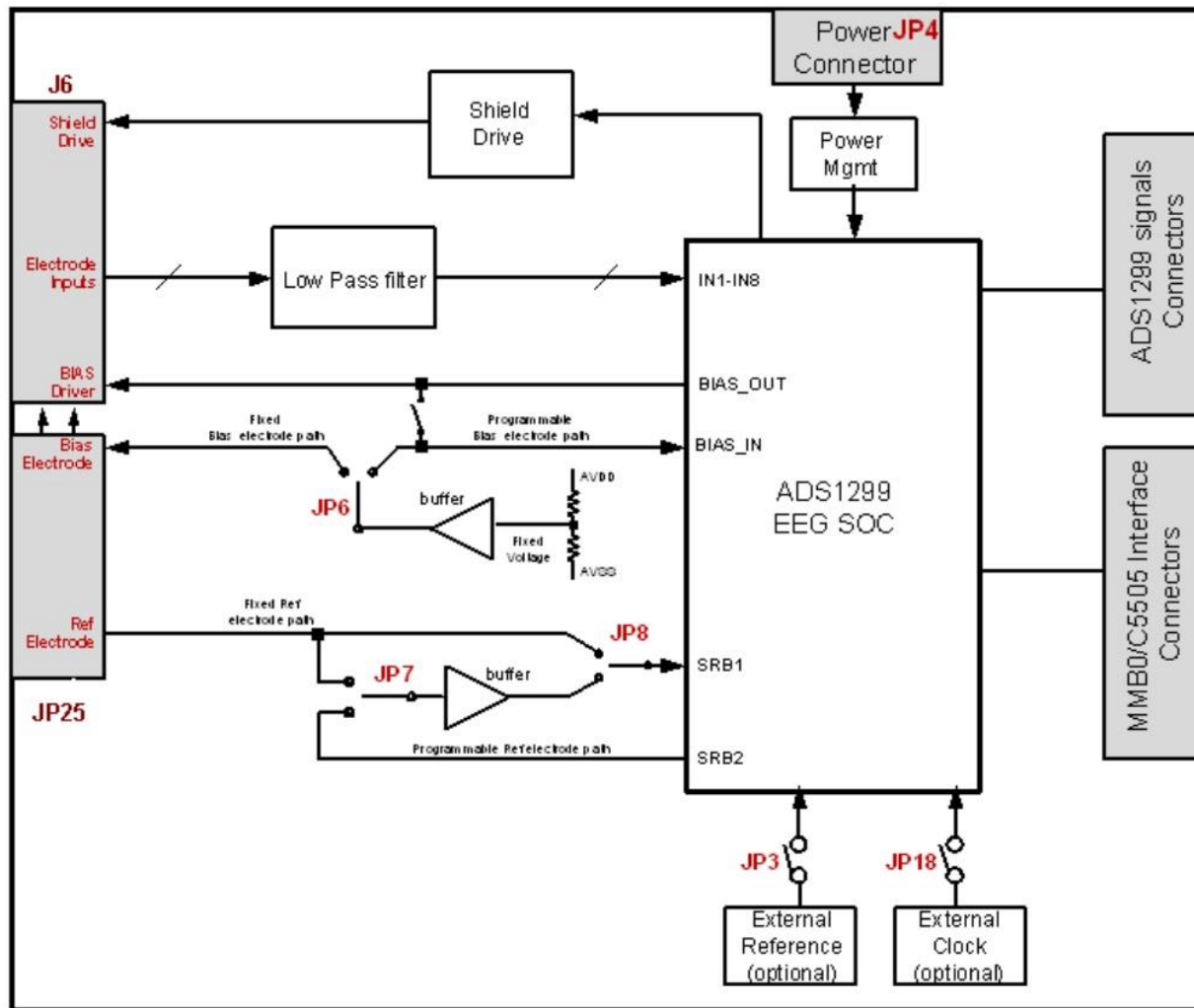


Figure 16: ADS1299 EEG-FE Daughter Card Block Diagram[19]

The Electrode Inputs go through Low Pass filters to screen out the spike and high-frequency component. The cut-off frequency for the Low Pass Filter:

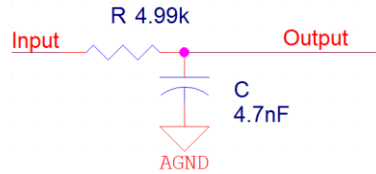


Figure 17: Low-pass filter for input channel

$$f_c = \frac{1}{2\pi RC} = \frac{1}{2\pi * 4990 * 4.7 * 10^{-9}} = \frac{1}{1.47 * 10^{-4}} = 6786 \text{ (Hz)}$$

The card is configurable for bipolar or unipolar supply operation via Jumper JP2 and JP20. The main power of the card is drawn from power connector JP4. For this project, Bipolar Analog Supply is deployed over unipolar power because bipolar design can take in negative input whereas unipolar power supply design cannot[20]. Jumpers are configured follow the third column in the table below.

Table 3: Analog Supply Configurations[19]

Power Supplies	Unipolar Analog Supply	Bipolar Analog Supply
	5V	±2.5V
JP2 (AVDD)	2-3 (default)	1-2
JP20 (AVSS)	1-2 (default)	2-3
U8	Don't Care	TPS72325
U9	Don't Care	TPS73225

Internal and external clock and reference are configurable via jumper settings. The external clock is used to ensure a reliable performance regardless outside temperature.

Table 4: Clock Jumper Options[19]

ADS1299 Clock	Internal Clock from the ADS1299	Clock from Oscillator on the EVM	External Clock Source
JP18	Not Installed	2-3 (default)	1-2
JP19	Don't Care	1-2	Don't Care
J3– pin 17	Don't Care	Don't Care	External Clock Source

This system employs a virtual ground bias. The virtual ground signal is routed out to an electrode via BIAS_ELEC net when JP6 pin 1 and pin 2 tied together. The signal is the mid-supply of $\pm 2.5V$ Analog supply. It's buffered by an Op-amp OPA371.

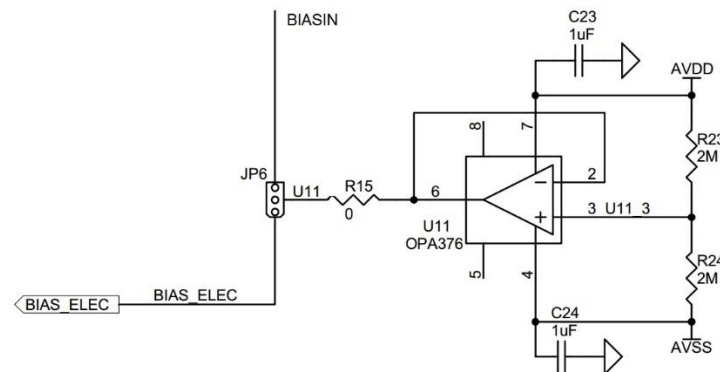


Figure 18: Mid Supply and Buffer Circuit

The board has an option to wire all negative input of channels to one channel via the SRB1 channel. This pin is useful for the unipolar measuring scheme because it minimizes negative wired connected to the subject. At the current stage of the project, as only maximum two channels are used for brain-computer interface application, all channel are kept independently without negative input tied together. One disadvantage of common reference is the interference between channels. Also, bipolar lead provides higher noise cancellation. The common noise will be attenuated significantly by the high CMRR op-amp.

The daughter card gives an option for active shielding. This technique reduces noise interference for long wires. Because the EEG electrodes in this project do not have shields, this function is not in use. Also, as positive and negative input wires are close together, noise will affect both lines. Later, the noise will be canceled out by the Op-Amp.

Besides, the motherboard MMB0 is a universal motherboard for a variety of data converter and communicate with the PC. This motherboard does not provide any filtering or signal processing. It forwards signal from the ADS12999 to the computer.



3.1.1.4.1. Evaluation Software

The Evaluation Software provides comprehensive configuration options for the device. Users may set ADS1299 register parameters via the software without writing any code. Other noticeable functions are offline FFT, virtual oscilloscope, and histogram.

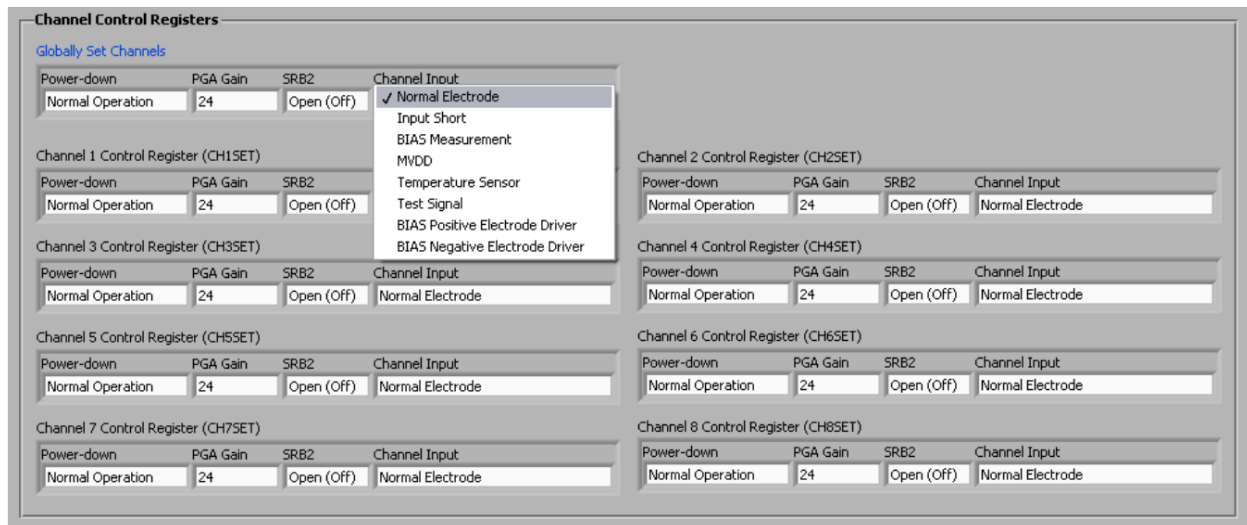


Figure 19: Channel Control Registers GUI Panel in the Software

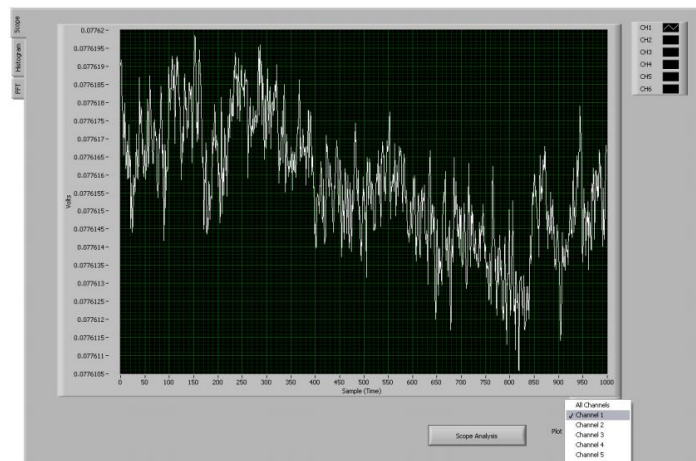
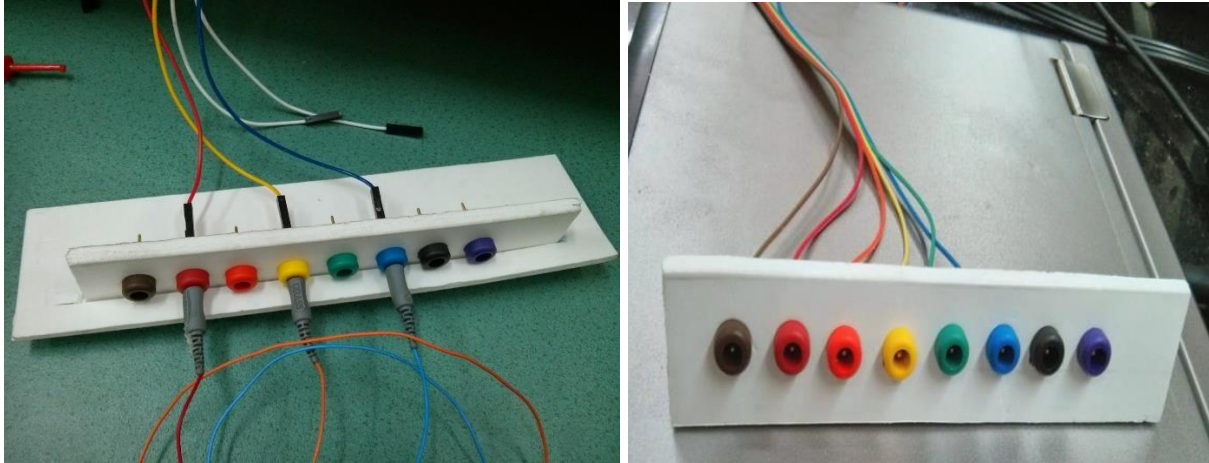


Figure 20: Scope Tool Feature of the Software



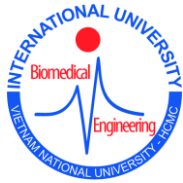
3.1.1.5. Electrode adapter



3.1.1.6. SPI Interfacing with the ADS1299 Module

This section will briefly describe the communication process between the ADS1299 SoC and the STM32F4 MCU via SPI Interface. Further in-depth details are discussed in Firmware development section.

At this point of the development process, the target is to read the device ID in the ID Control Register of the ADS1299 SoC. Every TI ADS SoC series has a unique device ID for identification. It's programmed into the factory at a read-only control register. For ADS1299, The ID is 1110, which is stored at the Address = 00h Register. This task verifies the digital communication process between the MCU and the ADS SoC.



ID: ID Control Register (Factory-Programmed, Read-Only)

Address = 00h

BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0
REV_ID3	REV_ID2	REV_ID1	1	DEV_ID2	DEV_ID1	NU_CH2	NU_CH1

This register is programmed during device manufacture to indicate device characteristics.

Bits[7:5] **Not used**
Bit 4 **Must be set to '1'**
Bits[3:0] **Factory-programmed device identification bits**
 1110 = ADS1299

Figure 21: ADS1299 Device ID Control Register from the Datasheet [13]

The snippet of the firmware to get this ID is presented below. The `mySPI_Send3Byte(0x20,0x00,0x00)` function read data at address 0x00 (follow the format of ADS1299 RREG Command). The switch case function identifies the SoC based on the return value. The idea of this snippet is derived from this project[21].

```
//Read data at address 0x00 of the Register MAP

mySPI_Send3Byte(0x20,0x00,0x00);

//Identify the ADS chip base on return value
switch (getIDval & 0x1F ) { //least significant bits reports channels
    case 0x12: //18
        gMaxChan = 8; //ads1298
        break;
    case 0x1E: //30
        gMaxChan = 8; //ads1299
        USART_puts(USART1, "ADS1299 Hooray!");
        break;
    default:
        gMaxChan = 0;
}
```

We successfully get the device ID with this code. This process is used at startup sequence to verify the connection between the ADS SoC and the MCU.

3.1.2. Hardware design

Discussion in this thesis reflexes characteristic of Prototype 3 because it is a stable version confirmed by comprehensive testing. Three main blocks of this design are consequently presented. First, the power management system in Figure 22 consists of a lithium battery charger Microchip MCP73832 to charge the lithium battery; a TI TPS63060 High Input Voltage, Buck-Boost Converter maintain a constant output, 3.5V on DVDD_STM net, despite input voltage variation causes by the onboard lithium battery. This voltage regulator is a critical component because the battery voltage fluctuates from 4.2V to 3.2V while the output voltage is required to be fixed at 3.5V. Only a buck-boost converter can handle this situation.

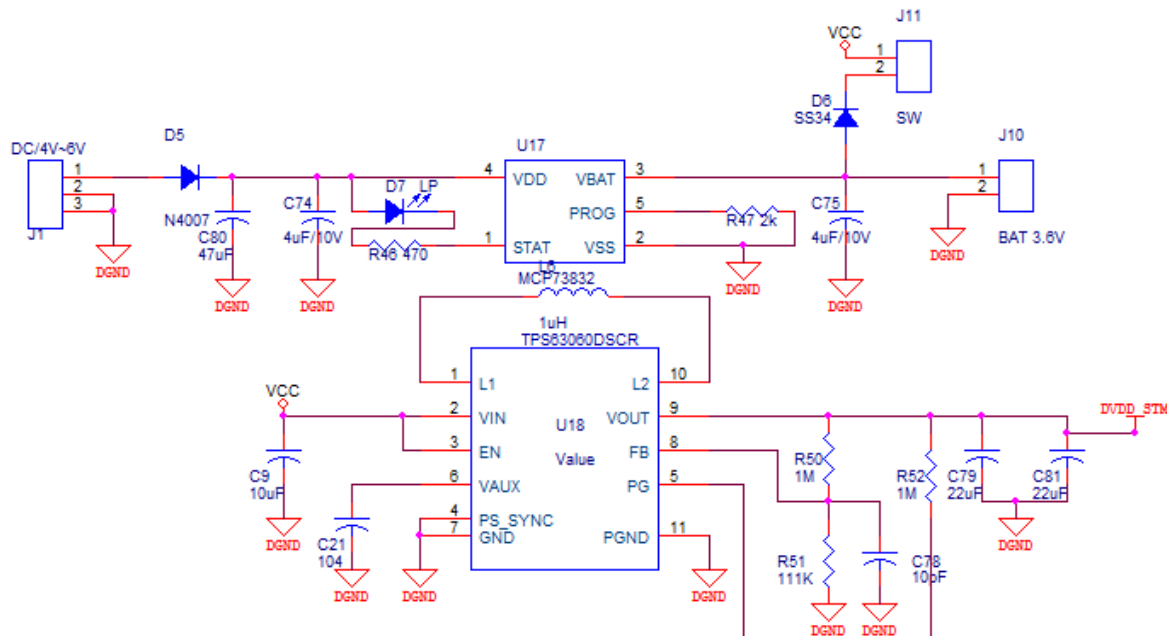


Figure 22: Main power management block

The sub power management block (Figure 23) takes the constant voltage 3.6V to produce a 2.5V by the TPS73225. This voltage supports the ADS1299 bipolar supply operation. In addition, this

sub module delivers -2.5V by the TPS60403 inverter chip and the negative output regulator TPS72325.

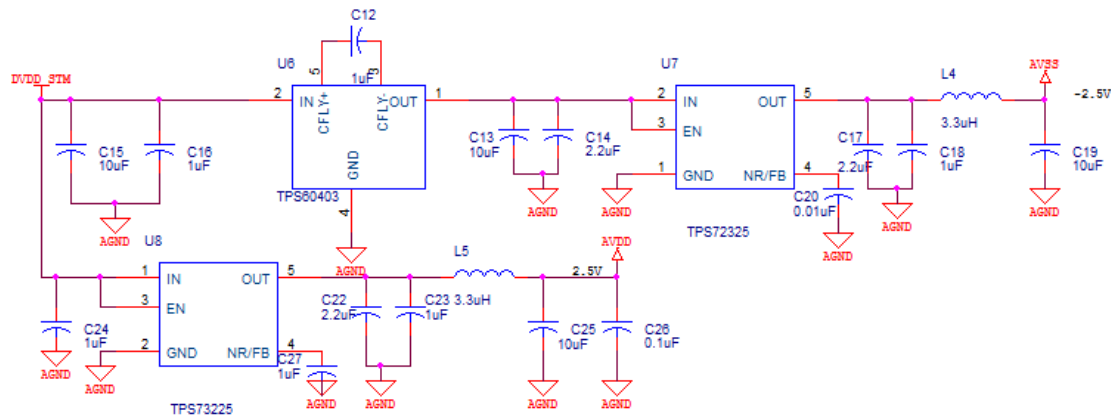


Figure 23: Sub power management block

Second, the analog block which built up around the ADS1299 is similar to the TI reference design. Pin 1 to pin 16 is eight differential channels input. The signal from the electrode will go through a low-pass filter in Figure 17 before reaching these channels. Figure 24 depicts the input for all eight channels. P1 to P8 are positive input nets; N1 to N8 are negative input nets.

The subsystem in Figure 19 illustrates another part of ADS1299 pinout. Pin 60 to pin 63 connect to the external bias circuit. In this design, the mid-supply bias scheme is deployed. The signal is generated by a bridge of 2 Mohm resistor R6 and R11. The mid supply signal is buffered by the U10 Opamp. The Oscillator OSC4 provides the clock for the ADS1299. It may be turned off to use internal oscillator because of energy saving. Pin 34, 38, 39, 40, 47, 43 are SPI pins of ADS1299 SoC. The rest pins are primarily for power supply.

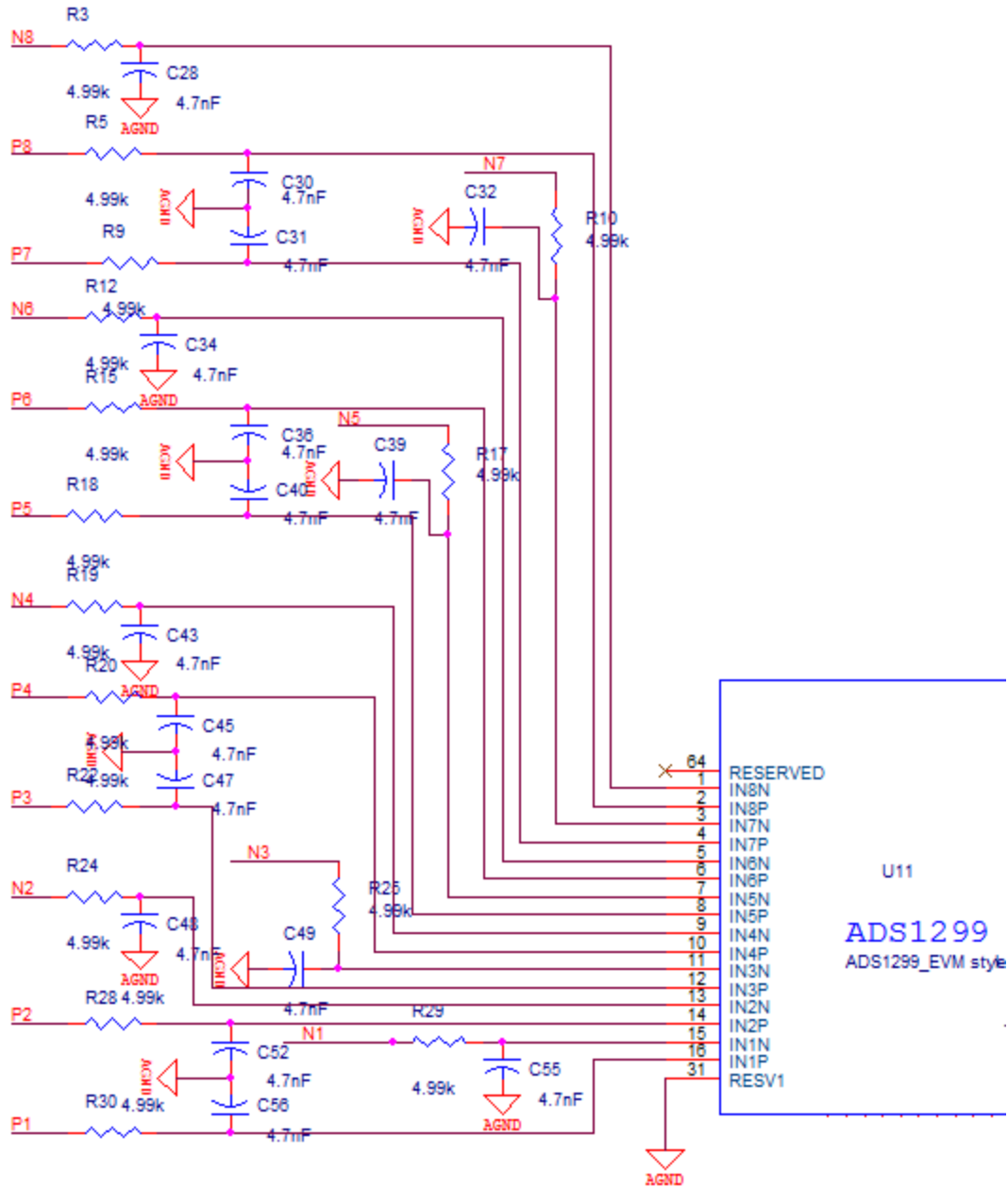


Figure 24: ADS1299 Input schematic

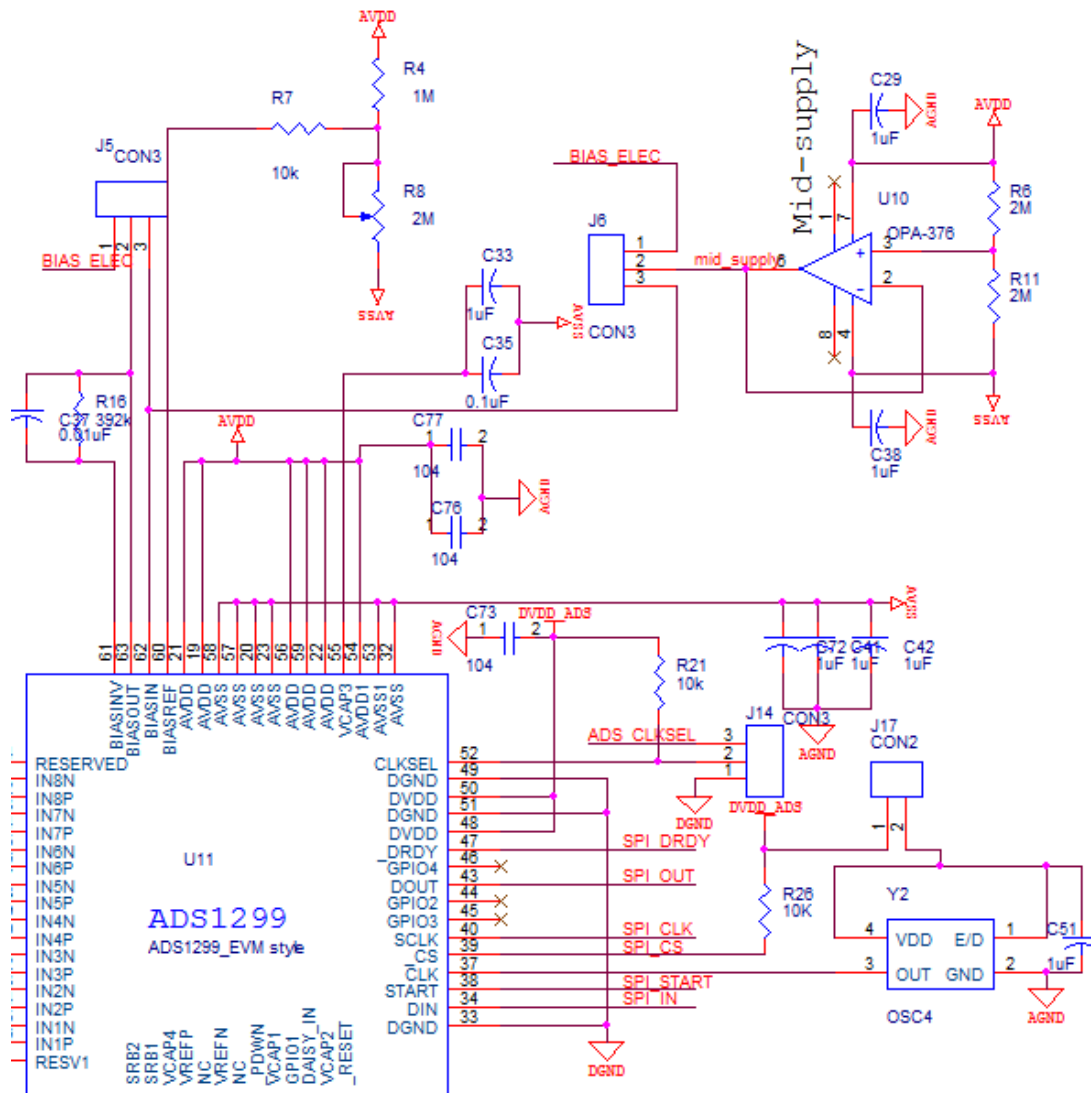


Figure 25: Sub Analog system

The last portion of the circuit board is the digital system which is centered around the STM32F407VG. Two major signal paths in this system are the communication path and the programmer path. The communication group consists of serial lines on pin PB6 and PB7; SPI lines on pin PA5, PA6, PA7, and PB1. In Prototype #4, the 64 pin package is used instead of the 100 pin package to reduce redundancy pins.

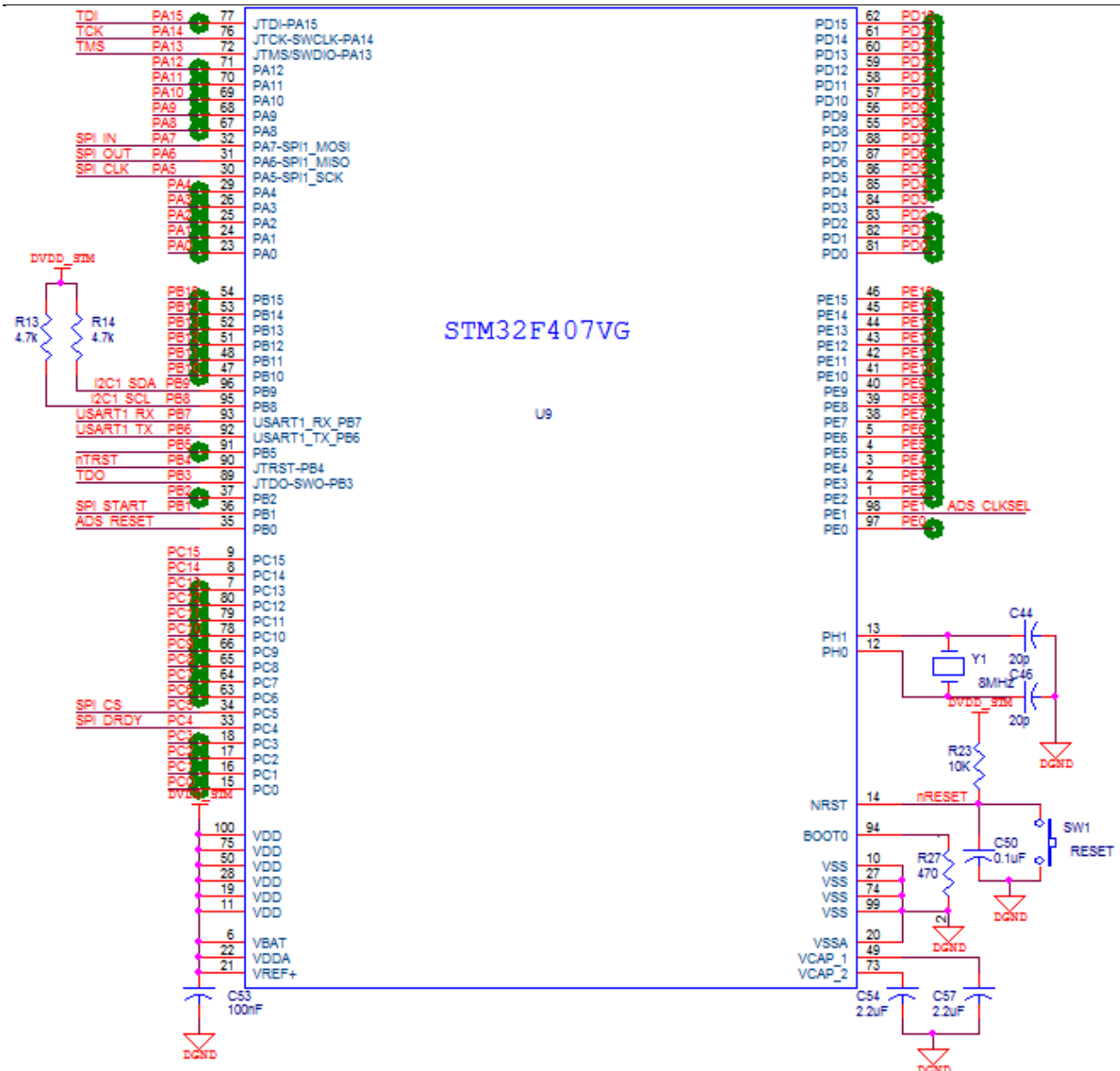


Figure 26: Digital system schematic (Prototype #3)

3.1.3. Communication

Because this is a physiological device which got many electrodes attach to a human body, it must have a proper isolation from the main power line. Otherwise, the subject under study may get an electrical shock with high voltage and low impedance. Even though meticulous isolation design



may be put in place, the best way to be safe is to run this device on battery all the time. All prototypes can be powered by a standard USB 500mA port. At the early stage, when a possible wireless connection is not developed, the WEEG is tethered to a laptop without a power cable or monitor cable plugging into it. Please pay attention that monitor cables such as VGA, HDMI, or DisplayPort may carry power line leak current to the laptop. In brief, unplug any cable attached to the laptop beside the USB-Serial cable.

Wireless communication is the ultimate goal for this project. At the current state, the device reaches a remarkable wireless transfer rate at 500SPS for simultaneous eight channels. This data rate is excess the speed of other consumer grade EEG devices. Other applications usually transfer at 250 SPS or lower.

3.1.3.1. Serial Communication

The UASRT Serial Communication is the protocol of choice for embedded system. It is ubiquitous, simple, and reliable. One limitation of UART is the data rate which is top at around 2Mbyte/second. Because this is an asynchronous system, the high data rate may lead to bit rate errors. The theatrical output of the scheme is estimated as below.

$$\text{Data rate} = \text{Data Per Sample} \times \text{Sample Per Second (SPS)}$$

Data per sample: equals to the size of every data frame mentioned in Figure 27. Each data frame consists of 37 bytes of information.

Sample per second: configurable in the firmware. The maximum SPS for this system is 500 SPS.

The UART configuration in this system is 8 bit and no parity bit. Combining every 8 bit of information with a start bit and a stop bit of the protocol, every serial data frame includes 10 bits.

$$\text{Data rate} = (37\text{byte} * 10\text{bit}) * 500 \frac{\text{sample}}{\text{second}} = 185,000 \text{ baud}$$

The next possible baud rate level is 230,400 baud (~30 kB/s). This value is tested and utilized in this system.

The bit rate error at this baud rate is lower than 1%. It is well below the recommend maximum bit rate error 2% to ensure error-less communication [22]

3.1.3.2. Wireless Communication

There are a plethora of wireless modules to transfer data. To narrow down choices, the basic parameters are reviewed to select an appropriate module. First, power consumption is one of the most important factors to choose a wireless module. The goal of this system is a battery powered device which last for a day (24 hours). Another major constraint is the data throughput of the wireless module. According to the transfer rate calculation in 3.1.3.1, the protocol should be able to transfer at 230kbps. With these bounds, Bluetooth stands out as the best fit for this application.

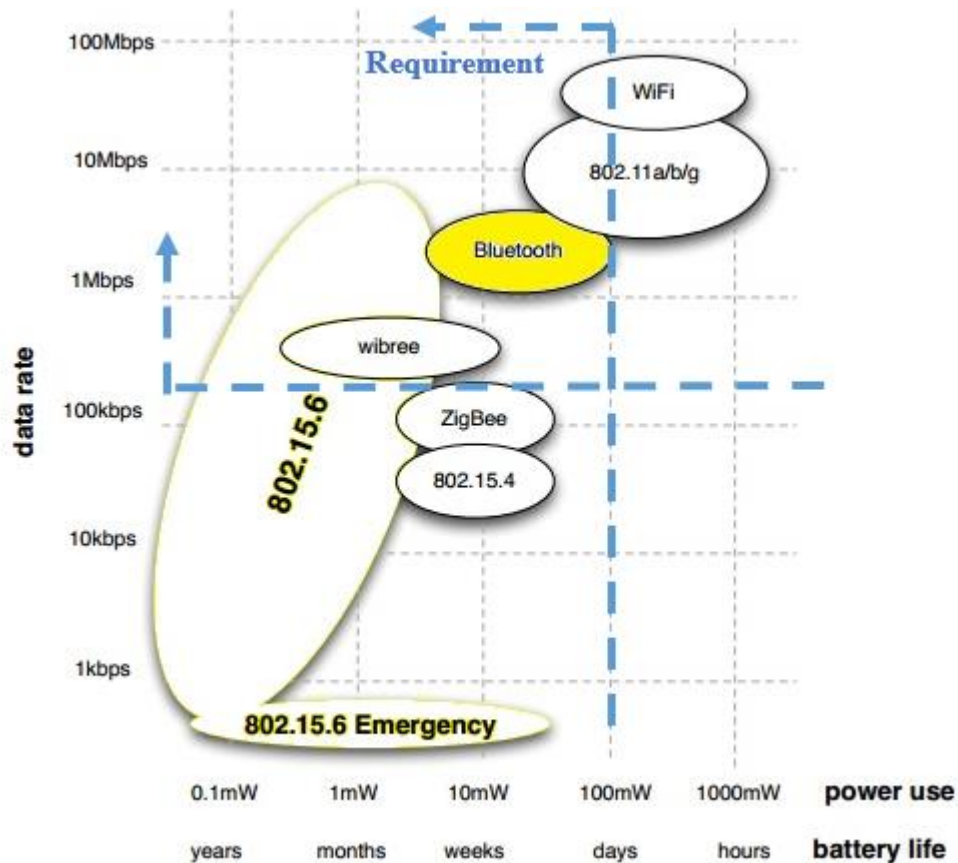


Figure 27: Data Rate vs. Power Consumption [23]

For implementation, two Bluetooth to serial port HC-05 modules is set up as Master and Slave. This module is relatively cheap at \$3.3 per module. The Master module is connected to a computer; whereas the slave module is attached to the WEEG. The baud rate for both modules are at 230800 baud. The address of the Slave module is “BIND”ed to the Master module by AT command. Two module automatically pair with each other shortly after turned on, as long as the Master module keeps the address of the slave module.

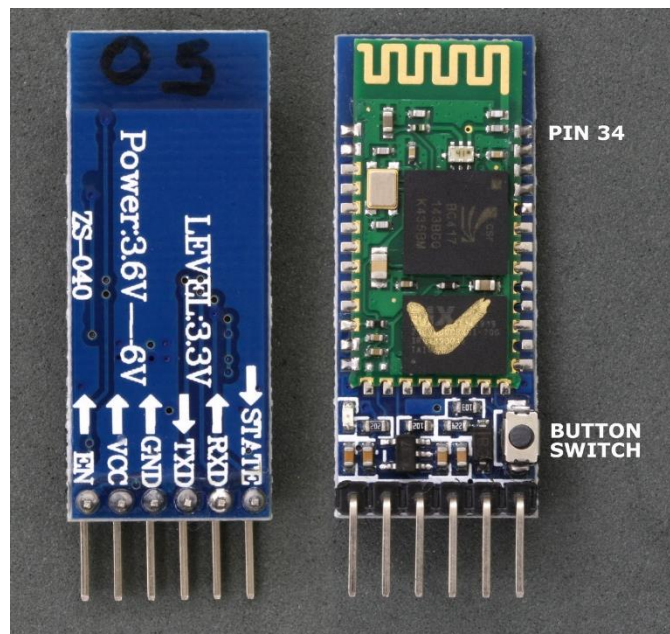


Figure 28: HC-05 Bluetooth Module[24]

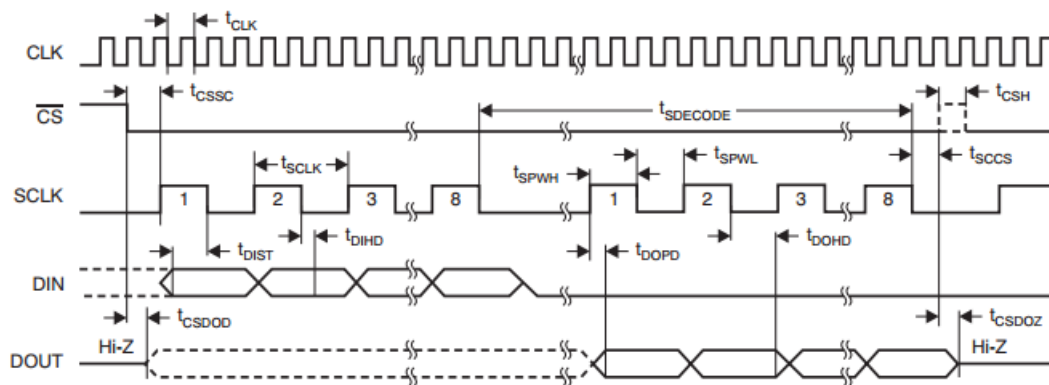
The HC-05 is a v2.0+Enhance Data Rate Bluetooth with theoretical data transfer rate up to 2Mbit/s.[25] This speed is more than enough for WEEG use. Bluetooth Low Energy was considered, however, the package size for BLE is limited at 23 bytes/package per spec. The throughput is 0.27 Mbit/s which is barely enough for this application.

3.2. Firmware development

3.2.1. Current design

The firmware content of the microcontroller, depicted in Figure 23, is consistent throughout four prototypes. Right after power up, the microcontroller initializes input and output ports, communication blocks (USART, I/O, SPI and Timer). The USART speed is usually set up as 230400 baud. SPI parameters are set up according to the ADS1299 SPI specification. SPI clock limit of the ADS1299 is 20MHz; the minimum speed is 110 kHz for 500 SPS mode[11]. Therefore, the chosen speed is 2.6 MHz which is in the middle of two bounds.

Figure 29: Initiate SPI parameters



NOTE: SPI settings are CPOL = 0 and CPHA = 1.

Figure 30: SPI Timing characteristic of ADS1299

Consecutively, ADS1299 SoC is enabled via Chip Select (CS) pin. The microcontroller will verify if the SoC is functional properly by checking information in the ID register of the IC at address 00h[13]. If the microcontroller receives a correct response from the ADC, it will proceed on initializing the ADC parameters such as sampling rate, reference voltage source, and clock source. Every recording channel property is also set up such as Programmable Gain Amplifier setting,



Vietnam National Universities – HCMC

International University

Biomedical Engineering Department

SRB2 connection, and MUX channel input. Other control registers, Bias Sensing and Lead Off Sensing, are also set up. Then, the microcontroller set the START pin High to allow ADS1299 to send out data.

As presented on the right side of Figure 31, the microcontroller read recording data from the ADC continuously after completing setting up the system. It reads every incoming data package, pre-processes them. During the pre-processing step, two's complement data from ADC is converted to an integer (binary). Every channel 24-bit data is put into a 32-bit variable. Because the UART for the STM32F4 is capable of sending 8bit serial only, 32-bit variable is shifted 8 bit at a time to the sending register. Then, the Bluetooth module HC-05 will convert the 8-bit signal on this serial line to RF signal for wireless transfer.

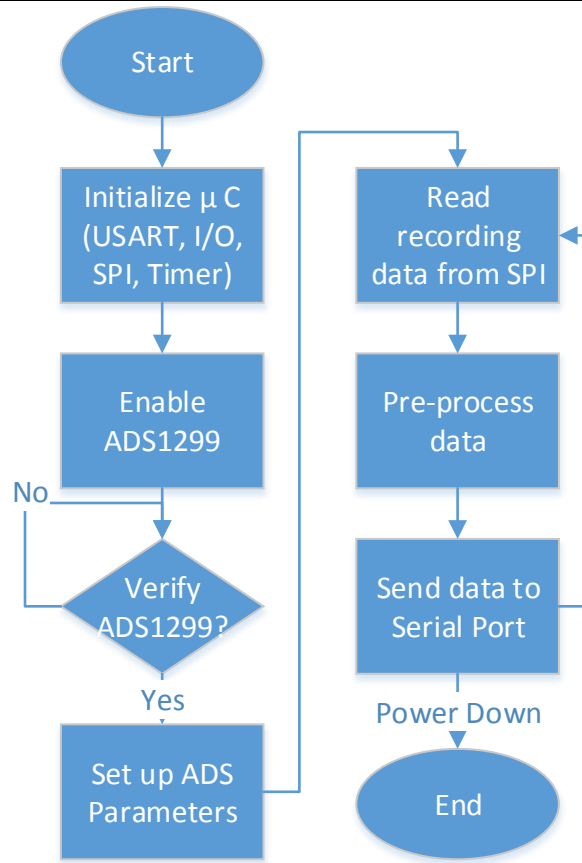
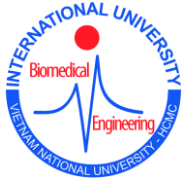


Figure 31: Firmware process flow

3.2.2. Data Format

Every data sample from the ADS1299 consists of 27 bytes of data. They include 3 bytes of system status information in a binary format and 24 bytes of recording information in two's complement format. This information is transferred to the MCU STM32F40x via SPI protocol. In the STM32F4 firmware, every channel data is converted from two's complement to unsigned binary for the ease of use in the computation software. After that, these binary numbers are split to 8-bit chunks for serial protocol because the UART of the STM32F4 can transfer only 8 bit at a time. The HC-05 Bluetooth module pairs act as a transparent system to replace Serial Port Wire.



Vietnam National Universities – HCMC

International University

Biomedical Engineering Department

Header bytes and Counters bytes are added by the MCU to support data integrity assessment in the system.

INPUT SIGNAL, V_{IN} ($A_{INP} - A_{INN}$)	IDEAL OUTPUT CODE ⁽²⁾
$\geq V_{REF}$	7FFFFFFh
$+V_{REF} / (2^{23} - 1)$	000001h
0	000000h
$-V_{REF} / (2^{23} - 1)$	FFFFFFh
$\leq -V_{REF} (2^{23} / 2^{23} - 1)$	800000h

(1) Only valid for 24-bit resolution data rates.

(2) Excludes effects of noise, linearity, offset, and gain error.

Figure 32: Ideal Output Code versus Input Signal

The minimum Voltage step of this system is calculated with following declarations. Programmable Gain Amplifier is set to 12. The internal reference voltage is 4.5V

$$V_{step} = \frac{V_{ref}}{(2^{23} - 1) \times PGA} = \frac{4.5}{(2^{23} - 1) \times 12} = 44.7 \text{ (nV)}$$

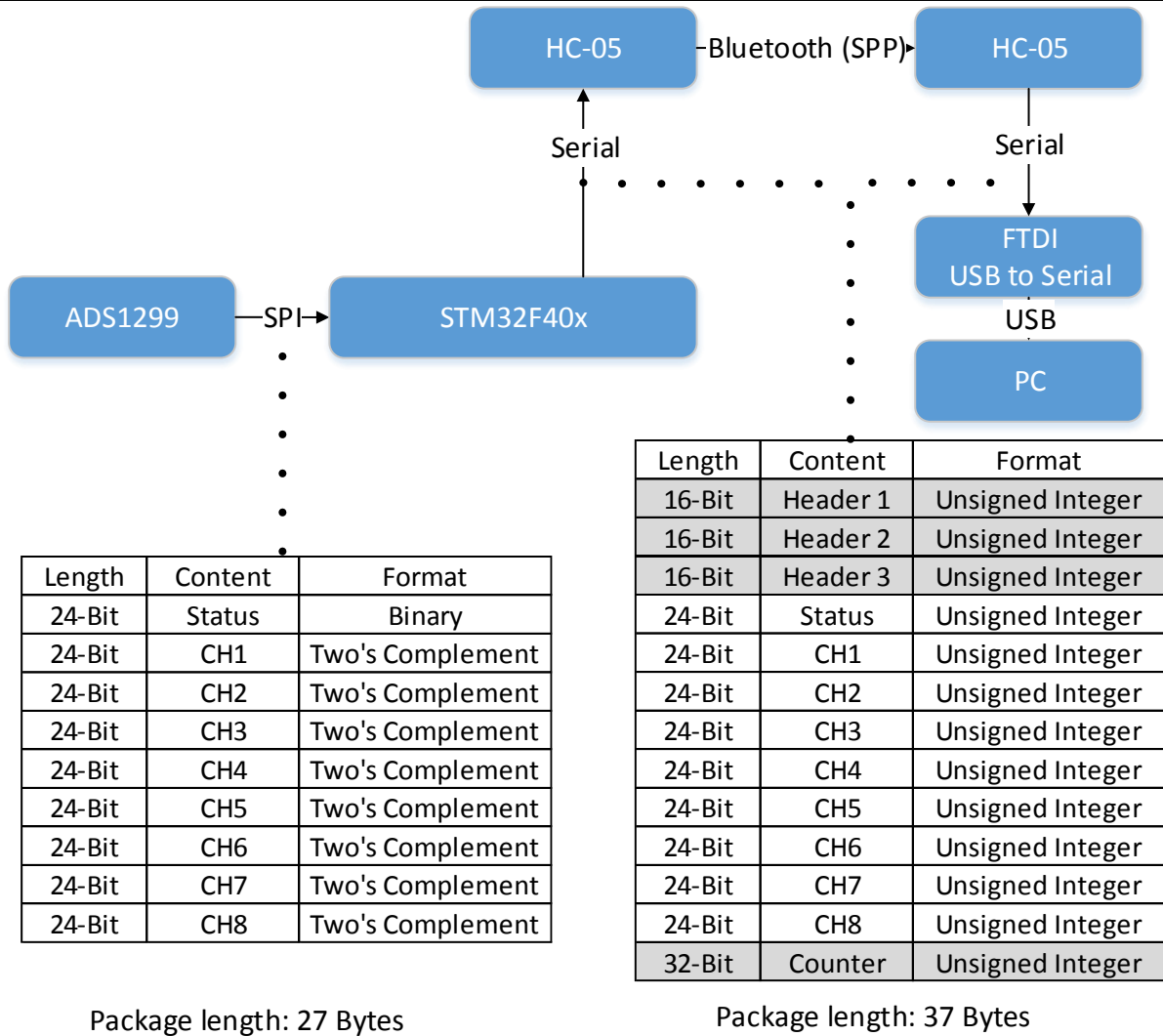
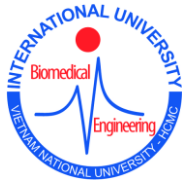


Figure 33: Communication Protocol and Data Type

Header 1, Header 2 and Header 3 are identical. They are 16-bit data which defined as 0xFE01. These information is used to identify the beginning of a new data byte. The Counter byte keeps track of number of bytes sent out. The signal processing software check this value to determine signal integrity.



3.3. Software development

3.3.1. Major software block

A graphical user interface to monitor signal acquisition is a must for this system. It will visualize the recording value and provide a configuration option to connect with the hardware. Two viable options are present at the beginning of the development process, either using an existing EEG acquisition software, e.g., Brainbay, OpenVibe or developing one from scratch. Brainbay was tested at first; ultimately, the main software is developed in Matlab for this project.

BrainBay is designed to support OpenEEG-Hardware. It is a Bio and Neurofeedback Application, developed by Christoph Veigl and Jeremy Wilkerson. Data acquisition of OpenEEG may be different compares to other hardware e.g. Emotive. Other users may write driver to connect other hardware with Brainbay by C++. Brainbay interpret 16-bits per channel[26] whereas my system is capable of output 24 bits per channel data. For those reasons, the author decided to build a software with Matlab.



Figure 34: Brainbay screenshot

3.3.2. Current design

The version 2 of the design is a stable and full of features software. It evolves from version 1 with the bug fixed and more functions. Key functions of the software are discussed below.

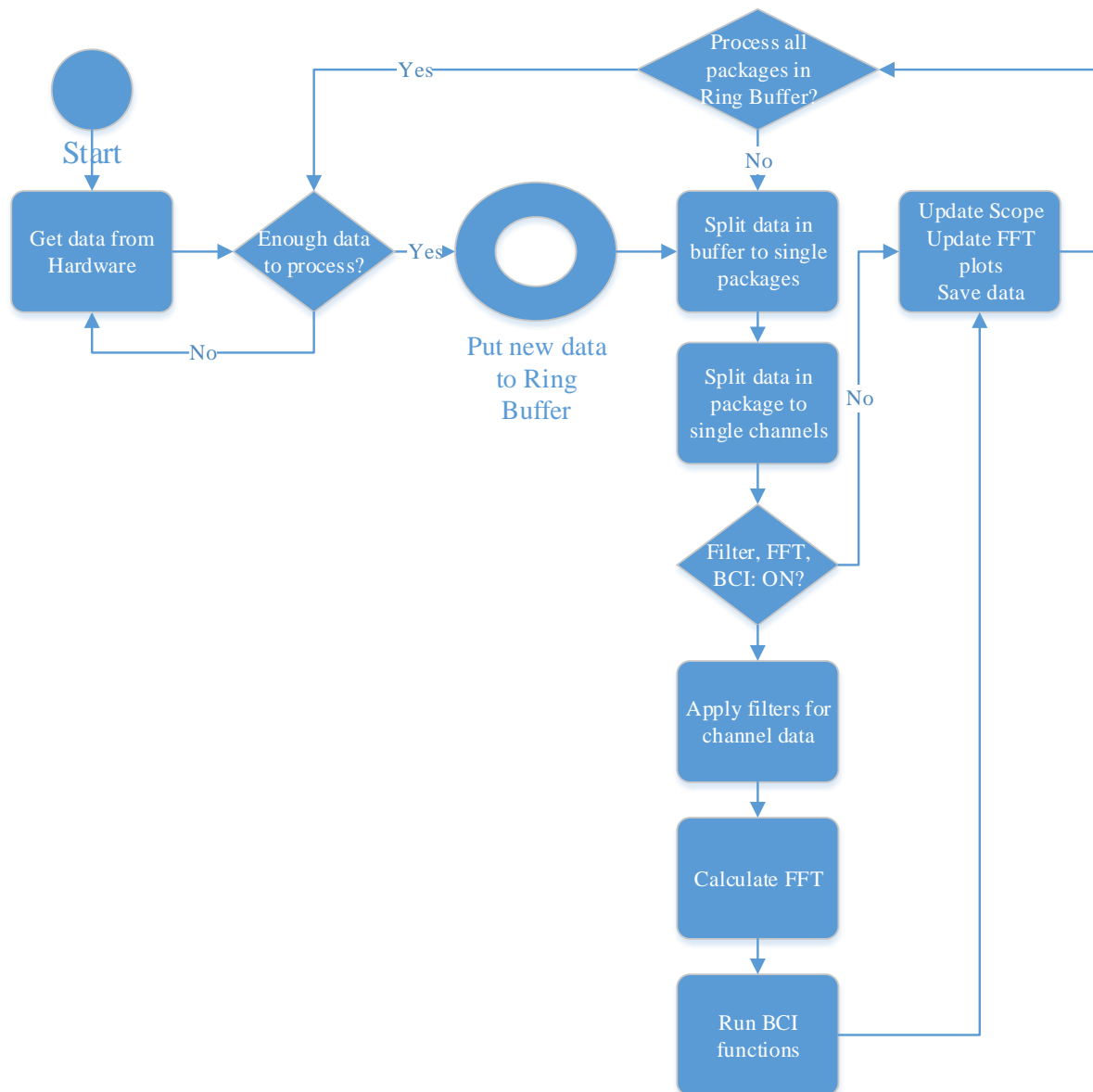


Figure 35: Dataflow diagram



3.3.3. Filtering

The real-time filtering is a crucial function for BCI application. It is also great to support real-time monitoring. The GUI provide both finite impulse response (FIR) filter and Infinite impulse response (IIR) filter. The user may specify desired cutoff frequency and filter order.

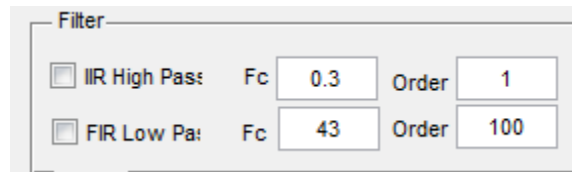


Figure 36: User Interface of Filter feature in the GUI

The recommended filter for this system is the IIR High Pass filter and FIR filter. Default filter values are initiated at the beginning. The IIR was used as the high-pass filter to block DC component. It provides great frequency respond with a low order filter. The FIR low pass filter is very effective to attenuate high-frequency components.

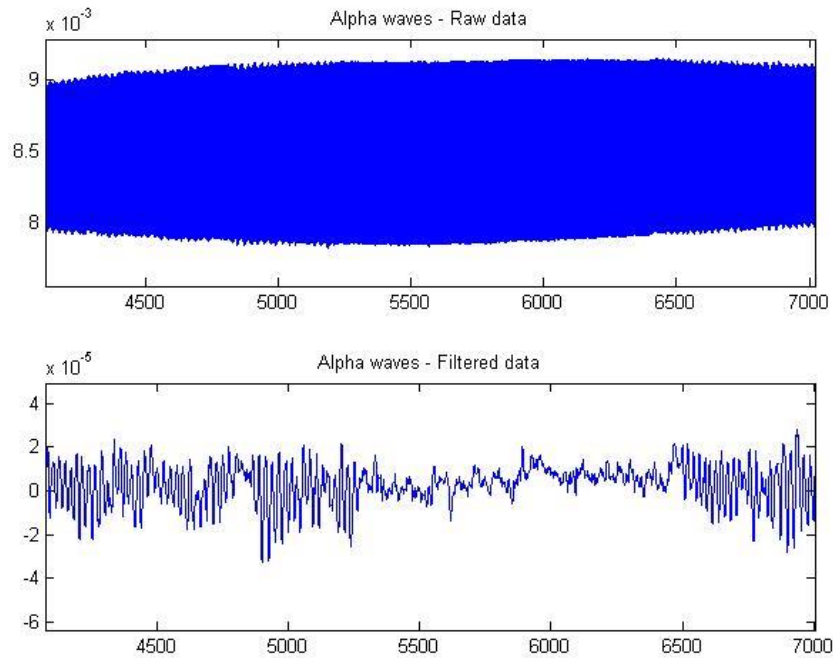


Figure 37: Demonstration the effect of filters

3.3.4. Ring Buffer

A variety of buffers is implemented in the software. The majority of those buffer are First In First Out buffer (FIFO). These buffers are simple to implement but may be inefficient in term of processing time. Especially to handle a data stream, the software performs many moves in and move out commands. The indexes of all elements inside the buffer have to be updated after every command. These tasks increase the processing time significantly. A Ring buffer or Circular buffer is implemented to resolve this issue.

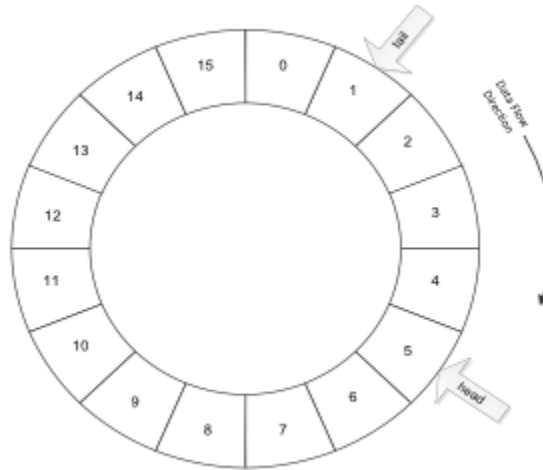


Figure 38: Structure of a Ring buffer

The circular buffer is a queue buffer which two of its ends are connected. The structure is well suited for the data stream. Data is PUT in at the Head index. Data is READ at the TAIL index. By moving these two indexes instead of moving every data values in the buffer, this structure saves time for program execution.

The ring buffer allows asynchronous data from the serial port move in at the write pointer. At the read pointer, data is taken out, one package at a time. In the MATLAB code, a 5000 location ring buffer is used to handle data stream. It starts at index 1 and ends at index 5000. This buffer contains ~135 data sample which is about 0.5s of data if SPS is 250. If the SPS is 500, the content is only 0.25s of recording time. Keeping these value low while maintaining a stable stream is a good practice.

3.3.5. FFT

The real-time FFT is beneficial for monitoring EEG in the frequency domain. During SSVEP experiment, stimulation frequency distinctly shows up in the FFT plot. When subjects close their eyes, alpha waves signal power rise upward.

The FFT algorithm in the software follows suggestions in [27]. A 1024 point FFT is performed with 512 data samples padded with zero. The zero padding technique supports peak finding and an algorithm for BCI application. The frequency resolution is $250/1024 = 0.244$ Hz. The interval between FFT plot refresh is 0.5s. The overlap of consecutive FFTs is 75%. These configurations serve well for examined BCI application.

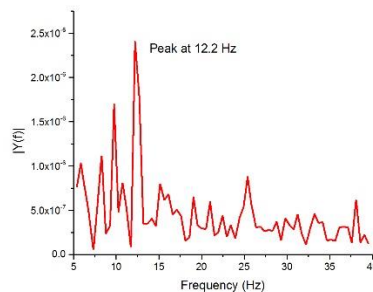


Figure 39: A sample of FFT plot during an SSVEP experiment

Figure 40 simulates the peak finding results for a cosine signal with a center frequency equal to 11. The sampling rate is 250. Some data for FFT are 100 with a zero-pad factor equals to 4. This is an exaggerate example to depict the benefit of zero padded signal in the peak finding process. Unpadded FFT shows that the examined signal has peak frequency at 10 Hz whereas the padded signal exposes the peak at 11.25 Hz. None of these values is the correct center frequency. However, the zero padded signal presents a closer guess with less error in terms of frequency and amplitude.

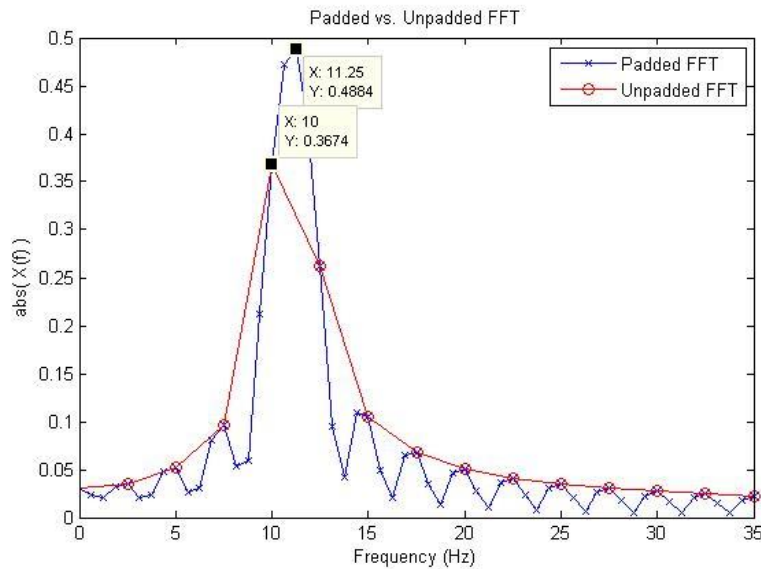


Figure 40: Padded vs. Unpadded FFT to find a peak frequency at 11 Hz

Matlab implementation for the FFT function.

```
%datatest stores 512 data point from recording values
Fs = 250; % Sampling frequency
T = 1/Fs; % Sample time
L = length(datatest); % Length of signal
t = (0:L-1)*T; % Time vector
% Multiply NFFT by 2 to get 1024 FFT data point, 512 point is from the data,
% zero padded the rest
NFFT = 2^nextpow2(L*2); % NFFT = 1024
Y = fft(datatest,NFFT)/L;
f = Fs/2*linspace(0,1,NFFT/2+1);
y = 2*abs(Y(1:NFFT/2+1));
```

3.3.6. Spectrogram

As EEG signal associated with frequency, the spectrogram is a key of every EEG signal processing software. Raw data from the recording are filtered with a band pass 0.3-43Hz. The spectrogram function leverages MATLAB spectrogram function. *spectrogram* (*x*, *window*, *NFFT*, *F*, *fs*)

spectrogram(*x*, *window*, *noverlap*, [], *fs*)



Vietnam National Universities – HCMC

International University

Biomedical Engineering Department

Where:

- x is the input signal vector
- $window$ is the number of DFT points
- $noverlap$ number of overlap point between two consecutive window
- f_s is the sampling frequency.

There are many combinations of configuration for the function. A heuristic proposal input as below:

- $segmentLength = round(numel(\frac{x}{4.5}));$ %. This is the default value of Matlab
- $window = round(\frac{segmentLength}{6})$ %. By default, window is equal to segment length
- $noverlap = round(80/100 * segmentLength/6)$ %. The default value is 50%. The author recommends increasing this number to 80%.

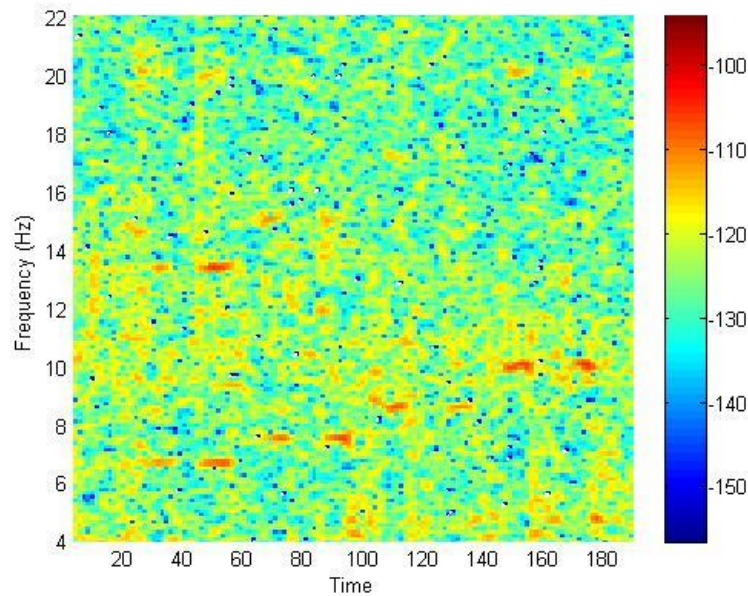


Figure 41: Sample of a spectrogram of an SSVEP Experiment

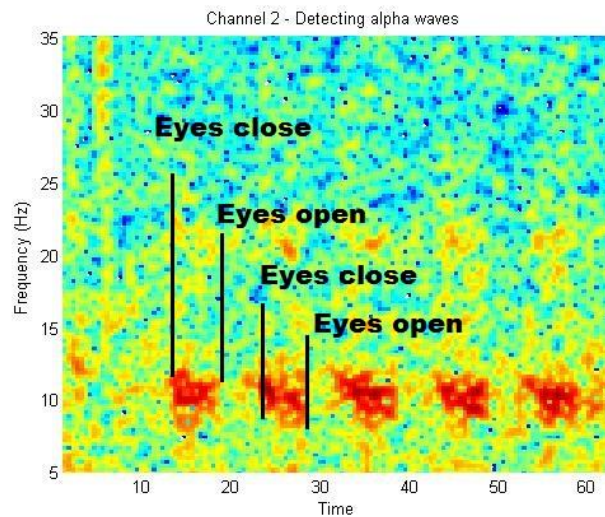


Figure 42: Sample of spectrogram to detect alpha waves



4. Chapter 4

Evaluation and results

Extensive evaluations were deployed to evaluate the performance of WEEG. Results prove that WEEG can record EEG signals such as alpha band and Steady State Evoked Potential. Moreover, the comparison between this device and another traditional “gold standard” EEG recording device, Biosemi Active Two, yields satisfactory values.

4.1. Alpha wave

This experiment concentrates on the analysis of the alpha rhythms (in the range of 8-12 Hz). Alpha brain waves boost up in EEG signal when subject's eyes are closed, and when subject's eyes are open, alpha waves' amplitude reduce. This is an ordinary feature of EEG data processing. From the standpoint of a hardware designer, it is an important assessment for EEG recording hardware to verify that the system can measure ultra low brainwave signal.

Protocol: The subject is a student (23-year-old) who participated voluntarily in this experiment. The experiment comprises two phases. First, the subject sits and relaxes on a chair with his eyes open for 15s. There are none stimuli in front of the subject's eyes. Second, the subject closes his eyes in 5s and opens his eye in 5s for every trial. This protocol is illustrated in Figure 43. Every run consists at least five trials. We conduct five runs in the session. EEG data is recorded in two differential channels. Gelled Electrodes are placed on the subject's scalp according to the International Electrode (10-20) Placement System. One Ground (bias) electrode is placed on the left mastoid. The differential pair of channel two is placed on the Oz and O₂. The sampling rate is 250 Hz. EEG data was high-pass filtered at 0.3Hz to avoid DC drift and a low-pass filter at 43Hz because the signal of interest would be around 8-12Hz. Data is analyzed and visualized using a Matlab Graphical User Interface (GUI). The Power Spectrum Density (PSD) graph of one run is presented in Figure 44. The result coherences with the hypothesis in which PSD of alpha waves is



high (in red color) when subject's eye is close; PSD of alpha waves is low when subject's eye is open and relax.

Eye open 15s	Eyes close 5s	Eyes open 5s	Eyes close 5s	Eyes open 5s	...	Eyes close 5s	Eyes open 5s
--------------------	---------------------	--------------------	---------------------	--------------------	-----	---------------------	--------------------

Figure 43: Protocol for detecting alpha waves

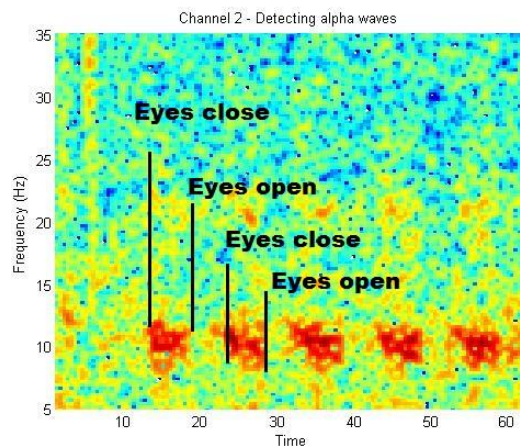


Figure 44: Spectrogram of alpha waves detection session

4.2. Steady State Evoke Potential experiment.

This experiment is set up based on the SSVEP phenomena, a common technique in BCI. When stimuli (i.e. flickering screen) are presented to the subject at a certain frequency, his brains wave will boost up at this frequency significantly. A 26-years old subject sits on a chair and stares at the stimulus LCD screen. The flickering screen will alternate between stimuli (showing flickering at 6.6Hz, 7.5Hz, 8.75Hz, and 10Hz) for 10s and non-stimuli for 10s. We record four trial in a run. The subject looks at one of these frequencies, 6.6Hz, 7.5Hz, 8.75Hz, and 10Hz, respectively for every trial. Electrodes placements and filters setting are the same as the previous experiment.



Figure 45: Illustration of a SSVEP test session

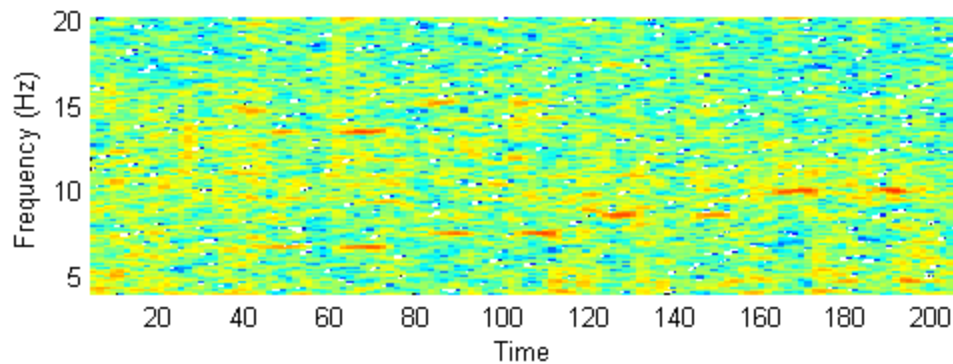


Figure 46: Spectrogram of an SSVEP session

Figure 46: Spectrogram of an SSVEP session is the spectrogram of the recording. The window for the spectrogram calculation is 2000; window overlap is 80%; sampling rate is 250 sample/second. Red marks in the spectrogram represent high power area where as blue marks denotes low power. Long red stripe at target frequencies (6.6Hz, 7.5Hz, 8.75Hz, and 10Hz) indicate Evoked Potential of the brain.

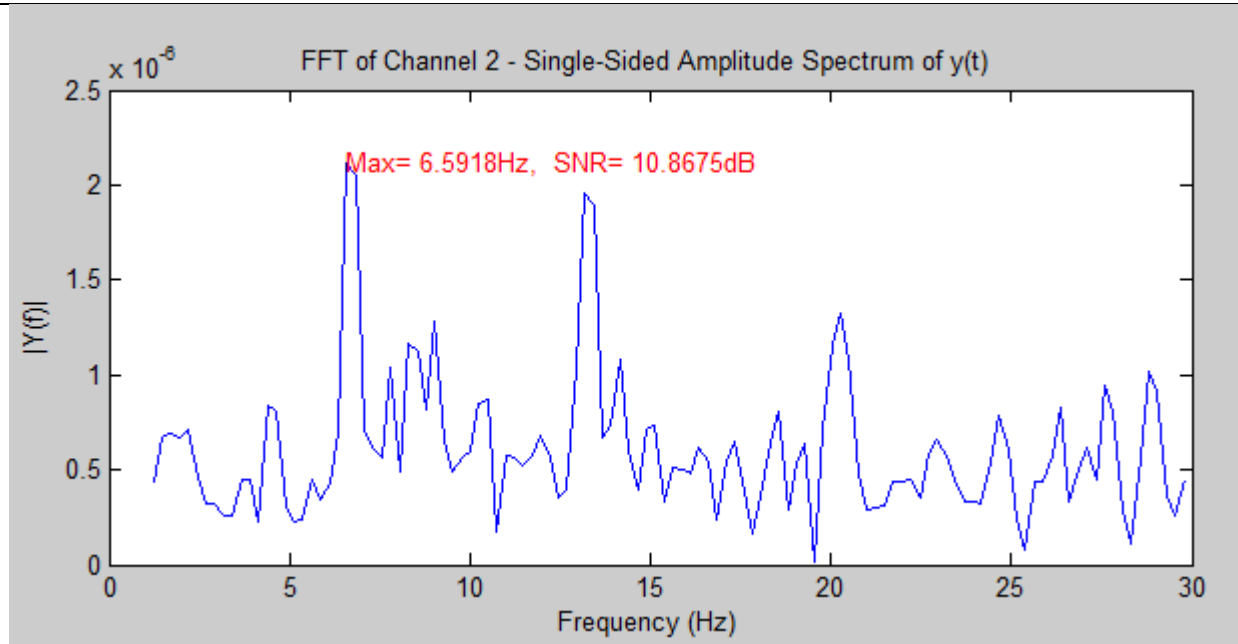


Figure 47: Amplitude Spectrum during stimulation

Figure 47 illustrates the FFT - Amplitude Spectrum of recorded brainwaves when the stimulation is at 6.6 Hz. The window length is 512 data point; zero padding to 1024 FFT point for better frequency resolution. The figure shows three majors peaks at the base frequency 6.6Hz, harmonics at 13.2Hz and 20.26Hz. During a good recording run, the mean Signal to Noise Ratio (SNR) at stimulation frequencies is 13 dB. A typical mean SNR is at 10dB. These conclusions are withdrawal from 5 runs at the different target frequency.

The SNR is calculated based on the equation below:

$$SNR = 20 \log \frac{Signal}{Noise} (db)$$

Whereas:

Signal is the amplitude at the stimulus frequency

Noise: is the average amplitude of frequency within the interest range from 5Hz to 25Hz

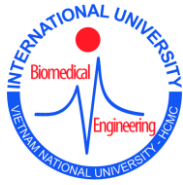
4.3. Compare with a traditional EEG system, Biosemi Active two.

4.3.1. General properties of an EEG system



Figure 48: Pictures of the five EEG acquisition systems[28]

General specifications comparison of an EEG system is listed in Table 5. The BioSemi's ActiveTwo is the traditional wired system with comprehensive electrode placement options with active electrode to reduce noise. Meanwhile, other novel systems have fewer electrodes location with the passive electrode. The B-Alert X10, EPOC, and HMS provides a custom headset for quick deployment which is an advantage. The WEEG leverages the headset of Biosemi's ActiveTwo which delivers standard 10-10 positions. At this stage of development, the reference location of



the WEEG is configurable. This flexibility offers developers options to optimize the reference location based upon on applications to reduce the noise.

Table 5: General specs comparision between systems [28]

System	Electrodes	Electrode locations	Electrode type	Scalp electrode connection to	Reference locations	Scalp area per electrode
WEEG	10	10–10 positions	Passive	Gel	Configurable	0.785 cm ²
ActiveTwo	64	10–10 positions	Active	Gel	Near Pz	Variable
B-Alert X10	9	F3, Fz, F4, C3, Cz, C4, P3, POz, P4	Passive	Gel-infused foam	External	1.77 cm ²
EPOC	14	AF3, AF4, F7, F3, F4, F8, FC5, FC6, T7, T8, P7, P8, O1, O2	Passive	Saline-infused felt	Mastoids and/or P3/P4	1.13 cm ²
HMS	9	F3, Fz, F4, C3, Cz, C4, P3, Pz, P4	Passive	Direct contact	P4, Fpz	3.80 cm ²

Table 6 shows major specifications of the signal acquisition module. Notably, the ActiveTwo is not a wireless module. This drawback is a major disadvantage of ActiveTwo compare to another system. On the other hand, a wire system enables ActiveTwo to transfer data at a much higher rate than other systems. The WEEG has a decent sampling rate at 500Hz compare to another mobile EEG module beside the ActiveTwo. In general, 250SPS is an adequate sampling frequency for an EEG acquisition system given the fact that most brain waves are lower than 125Hz Nyquist frequency. However, some experiments, such as Event Related Potential, require higher SPS, i.e. at 500Hz or more, to better detect the response brainwave.

The theoretical bandwidth of the WEEG is 0.3-200Hz. Bitrate is another crucial measurement of an EEG system. WEEG offers the best in class bitrate at 24 bit/channel, thanks to the new Analog to Digital Converter SoC. This advantage yields a higher signal resolution.

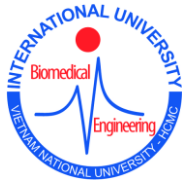


Table 6: General specs comparison between systems (continue)

System	Manufacturer	Sampling rate (Hz)	Bandwidth (Hz)	Wireless transmission	Bitrate	Trigger inputs
WEEG	BME-IU	500	0.3-200	Bluetooth	24 bits	Software
ActiveTwo	Biosemi	2048 or greater	User defined	None	24 bits	Parallel port
B-Alert X10	ABM	256	0.1–100	Bluetooth	16 bits	Parallel port
EPOC	Emotiv	128	0.2–43	Proprietary wireless	16 bits	Serial port
HMS	QUASAR	240	0.02–120	Proprietary wireless	16 bits	Single TTL

4.3.2. Compare Power Signal Densities (PSD) at alpha band

The PSD of the brain are measured by both the WEEG and Biosemi's ActiveTwo during eye closed for comparison. The protocol is slightly different compared to the previous experiment in 4.1. Electrodes of WEEG are placed the same as Biosemi's ActiveTwo set up. The recorded channel is at O2, the Common Mode Sense (CMS) and Driven Right Leg (DRL) of both system are at the same location near Pz.

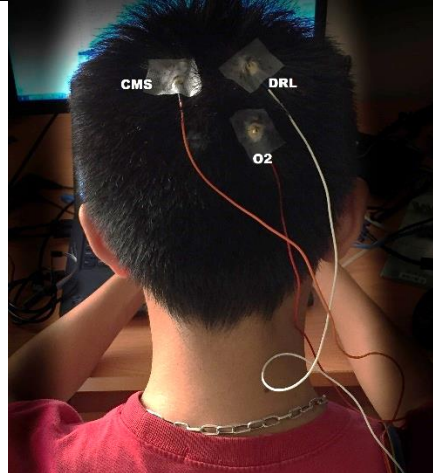


Figure 49: Electrode placement for comparison with Biosemi's ActiveTwo

Protocol: Two healthy subjects (all male, age 27 and 21) participated in the study. None had a history of diabetes, epilepsy, or any other chronic disease and were not taking regular medication. Subjects sit and relax on a chair with his eyes open for 45s. There are no stimuli in front of the subject's eyes. Second, the subject closes his eyes in 45s. This protocol is illustrated in Figure 50. One run consists of four trials. Gelled Electrodes are placed on the subject's scalp according to the Biosemi head cap. One channel is recorded at O2 location. One Ground (bias) electrode is placed on the DRL location of the cap. The differential pair is placed on the O2 and the CMS location. The sampling rate is 250 Hz for WEEG and 256Hz for ActiveTwo. EEG data was high-pass filtered at 0.3Hz to avoid DC drift and a low-pass filter at 43Hz for both systems.

Eye open 45s	Eyes close 45s	Eyes open 45s	Eyes close 45s	Eyes open 45s	...	Eyes close 45s	Eyes open 45s
--------------------	----------------------	---------------------	----------------------	---------------------	-----	----------------------	---------------------

Figure 50: Alpha waves comparison protocol



Vietnam National Universities – HCMC

International University

Biomedical Engineering Department

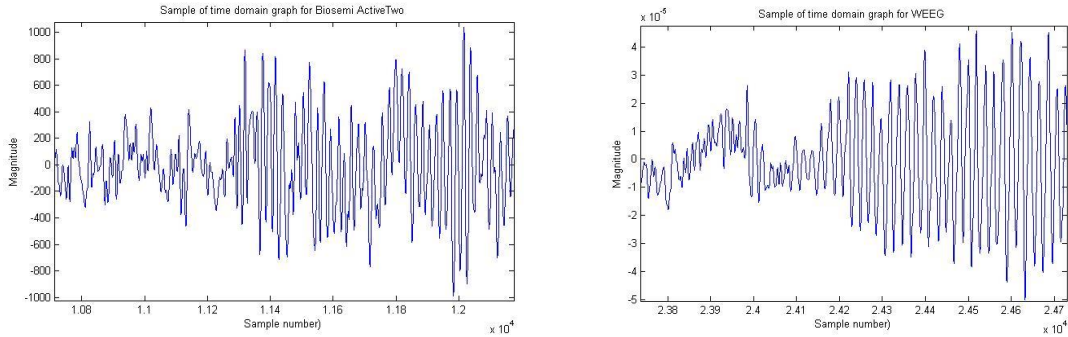


Figure 51 Sample of alpha waves in time domain from both systems (left: Biosemi ActiveTwo; right WEEG)

Using Welch's method to estimate power spectral density. 20 second of data during eye closed are analyzed. The data are divided into 8 segments with 50% overlap. Each section is windowed with a Hamming window. Four epochs are averaged out to get the final mean. The WEEG signal is normalized by matching the mean of WEEG signal with that of Biosemi ActiveTwo, the different is 149 db. The result is demonstrated in Figure 52.

The WEEG's signal in blue is similar to the Biosemi's signal in red. Power in the alpha band from 8 Hz to 13Hz is significantly higher than other frequencies. Both graphs roll off gradually from 15Hz toward 22Hz. Higher frequencies' powers are notably trending down steeply.

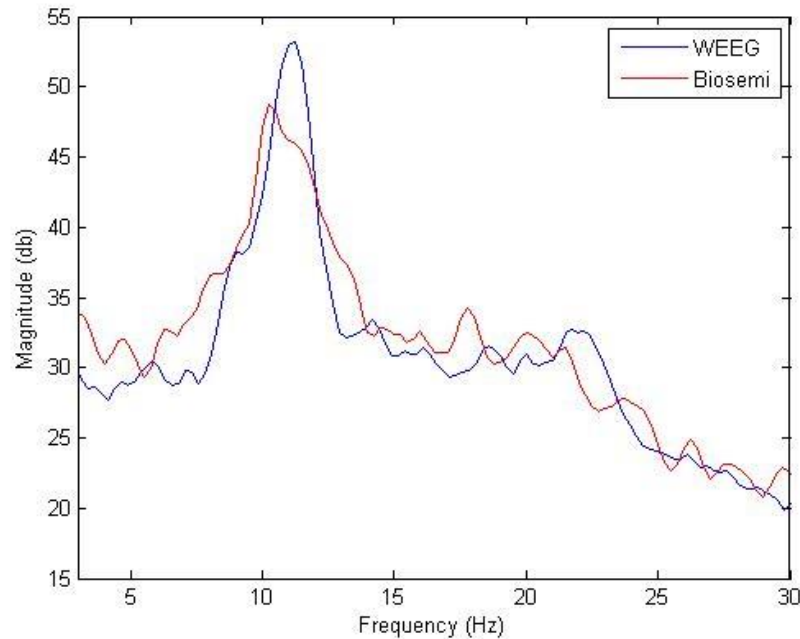


Figure 52: A comparison of power spectral densities (PSDs) for WEEG and Biosemi recordings for the “eyes closed.”

The normalized cross-correlation between two signal is very high at 0.98. The calculation for this value is described as below

```
% WEEG_PSD is the Power Spectral Densities from WEEG system
% Biosemi_PSD is the Power Spectral Densities from WEEG system
a = WEEG_PSD;
b = Biosemi_PSD;
% xcorr is the function to calculate the cross correlation between two
signals
c = xcorr(a,b);
%normalized the cross correlation
%maxCorrNor is the maximum correlation between a and b, normalized to
0-1 scale
maxCorrNor = max(c)/max(max(xcorr(a,a)),max(xcorr(b,b)));
```




4.3.3. Compare SSVEP PSD

The PSD of the brain are measured by both the WEEG and Biosemi's ActiveTwo during stimulation at a fixed frequency 6.6Hz.

Protocol: Two healthy subjects (all male, age 27 and 21) participated in the study. None had a history of diabetes, epilepsy, or any other chronic disease and were not taking regular medication. The subject sits on a chair and stares at the LCD screen. The flickering screen will alternate between stimuli (showing flickering at 6.6Hz) for 10s and non-stimuli (showing black screen) for 10s. Gelled electrodes are placed on the subject's scalp. One channel is recorded at O2 location. The DRL of the WEEG is put at the left mastoid, whereas the reference electrode is placed at Oz. The sampling rate is 250 Hz for WEEG and 256Hz for ActiveTwo. EEG data was high-pass filtered at 0.3Hz to avoid DC drift and a low-pass filter at 43Hz for both systems.

Data recorded by the Biosemi's ActiveTwo is normalized from 256 SPS to 250SPS for comparison. Ten seconds of measurement data from both systems were analyzed. The magnitude squared coherence spectrum were calculated by Matlab function mscohere.

According to the definition of the term, the magnitude-squared coherence estimate is a function of frequency with ranges between 0 and 1 that indicates how well x corresponds to y at each frequency. The magnitude-squared coherence is a function of the power spectral densities, $P_{xx}(f)$ and $P_{yy}(f)$, of x and y, and the cross power spectral density, $P_{xy}(f)$, of x and y.[29]

$$C_{xy}(f) = \frac{|P_{xy}(f)|^2}{|P_{xx}(f)| \cdot |P_{yy}(f)|}$$

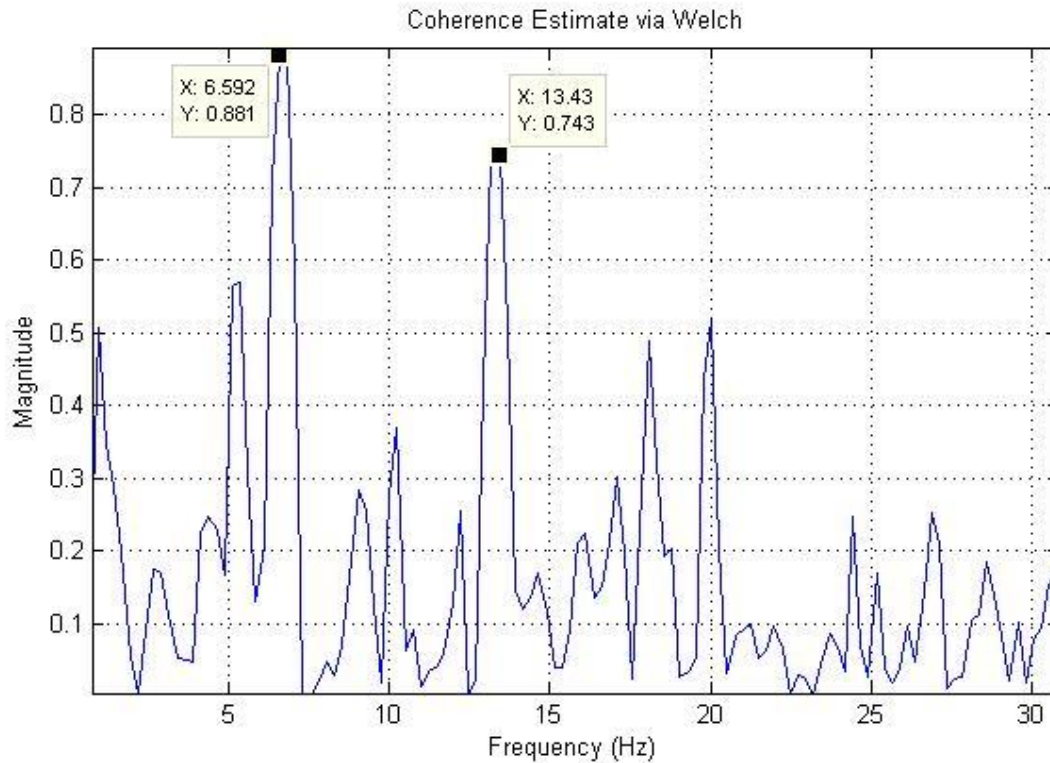


Figure 53: Coherence Estimation via Welch's method.

At target frequency and its harmonics, there is strong coherence (> 0.7) between data sets from both systems. It proves that the WEEG can produce meaningful data for the certain experiment.



5. Chapter 5

Conclusions and Future Work

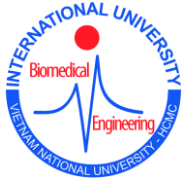
5.1. Conclusion

This project successfully delivers a wireless EEG recording devices for certain applications such as Brain-Computer Interface. The current prototype which is smaller than a credit card includes simultaneous eight channels logging at 24-bit resolution per channel, 500 SPS. Data are wirelessly transferred via Bluetooth to a computer for real-time signal processing. The ADC system employs a TI low-noise ADS1299, whereas the digital system powered by an ARM Cortex-M4 STM32F4. The concurrent signal processing task is handled by MATLAB on a tethered computer. The GUI on the computer presents different characteristics of the EEG signal in frequency domain and time domain. The system can capture prominent EEG signatures, i.e., alpha wave, evoked potential, which are usable for Brain Computer Interface application.

Extensive evaluation results point out that the signal quality of WEEG is comparable to “gold standard” devices. In some applications, the WEEG platform is a better choice because of its portability and simplicity. An SSVEP based BCI game, with an accuracy > 90%, is devised to prove the usefulness of the device. The coherence between signals produced by WEEG and signals produced by Biosemi ActiveTwo at target frequency is higher than 0.6. Considering these factors. This is a promising platform to develop a novel application. The WEEG can enable a low-cost EEG related device which was limited due to high component price.

5.2. Future work

Some area for improvements are left for future development. First, all processing tasks can be put to the microcontroller. The device becomes standalone system without a tethering computer. This is an achievable goal given the fact that the ARM Cortex processor can compute FFT quickly by it DSP instructions. The device may be used for a sleep study because Polysomnography records EEG, EOG, EMG and ECG. If the system is capable of recording EEG signal, the noisiest signal,



Vietnam National Universities – HCMC

International University

Biomedical Engineering Department

it can record other larger signals well. Another interesting application to explore is a brain-computer interface. Event-related potential such as P300 can be exploited to develop useful human-computer interaction. On the other hand, examining this system with dry electrodes for a better user experience is an interesting task.



References

- 1 Teplan, M.: 'Fundamentals of EEG measurement', MEASUREMENT SCIENCE REVIEW, 2002
- 2 Heraz, A., Razaki, R., and Frasson, C.: 'Using machine learning to predict learner emotional state from brainwaves', 'Book Using machine learning to predict learner emotional state from brainwaves', 2007
- 3 Chi, Y.M., Ng, P., Kang, E., Kang, J., Fang, J., and Cauwenberghs, G.: 'Wireless non-contact cardiac and neural monitoring'. Proc. Wireless Health 2010, San Diego, California, 2010
- 4 Kim, Y.S., Baek, H.J., Kim, J.S., Lee, H.B., Choi, J.M., and Park, K.S.: 'Helmet-based physiological signal monitoring system', European journal of applied physiology, 2009
- 5 Pai-Yuan, T., Weichih, H., Kuo, T.B.J., and Liang-Yu, S.: 'A portable device for real time drowsiness detection using novel active dry electrode system', 'Book A portable device for real time drowsiness detection using novel active dry electrode system', 2009
- 6 Emotiv: 'EMOTIV EPOC+ 14 Channel Mobile EEG', available online at <http://emotiv.com/product/emotiv-epoc-14-channel-mobile-eeeg/>, 2016
- 7 Badcock, N.A., Mousikou, P., Mahajan, Y., de Lissa, P., Thie, J., and McArthur, G.: 'Validation of the Emotiv EPOC® EEG gaming system for measuring research quality auditory ERPs', PeerJ, 2013
- 8 Robert, L., Martin, W., and David, E.: 'The Emotiv EPOC neuroheadset: an inexpensive method of controlling assistive technologies using facial expressions and thoughts?', Journal of Assistive Technologies, 2011
- 9 Nathan, V., Wu, J., Zong, C., Zou, Y., Dehzangi, O., Reagor, M., and Jafari, R.: 'A 16-channel bluetooth enabled wearable EEG platform with dry-contact electrodes for brain computer interface'. Proc. Proceedings of the 4th Conference on Wireless Health, Baltimore, Maryland, 2013
- 10 Lovelace, J.A., Witt, T.S., and Beyette, F.R.: 'Modular, bluetooth enabled, wireless electroencephalograph (EEG) platform' : 'Book Modular, bluetooth enabled, wireless electroencephalograph (EEG) platform', 2013
- 11 Joel Murphy, C.R.: 'OpenBCI Homepage', available online at: <http://openbci.com/>, 2016
- 12 PlasticsOne: 'DIN standard 42-802, the Touch Proof connector', available online at <http://www.plastics1.com/Touch-Proof-Connector.php>, 2016
- 13 TexasInstruments: '"Low-Noise, 8-Channel, 24-Bit Analog Front-End for Biopotential Measurements" ADS1299 Datasheet', 2012
- 14 STMicroelectronic: " ARM Cortex-M4 32b MCU+FPU, 210DMIPS, up to 1MB Flash/192+4KB RAM, USB OTG HS/FS, Ethernet, 17 TIMs, 3 ADCs, 15 comm. interfaces & camera ", ARM Cortex-M4 STM32F40x Datasheet, 2016
- 15 STMicroelectronic: "STM32F407 Block diagram", Available online at <http://www.mouser.it/images/microsites/stm32f4block.png>, 2016



-
- 16 Microchip: ‘MCP73831/2 Miniature Single-Cell, Fully Integrated Li-Ion, Li-Polymer Charge Management Controllers’, MCP7383x datasheet, Available online at: <http://www1.microchip.com/downloads/en/DeviceDoc/20001984g.pdf>, 2016
 - 17 TexasInstruments: ‘TPS6306x High Input Voltage, Buck-Boost Converter With 2-A Switch Current’, TPS6306x datasheet, Available online: <http://www.ti.com/lit/ds/symlink/tps63060.pdf>, 2016
 - 18 TexasInstruments: ‘200-mA Low-Noise, High-PSRR, Negative-Output, Low-Dropout Linear Regulators’, TPS723xx datasheet, Available online: <http://www.ti.com/lit/ds/symlink/tps723-q1.pdf>, 2012
 - 19 TexasInstruments: ‘User's Guide, EEG Front-End Performance Demonstration Kit’, 2016
 - 20 Andrews, R.: ‘Pros and cons of Unipolar/Bipolar power supplies for ADS1299’, https://e2e.ti.com/support/data_converters/precision_data_converters/f/73/t/272254, 2013
 - 21 Rorden, C.: ‘Open source EEG/ECG/EMG’, Chris Rorden's Neuropsychology Lab, 2014
 - 22 Atmel: ‘Half Duplex UART Using the USI Module’, Application Note, 2003
 - 23 Movassaghi, S., Abolhasan, M., Lipman, J., Smith, D., and Jamalipour, A.: ‘Wireless Body Area Networks: A Survey’, IEEE Communications Surveys & Tutorials, 2014
 - 24 Currey, M.: ‘HC-05’, HC-05 reference manual, Available online: http://www.martyncurrey.com/wp-content/uploads/2014/10/HC-05_zs-040_01_1200.jpg, 2016
 - 25 SIG, B.: ‘Basic Rate/Enhanced Data Rate (BR/EDR)’, 2016
 - 26 Veigl, C.: ‘BrainBay - Developer Manual’, 2011
 - 27 Ming, C., Xiaorong, G., Shangkai, G., and Dingfeng, X.: ‘Design and implementation of a brain-computer interface with high transfer rates’, IEEE Transactions on Biomedical Engineering, 2002
 - 28 David Hairston, W., Whitaker, K.W., Ries, A.J., Vettel, J.M., Cortney Bradford, J., Kerick, S.E., and McDowell, K.: ‘Usability of four commercially-oriented EEG systems’, Journal of neural engineering, 2014
 - 29 Mathworks: ‘Magnitude-squared coherence - MATLAB mscohere’, Available online: <http://www.mathworks.com/help/signal/ref/mscohere.html>, 2016
 - 30 Jonas Duun-Henriksen, T.W.K., David Looney, Mary Doreen Atkins, Jens Ahm Sørensen, Martin Rose, Danilo P. Mandic, Rasmus Elsborg Madsen, and Claus Bogh Juhl: ‘EEG Signal Quality of a Subcutaneous Recording System Compared to Standard Surface Electrodes’, Journal of Sensors, 2015



Appendix

MATLAB code

```
function varargout = SPCRS232(varargin)
% SPCRS232 M-file for SPCRS232.fig
%     SPCRS232, by itself, creates a new SPCRS232 or
raises the existing
%     singleton*.
%
%     H = SPCRS232 returns the handle to a new SPCRS232 or
the handle to
%     the existing singleton*.
%
%     SPCRS232('CALLBACK',hObject,eventData,handles,...)
calls the local
%     function named CALLBACK in SPCRS232.M with the given
input arguments.
%
%     SPCRS232('Property','Value',...) creates a new
SPCRS232 or raises the
%     existing singleton*. Starting from the left,
property value pairs are
%     applied to the GUI before SPCRS232_OpeningFcn gets
called. An
%     unrecognized property name or invalid value makes
property application
%     stop. All inputs are passed to SPCRS232_OpeningFcn
via varargin.
%
%     *See GUI Options on GUIDE's Tools menu. Choose "GUI
allows only one
%     instance to run (singleton)".
%
% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help
SPCRS232
```



Vietnam National Universities – HCMC

International University

Biomedical Engineering Department

```
% Last Modified by GUIDE v2.5 24-May-2016 13:34:06

% Begin initialization code - DO NOT EDIT
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
    'gui_Singleton',  gui_Singleton, ...
    'gui_OpeningFcn', @SPCRS232_OpeningFcn, ...
    'gui_OutputFcn',  @SPCRS232_OutputFcn, ...
    'gui_LayoutFcn',  [] , ...
    'gui_Callback',    []);
if nargin && ischar(varargin[30])
    gui_State.gui_Callback = str2func(varargin[30]);
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State,
varargin[31]);
else
    gui_mainfcn(gui_State, varargin{:});
end
% End initialization code - DO NOT EDIT

% --- Executes just before SPCRS232 is made visible.
function SPCRS232_OpeningFcn(hObject, ~, handles, varargin)
% This function has no output args, see OutputFcn.
% hObject    handle to figure
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles     structure with handles and user data (see
GUIDATA)
% varargin    command line arguments to SPCRS232 (see
VARARGIN)

% Choose default command line output for SPCRS232
handles.output = hObject;
delete(instrfindall);      % Reset Comport

%Initiate global parameter
```




```
global dataAll; dataAll = zeros(1, 1e7, 'int16'); %pre-
allocate a matrix to store session data
global data1; data1 = zeros(1, 1e6, 'single'); %pre-
allocate a matrix to store data for each channel
global data2; data2 = zeros(1, 1e6, 'single');
global data3; data3 = zeros(1, 1e6, 'single');
global data4; data4 = zeros(1, 1e6, 'single');
global data5; data5 = zeros(1, 1e6, 'single');
global data6; data6 = zeros(1, 1e6, 'single');
global data7; data7 = zeros(1, 1e6, 'single');
global data8; data8 = zeros(1, 1e6, 'single');

%Buffer
global linebuffSize; linebuffSize = 1000; %number of
sample in the plot
global t; t=1:linebuffSize; % x-axis of the
plot
global linebuffer_x
global linebuffer_y
global linebuffer_1; global linebuffer_2; global
linebuffer_3; global linebuffer_4; global linebuffer_5; global
linebuffer_6; global linebuffer_7; global linebuffer_8; global
linebuffer_9;

global h; global h1; global h2; global h3; global h4; global
h5; global h6; global h7; global h8; global h9;

linebuffer_x = nan(1, linebuffSize);
linebuffer_y = nan(1, linebuffSize);
%buffer 8 channel
linebuffer_1 = nan(1, linebuffSize);
linebuffer_2 = nan(1, linebuffSize);
linebuffer_3 = nan(1, linebuffSize);
linebuffer_4 = nan(1, linebuffSize);
linebuffer_5 = nan(1, linebuffSize);
linebuffer_6 = nan(1, linebuffSize);
linebuffer_7 = nan(1, linebuffSize);
linebuffer_8 = nan(1, linebuffSize);
linebuffer_9 = nan(1, linebuffSize);
```



Vietnam National Universities – HCMC

International University

Biomedical Engineering Department

```
t=1:linebuffSize; %x spacing

%
global bufferSize
bufferSize = 5000; % the interrupt of Matlab byteavailable
is planed to be 500, here we use 20 times the size of that
data as the size of the ringbuffer, unknown optimized size
global packageLength; packageLength =37; % Armbrain 2
header package size (2h+9x3data+4count)
global data; data = nan(bufferSize,1)'; %' for transpose
%global datastream; datastream = nan(bytesToRead,1); %pre
allocate for data
global ind
ind = 1; %buffer index
global variable;
variable =1;
%init data
global last; last = -1;
global first; first = 1;

global dataPointIndex; dataPointIndex = 0; %index to save
data of single channel

set(handles.ListBaudrate,'Value',12); % set default value
for serial baud rate
set(handles.ListPortname,'Value',4); % set default value
for serial baud rate
set(handles.popupmenuMultiplier,'Value',6); % set default
value for serial baud rate

global markerpoint %store marker index value
global markertext; %store marker note from gui
global markerstruct %store all marker note data
global markerindex
markertext = {}; %init this is a cell array
markerpoint = {};
markerindex = {};
```



```
%Filter initialize
global FIRfilter;
global IIRfilter;
FIRfilter = 0;           %Don't use filter until it's tick
on the GUI
IIRfilter = 0;
global TCP;
TCP =0;

if TCP == 1

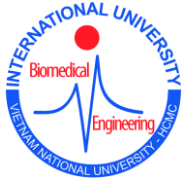
% Init tcpip object
handles.tcpipClient =
tcpip('10.8.122.147',55000,'NetworkRole','Client');
set(handles.tcpipClient,'InputBufferSize',7688);
set(handles.tcpipClient,'Timeout',5);

% Connect
fopen(handles.tcpipClient);
end
%SerialPort.Baudrate=str2double(getCurrentPopupString(handles.ListBaudrate));
% Update handles structure
guidata(hObject, handles);
save('handles.mat', 'handles');

% UIWAIT makes SPCRS232 wait for user response (see
UIRESUME)
% uiwait(handles.figure1);

% --- Outputs from this function are returned to the
command line.

function varargout = SPCRS232_OutputFcn(~, eventdata,
handles)
% varargout    cell array for returning output args (see
VARARGOUT);
% hObject      handle to figure
```



```
% eventdata reserved - to be defined in a future version
of MATLAB
% handles structure with handles and user data (see
GUIDATA)

% Get default command line output from handles structure
varargout[30] = handles.output;

% --- Executes on selection change in ListPortname.

function ListPortname_Callback(hObject, eventdata, handles)
% hObject handle to ListPortname (see GCBO)
% eventdata reserved - to be defined in a future version
of MATLAB
% handles structure with handles and user data (see
GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
ListPortname contents as cell array
% contents{get(hObject,'Value')} returns selected
item from ListPortname
% PortNumber=get(handles.ListPortname,'Value');
% Portname=['COM',num2str(PortNumber)];
%set(handles.TextNumber,'String',str);

% --- Executes during object creation, after setting all
properties.

function ListPortname_CreateFcn(hObject, eventdata,
handles)
% hObject handle to ListPortname (see GCBO)
% eventdata reserved - to be defined in a future version
of MATLAB
% handles empty - handles not created until after all
CreateFcns called

% Hint: popupmenu controls usually have a white background
on Windows.
```



```
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in ListBaudrate.
function ListBaudrate_Callback(hObject, eventdata, handles)
% hObject      handle to ListBaudrate (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      structure with handles and user data (see
GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
ListBaudrate contents as cell array
%           contents{get(hObject,'Value')} returns selected
item from ListBaudrate

% --- Executes during object creation, after setting all
properties.

function ListBaudrate_CreateFcn(hObject, eventdata,
handles)
% hObject      handle to ListBaudrate (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      empty - handles not created until after all
CreateFcns called

% Hint: popupmenu controls usually have a white background
on Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```



```
% --- Executes on selection change in ListDatabits.

function ListDatabits_Callback(hObject, eventdata, handles)
% hObject      handle to ListDatabits (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      structure with handles and user data (see
GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
ListDatabits contents as cell array
%           contents{get(hObject,'Value')} returns selected
item from ListDatabits

% --- Executes during object creation, after setting all
properties.

function ListDatabits_CreateFcn(hObject, eventdata,
handles)
% hObject      handle to ListDatabits (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      empty - handles not created until after all
CreateFcns called

% Hint: popupmenu controls usually have a white background
on Windows.
%           See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in ListParity.

function ListParity_Callback(hObject, eventdata, handles)
```



```
% hObject      handle to ListParity (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      structure with handles and user data (see
GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
ListParity contents as cell array
%           contents{get(hObject,'Value')} returns selected
item from ListParity

% --- Executes during object creation, after setting all
properties.

function ListParity_CreateFcn(hObject, eventdata, handles)
% hObject      handle to ListParity (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      empty - handles not created until after all
CreateFcns called

% Hint: popupmenu controls usually have a white background
on Windows.
%           See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in ListStopbits.
function ListStopbits_Callback(hObject, eventdata, handles)
% hObject      handle to ListStopbits (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      structure with handles and user data (see
GUIDATA)
```



```
% Hints: contents = cellstr(get(hObject,'String')) returns
ListStopbits contents as cell array
%         contents{get(hObject,'Value')} returns selected
item from ListStopbits

% --- Executes during object creation, after setting all
properties.
function ListStopbits_CreateFcn(hObject, eventdata,
handles)
% hObject    handle to ListStopbits (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: popupmenu controls usually have a white background
on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
guidata(hObject, handles);

% --- Executes on button press in ButtonConnect.
function ButtonConnect_Callback(hObject, eventdata,
handles)
% hObject    handle to ButtonConnect (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)Portname =
% getCurrentPopupSjhtring(handles.ListPortname);
% SerialPort=serial(getCurrentPopupString(ListPortname));

%-----Declare global variable-----
%
bytesToRead = 500; %500
```




```
serialInputBufferSize = 5000; %5000
global plotHandle1;
global plotHandle2;
global plotHandle3;
global plotHandle4;
global plotHandle5;
global plotHandle6;
global plotHandle7;

%Initiate handle
global single ADSmultiplier;
ADSmultiplier =
str2double(getCurrentPopupString(handles.popupmenuMultiplie
r));
%TCP
global TCP;
hold on;
plotHandle1= plot(handles.axes1,0,'-b','LineWidth',1);
%set(plotHandle1, 'DoubleBuffer', 'on' );
plotHandle2= plot(handles.axes28,0,'-b','LineWidth',1);
plotHandle3= plot(handles.axes25,0,'-b','LineWidth',1);
plotHandle4= plot(handles.axes29,0,'-b','LineWidth',1);
plotHandle5= plot(handles.axesFFT1,0,'-b','LineWidth',1);
plotHandle6= plot(handles.axesFFT2,0,'-b','LineWidth',1);
plotHandle7= plot(handles.axesFFT3,0,'-b','LineWidth',1);
%plotHandle1= plot(t,linebuffer_1);
%axis(handles.axes1,[0 1000 -3 3]); %set the range for x
and y of the axes1 plot
%-----Initiate Serial Port-----
-----%
% Declare variable for serial port
Databits = 8;
Parity = 'None';
Stopbits = 1;
a=get(handles.ButtonConnect,'String');
if strcmp(a,'Connect')
    Portname=getCurrentPopupString(handles.ListPortname);
    SerialPort=serial(Portname);
```



```
SerialPort.Baudrate=str2double(getCurrentPopupString(handles.  
s.ListBaudrate));  
    SerialPort.Databits=Databits;  
    SerialPort.Parity=Parity;  
    SerialPort.Stopbits=Stopbits;  
  
SerialPort.Baudrate=str2double(getCurrentPopupString(handles.  
s.ListBaudrate));  
    SerialPort.InputBufferSize=serialInputBufferSize;  
    SerialPort.BytesAvailableFcnCount = bytesToRead;  
    %SerialPort.BytesAvailableFcnMode = 'terminator';  
    SerialPort.BytesAvailableFcnMode = 'byte';  
  
%Channel=str2double(getCurrentPopupString(handles.ListChan)  
);  
    SerialPort.BytesAvailableFcn =  
{@localReadAndPlot,plotHandle1,bytesToRead};  
% Open serial port  
    try  
        handles.SerialPort = SerialPort; % s chính là  
handles.s  
        fopen(handles.SerialPort);  
        % hiện thì Disconnect  
        set(handles.ButtonConnect, 'String','Disconnect')  
        %%also open TCP port  
  
        %end tcpip  
        drawnow;  
    catch e  
        if(strcmp(handles.SerialPort.status,'open')==1)  
            fclose(handles.SerialPort);  
            if TCP == 1  
                %close TCP  
                % Close port  
                fclose(handles.tcpipClient);  
            end  
        end  
    end
```

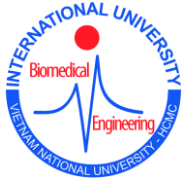


```
        errordlg(e.message); % xu ly loi ngoai le, neu
        khong co ngoai le xay ra thi se thuc hien catch
    end
    %Disconnect serial port
else
    set(handles.ButtonConnect, 'String','Connect')
    fclose(handles.SerialPort);
end
guidata(hObject, handles); % hObject la cai hien tai

function
localReadAndPlot(interfaceObject,~,figureHandle1,bytesToRead)

%% Declare global variables
global dataAll;
global plotHandle1;
global plotHandle2;
global plotHandle3;
global plotHandle4;
global plotHandle5;
global plotHandle6;
global plotHandle7;
    %rawdata package stream to computer
global datac; %number of data send: count
packets/frames
global data1; % 24 bit data from channel 1 to 8 raw data
extracted from the serial stream
global data2; %
global data3;
global data4;
global data5;
global data6;
global data7;
global data8;

global linebuffer_1;global linebuffer_2;global
linebuffer_3;global linebuffer_4;global linebuffer_5;global
```



```
linebuffer_6;global linebuffer_7;global linebuffer_8;global  
linebuffer_9;
```

```
global L;
```

```
% declare variables for buffer
```

```
global datastream
```

```
global last
```

```
global first
```

```
global bufferSize
```

```
global ind
```

```
global data
```

```
global packageLength
```

```
global linebuffSize
```

```
global linebuffer_x
```

```
global linebuffer_y
```

```
global linebuffer_1;global linebuffer_2;global
```

```
linebuffer_3;global linebuffer_4;global linebuffer_5;global
```

```
linebuffer_6;global linebuffer_7;global linebuffer_8;global
```

```
linebuffer_9;
```

```
global t;
```

```
global h; global h1;global h2;global h3;global h4;global
```

```
h5;global h6;global h7;global h8;global h9;
```

```
global dataPointIndex;
```

```
global ADSmultiplier;
```

```
%
```

```
load handles; %??? what is this for?
```

```
%% New filter parameter FIR/IIR
```

```
% Parameter for filter
```

```
persistent wDataCH1 ;persistent vDataCH1; % for  
filter
```

```
persistent lastFFT;
```

```
global dataplot; global Type; global af, global bf;
```

```
global FIRfilter;
```



```
global IIRfilter;
global TCP;
%initialize value if FIR filter is on
if (FIRfilter)
    global h_FIR;
    persistent wDataCH1_FIR, persistent wDataCH2_FIR,
persistent wDataCH3_FIR;
    %Init matrix value for the filter at the first run
    if isempty(wDataCH1_FIR)    %only initialize these
matrixes on the first run
        wDataCH1_FIR=zeros(1,1e3);
        wDataCH2_FIR=zeros(1,1e3);
        wDataCH3_FIR=zeros(1,1e3);
    end
end

%initialize value if IIR filter is on
if (IIRfilter)
    global af_IIR, global bf_IIR;
    persistent wDataCH1_IIR, persistent wDataCH2_IIR,
persistent wDataCH3_IIR;
    persistent vDataCH1_IIR, persistent vDataCH2_IIR,
persistent vDataCH3_IIR;
    %Init matrix value for the filter at the first run
    if isempty(wDataCH1_IIR)    %only initialize these
matrixes on the first run

        wDataCH1_IIR=zeros(1,1e3);    % initialize w and
v for channels
        wDataCH2_IIR=zeros(1,1e3);
        wDataCH3_IIR=zeros(1,1e3);
        vDataCH1_IIR=zeros(1,1e3);
        vDataCH2_IIR=zeros(1,1e3);
        vDataCH3_IIR=zeros(1,1e3);

%           wDataCH1_IIR=zeros(1,length(af_IIR));    %
initialize w CH1
%           wDataCH2_IIR=zeros(1,length(af_IIR));
%           wDataCH3_IIR=zeros(1,length(af_IIR));
```



```
%          vDataCH1_IIR=zeros(1,length(bf_IIR));
%          vDataCH2_IIR=zeros(1,length(bf_IIR));
%          vDataCH3_IIR=zeros(1,length(bf_IIR));

%
assignin('base','w_init_IIR',wDataCH1_IIR);
%          assignin('base','v_init_IIR',vDataCH1_IIR);
%          assignin('base','af_IIR_init',af_IIR);
%          disp('init IIR');

        end
    end

%% SNR calculation initiation

if ~exist('sumweight')    %initiate value in the 1st run
    persistent lastcommand1; lastcommand1 =0;
    persistent lastcommand2; lastcommand1 =0;
    persistent peakf;
    %harmonic to detect > these are peak value of the
interest F with idx =
    %find(f1>=5 & f1<=25); f has512 values
    peakf = [7, 8, 34; 10 , 11, 41; 15, 16, 15; 28 , 29,
30]; %index in the interestF matrix
    peakf = peakf + 21;
%offset index in the f matrix - 1024 NFFT
    %weight of different frequency in the harmonic
    persistent weight;
    weight = [1, 1, 1];
%same weight for all frequency in the harmonic
    sumweight = 0;
    %Threshold to decide a valid target
    persistent freqthreshold; freqthreshold = 2.2;
    %Index of the ringbuffer sumsnr1 and 2
    persistent idxsumsnr1; idxsumsnr1 = 1; persistent
idxsumsnr2; idxsumsnr2 = 1;
    persistent snrsum1; persistent snrsum2;
    snrqueue = 4;
% Define how many recent value to keep
```



```
snrsum1 = zeros(length(peakf(:,1)),snrqueue);
% 4x4 matrix to hold recent snrsum value everyrow is a
target freq
snrsum2 = zeros(length(peakf(:,1)),snrqueue);
% this hold last 4 freq snr sum.
for iw = 1:length(weight)
    sumweight = sumweight + weight(iw);
end
weight = weight/sumweight;
end
%% End of cell

%% Start reading data
% Read the desired number of data bytes
datastream = fread(interfaceObject,bytesToRead); % read
binary data from serial port

%put datastream in the buffer
for indDatastream = 1:bytesToRead %5 is the length of
the data stream
    last = mod(ind-1, bufferSize)+1; %calculate the
index to put data in. The mod function help to maintain the
ring buffer
    data(last) = datastream (indDatastream); %copy data
from datastream buffer to ringbuffer
    dataAll(1,ind) = datastream (indDatastream);
    % A(i,:) = rowVelec
    ind = ind +1; %index indicate
number of bytes were read
end

%condition to analyze data
% (last > packageLength) : make sure there are enough
data to analyze
% also prevent the half data package
% (last > packageLength) && first > last : full buffer,
last cross over
% the bufferlimit 1 round before the first
```



```
% (last > packageLength) && (last >
first+packageLength) : normal
% condition
if (last > packageLength) && (( first > last)||(last >
first+packageLength))
    %% process all data in the ring
    while (      (last > (first + packageLength)) ||
(first > last)      )      %condition to make sure the
available byte in the buffer is more than 1 package
        %look for new header

        % range of first: from 1 to bufferSize. That's
why we need to
        % -1 in the modular function, then +1 outside
to make sure that
        % we got first = bufferSize value after the mod
function

        if ((data(mod(first-1 +0, bufferSize)+1) ==
254) && (data(mod(first-1 + 1, bufferSize)+1) == 1) &&
(data(mod(first-1 + 2, bufferSize)+1) == 254)&&
(data(mod(first-1 + 3, bufferSize)+1) == 1)&&
(data(mod(first-1 + 4, bufferSize)+1) == 254)&&
(data(mod(first-1 + 5, bufferSize)+1) == 1) ) %fake header
            %frame header detected, start to analyze data
of a frame
            %output_x = [output_x ; data(mod(first-1 + 2,
bufferSize)+1)] ;          %2 in the index within a frame of
xput meaningful data after the header to output matrix
check t

%=====*****=====
=====

        % A data structure sample
        % Byte 1: Header 1
        % Byte 2: Header 2
        % Byte 3: Status 1
        % Byte 4: Status 2
        % Byte 5: Status 3
```




```
% Byte 6: Channel 1 Byte 1
% Byte 7: Channel 1 Byte 2
% Byte 8: Channel 1 Byte 3
% Byte 9: Channel 2 Byte 1
% Byte 10: Channel 2 Byte 2
% Byte 11: Channel 2 Byte 3
%
%
% Byte 33: Channel 8 Byte 3

%=====*****=====
=====

%plot channel 1: 5 6 7
%
%debug

dataPointIndex = dataPointIndex +1;
%index to the next location to save new data
%disp(dataPointIndex);
datatmp = data(mod(first-1 + 5 +4,
bufferSize)+1)*65536 + data(mod(first-1 + 6+4,
bufferSize)+1)*256+data(mod(first-1 + 7 +4, bufferSize)+1);
%combine seperate bytes of channel 1
datatmp = (datatmp*4.5/(2^23-1)-
4.5)/ADSmultiplier; %Convert to volt
linebuffer_1 =[linebuffer_1(2:end) datatmp];
%4 is the location of 16 in dataall. location of 254 is 0
data1(dataPointIndex)= datatmp;

%plot channel 2: 8 9 10
datatmp = data(mod(first-1 + 8 +4,
bufferSize)+1)*65536 + data(mod(first-1 + 9 +4,
bufferSize)+1)*256+data(mod(first-1 + 10+4, bufferSize)+1);
%combine seperate bytes of channel 1
datatmp = (datatmp*4.5/(2^23-1)-
4.5)/ADSmultiplier;
linebuffer_2 =[linebuffer_2(2:end) datatmp];
%4 is the location of 16 in dataall. location of 254 is 0
```



Vietnam National Universities – HCMC

International University

Biomedical Engineering Department

```
data2(dataPointIndex)= datatmp;

%plot channel 3: 11 12 13    -temperature
channel
    datatmp = data(mod(first-1 + 11 +4,
bufferSize)+1)*65536 + data(mod(first-1 + 12 +4,
bufferSize)+1)*256+data(mod(first-1 + 13 +4,
bufferSize)+1);    %combine sepearate bytes of channel 1
    datatmp = (datatmp*4.5/(2^23-1)-
4.5)/ADSmultiplier;
    linebuffer_3 =[linebuffer_3(2:end) datatmp];
%4 is the location of 16 in dataall. location of 254 is 0
    data3(dataPointIndex)= datatmp;

%plot channel 4: 14 15 16
    datatmp = data(mod(first-1 + 14 +4,
bufferSize)+1)*65536 + data(mod(first-1 + 15 +4,
bufferSize)+1)*256+data(mod(first-1 + 16 + 4,
bufferSize)+1);    %combine sepearate bytes of channel 1
    datatmp = (datatmp*4.5/(2^23-1)-
4.5)/ADSmultiplier;
    linebuffer_4 =[linebuffer_4(2:end) datatmp];
%4 is the location of 16 in dataall. location of 254 is 0
    data4(dataPointIndex) = datatmp;

%plot channel 5: 17 18 19
    datatmp = data(mod(first-1 + 17 +4,
bufferSize)+1)*65536 + data(mod(first-1 + 18+4,
bufferSize)+1)*256+data(mod(first-1 + 19+4, bufferSize)+1);
%combine sepearate bytes of channel 1
    datatmp = (datatmp*4.5/(2^23-1)-
4.5)/ADSmultiplier;
    linebuffer_5 =[linebuffer_5(2:end) datatmp];
%4 is the location of 16 in dataall. location of 254 is 0
    data5(dataPointIndex) = datatmp;

%plot channel 6: 20 21 22
```



```
        datatmp = data(mod(first-1 + 20 +4,
bufferSize)+1)*65536 + data(mod(first-1 + 21 +4,
bufferSize)+1)*256+data(mod(first-1 + 22+4, bufferSize)+1);
%combine seperate bytes of channel 1
        datatmp = (datatmp*4.5/(2^23-1)-
4.5)/ADSmultiplier;
        linebuffer_6 =[linebuffer_6(2:end) datatmp];
%4 is the location of 16 in dataall. location of 254 is 0
        data6(dataPointIndex) = datatmp;

        %plot channel 7: 23 24 25
        datatmp = data(mod(first-1 + 23+4,
bufferSize)+1)*65536 + data(mod(first-1 + 24+4,
bufferSize)+1)*256+data(mod(first-1 + 25+4, bufferSize)+1);
%combine seperate bytes of channel 1
        datatmp = (datatmp*4.5/(2^23-1)-
4.5)/ADSmultiplier;
        linebuffer_7 =[linebuffer_7(2:end) datatmp];
%4 is the location of 16 in dataall. location of 254 is 0
        data7(dataPointIndex) = datatmp;

        %plot channel 8: 26 27 28
        datatmp = data(mod(first-1 + 26+4,
bufferSize)+1)*65536 + data(mod(first-1 + 27+4,
bufferSize)+1)*256+data(mod(first-1 + 28+4, bufferSize)+1);
%combine seperate bytes of channel 1
        datatmp = (datatmp*4.5/(2^23-1)-
4.5)/ADSmultiplier;
        linebuffer_8 =[linebuffer_8(2:end) datatmp];
%4 is the location of 16 in dataall. location of 254 is 0
        data8(dataPointIndex) = datatmp;

        %output_y = [output_y ; data(mod(first-1 + 3,
bufferSize)+1)] ;
        linebuffer_9 =[linebuffer_9(2:end)
data(mod(first-1 + 32+ 4, bufferSize)+1)]; %32 is the
location of the last counter
```



```
        first = mod(first-1 +packageLength,
bufferSize)+1;                                %completed reading current
package, index to the next package

        else first = mod(first-1 +1, bufferSize)+1 ;

    end

% Filter section: every data of channel 1,2,3
%Filter FIR for all channels
if(FIRfilter)
    M0_FIR = length(h_FIR)-1;    %M0 order
    %=====filter channel
1=====
    [linebuffer_1(end),wDataCH1_FIR] =
FilterRealtimeFIR(linebuffer_1(end),h_FIR,M0_FIR,wDataCH1_F
IR);
    [linebuffer_2(end),wDataCH2_FIR] =
FilterRealtimeFIR(linebuffer_2(end),h_FIR,M0_FIR,wDataCH2_F
IR);
    [linebuffer_3(end),wDataCH3_FIR] =
FilterRealtimeFIR(linebuffer_3(end),h_FIR,M0_FIR,wDataCH3_F
IR);

    end
    %Filter IIR for all channels
    if(IIRfilter)
        M0_IIR=length(af_IIR)-1;
        L0_IIR=length(bf_IIR)-1;    %M0/L0 Order order
        %=====filter channel
1=====
    %
    assignin('base','vDataCH1_IIR_before',vDataCH1_IIR);
    %
    assignin('base','vDataCH1_IIR_before',wDataCH1_IIR);
        [linebuffer_1(end),wDataCH1_IIR,vDataCH1_IIR]=
FilterRealtimeIIR(M0_IIR,af_IIR,L0_IIR,bf_IIR,wDataCH1_IIR,
vDataCH1_IIR,linebuffer_1(end));
        [linebuffer_2(end),wDataCH2_IIR,vDataCH2_IIR]=
FilterRealtimeIIR(M0_IIR,af_IIR,L0_IIR,bf_IIR,wDataCH2_IIR,
vDataCH2_IIR,linebuffer_2(end));
```



```
[linebuffer_3(end),wDataCH3_IIR,vDataCH3_IIR]=
FilterRealtimeIIR(M0_IIR,af_IIR,L0_IIR,bf_IIR,wDataCH3_IIR,
vDataCH3_IIR,linebuffer_3(end));
end
% End filter section
%% FFT calculation
    %% Plot FFT _ SNR and calculation
    if(mod(dataPointIndex,128) == 0) %calculate FFT
every 128 samples
        datatestvalue1 = linebuffer_1((end-510):end);
%take exactly 511 last values
        Fs = 250; % Sampling frequency
        T = 1/Fs; % Sample time
        L = length(datatestvalue1); % Length of signal
        t = (0:L-1)*T; % Time vector
        %Multiply NFFT by 2 to get 1024 FFT datapoint,
512 point is from the data,
        %zeropadded the rest
        NFFT = 2^nextpow2(L*2); % Next power of 2 from
length of y
        %y1
        Y1 = fft(datatestvalue1,NFFT)/L;
        f1 = Fs/2* linspace(0,1,NFFT/2+1);
        y1=2*abs(Y1(1:NFFT/2+1));
        % Plot single-sided amplitude spectrum.
        idx = find(f1>=5 & f1<=25);
        interestY1 = y1(idx); %f value from 5 to 40Hz
        interestF1 = f1(idx);
        %% Calculate SNR channel 1
        meanY1 = mean(interestY1);
        for targeti = 1:length(peakf(:,1))
%run through peakf matrix
            snrsumtmp1 = 0; %reset for
this loop
                for peakfidx = 1: length(peakf(1,:))
                    snrsumtmp1 = snrsumtmp1 +
y1(peakf(targeti,peakfidx)) * weight(peakfidx);
                end
            end
        end
    end
```



```
snrsum1(targeti,idxsumsnr1) =
snrsumtmp1/meanY1;
    end
    %Reset idx for the ring buffer
    if idxsumsnr1 >= 4
        idxsumsnr1 = 1;
    else
        idxsumsnr1 = idxsumsnr1+1;
    end
    % plot channel 1
    %strtitle1 = ['6.6:
',num2str(snrsum1(1,idxsumsnr1)),'7.5:
',num2str(snrsum1(2,idxsumsnr1)),'8.75:
',num2str(snrsum1(3,idxsumsnr1)),'10:
',num2str(snrsum1(4,idxsumsnr1))];
    %         subplot(2,1,1);
    %         plot(interestF1,interestY1,'-x');

    for snrsum1ridx = 1 : length(snrsum1(:,1)) %go
through all the column
        if snrsum1(snrsum1ridx,:) > freqthreshold
            if (snrsum1ridx ~= lastcommand1)
                %disp('true');
                strtitle1 =
['Target',num2str(snrsum1ridx)];
                %Send TCP command
                % Send data
                %fwrite(handles.tcpipClient,
num2str(snrsum1ridx), 'char');
                lastcommand1 = snrsum1ridx;
            end
            break; %quite loop if frequency if
found

        else
            strtitle1 = ['wait_C1'];
            lastcommand1 = 0;
        end
    end
end
```



```
%text(6,0,strtitle1,'HorizontalAlignment','left','Parent',h
andles.axesFFT1);
%           %           disp(tc);

%           title(strtitle1);
%           xlabel('Frequency (Hz)')
%           ylabel('|Y(f)|')

%           [sorted1,I1] = sort(interestY1,'descend');
%sort output Y
%           [r1,c1] = ind2sub(size(interestY1),I1(1:5));
%//Change 10 to any other required value
%
%           xmax1 = interestF1(c1(1));
%           ymax1 = interestY1(c1(1));

%end %if mode 128
%% FFT channel 2
%           %if(mod(dataPointIndex,128) == 0) %calculate
FFT every 128 samples
%           datatestvalue2 = linebuffer_2((end-510):end);
%take exactly 511 last values
%           Fs = 250; % Sampling frequency
%           T = 1/Fs; % Sample time
%           L = length(datatestvalue2); % Length of signal
%           t = (0:L-1)*T; % Time vector
%           %Multiply NFFT by 2 to get 1024 FFT datapoint,
512 point is from the data,
%           %zeropadded the rest
%           NFFT = 2^nextpow2(L*2); % Next power of 2 from
length of y
%           %y1
%           Y2 = fft(datatestvalue2,NFFT)/L;
%           f2 = Fs/2*linspace(0,1,NFFT/2+1);
%           y2=2*abs(Y2(1:NFFT/2+1));
%           % Plot single-sided amplitude spectrum.
%           idx = find(f2>=5 & f2<=25);
```



```
interestY2 = y2(idx); %f value from 5 to 40Hz
interestF2 = f2(idx);
%% Calculate SNR channel 1
meanY2 = mean(interestY2);
for targeti = 1:length(peakf(:,1))
%run through peakf matrix
    snrsumtmp2 = 0; %reset for
this loop
    for peakfidx = 1: length(peakf(1,:))
        snrsumtmp2 = snrsumtmp2 +
y2(peakf(targeti,peakfidx)) * weight(peakfidx);
    end
    snrsum2(targeti,idxsumsnr2) =
snrsumtmp2/meanY2;
end
%Reset idx for the ring buffer
if idxsumsnr2 >= 4
    idxsumsnr2 = 1;
else
    idxsumsnr2 = idxsumsnr2+1;
end
% plot channel 1
%strtitle1 = ['6.6:
',num2str(snrsum1(1,idxsumsnr1)),'7.5:
',num2str(snrsum1(2,idxsumsnr1)),'8.75:
',num2str(snrsum1(3,idxsumsnr1)),'10:
',num2str(snrsum1(4,idxsumsnr1))];
%
    subplot(2,1,1);
%
    plot(interestF1,interestY1,'-x');

    for snrsum2ridx = 1 : length(snrsum2(:,1)) %go
through all the column
        if snrsum2(snrsum2ridx,:) > freqthreshold
            if (snrsum2ridx ~= lastcommand2)
%only update if new command is present
                %disp('true');
                strtitle2 =
['Target',num2str(snrsum2ridx)];
                %Send TCP command
```




```
        if TCP == 1
            % Send data
            fwrite(handles.tcpipClient,
num2str(snrsum2ridx), 'char');
            end
            lastcommand2 = snrsum2ridx;
            end

            break; %quite loop if frequency if
found
            else
                strtitle2 = ['wait_C2'];
                lastcommand2 = 0;
            end
        end
    end

    %text(6,0,strtitle1,'HorizontalAlignment','left','Parent',h
andles.axesFFT1);
    %      %      disp(tc);

    %      title(strtitle1);
    %      xlabel('Frequency (Hz)')
    %      ylabel('|Y(f)|')

    %      [sorted1,I1] = sort(interestY1,'descend');
    %sort ouput Y
    %      [r1,c1] = ind2sub(size(interestY1),I1(1:5));
    %//Change 10 to any other required value
    %
    %      xmax1 = interestF1(c1(1));
    %      ymax1 = interestY1(c1(1));

    %end %if mode 128
    end %end while loop

    %Move plot to this plot to lessen the update rate
    set(plotHandle1,'Ydata',linebuffer_1);
    set(plotHandle2,'Ydata',linebuffer_2);
```



Vietnam National Universities – HCMC

International University

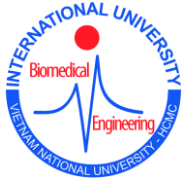
Biomedical Engineering Department

```
set(plotHandle3, 'Ydata', linebuffer_3);
set(plotHandle4, 'Ydata', linebuffer_9);
%Channel 1
set(plotHandle5, 'Xdata', interestF1);
set(plotHandle5, 'Ydata', interestY1);
set(handles.textFFT1, 'String', strtitle1,
'ForegroundColor', 'r');
%Channel 2
set(plotHandle6, 'Xdata', interestF2);
set(plotHandle6, 'Ydata', interestY2);
set(handles.textFFT2, 'String', strtitle2,
'ForegroundColor', 'r');
%% Plot FFT_peak
%Plot every second FFT of 3 channels; FFT value of last 512
value (2 seconds)
%
%      %if(mod(dataPointIndex,128) == 0) %calculate FFT
every 128 samples
%      %plot FFT
%      datatest = linebuffer_2(1:510);
%      Fs =
str2double(getCurrentPopupString(handles.PopSPS)); %
Sampling frequency
%      T = 1/Fs; % Sample time
%      L = length(datatest); % Length of signal
%      t = (0:L-1)*T; % Time vector
%
%      NFFT = 2^nextpow2(L); % Next power of 2 from
length of y
%      Y = fft(datatest,NFFT)/L;
%      f = Fs/2*linspace(0,1,NFFT/2+1);
%      % Plot single-sided amplitude spectrum.
%      y=2*abs(Y(1:NFFT/2+1));
%      %y=smooth(y);
%
%      %Plotting
%      interestY = y(6:82); %f value from 5 to 40Hz
%      interestF = f(6:82);
%      %plotfft
```



```
% set(plotHandle5,'Xdata',interestF);
% set(plotHandle5,'Ydata',interestY);

%      %      plot(interestF,interestY);
%      %      title('FFT of 25Hz Stimulus - Single-Sided
Amplitude Spectrum of y(t) - smoothed')
%      %      xlabel('Frequency (Hz)')
%      %      ylabel('|Y(f)|')
%      %      %xlim([5 40]);
%
%
%      % Detect peak
%      %
%      indexmax = find(max(interestY) == interestY);
%find index of peak y
%      xmax = interestF(indexmax);
%      ymax = interestY(indexmax);
%
%      %annotate
%
%      %%SNR
%      meanY = mean(interestY);
%      snr = ymax/meanY;
%
%      %snr = 20 * log10((ymax/meanY)^2);
%      strmax = ['Max= ',num2str(xmax),' SNR=
',num2str(snr)];
%      textcolor='rbkymg';
%      %change color based on algo decision
%      if ((snr) > 1.06) & ( (xmax) >9 ) & (
(xmax) <11))
%          tc = 1; %assign red if it is a detection
%          %beep;
%      else
%          tc = 2;
%      end
%      %
text(xmax,ymax,strmax,'HorizontalAlignment','left','color',
textcolor(tc),'Parent',handles.axesFFT1);
```



```
%          %          disp(tc);
%          %          set(handles.textFFT1, 'String', strmax,
'ForegroundColor',textcolor(tc));
%
%          %end          %end of if FFT no more
% % next channel FFT
% %
% %   PREFORMATTED
% %   TEXT
% %
%          %if(mod(dataPointIndex,128) == 32) %calculate FFT
every 128 samples
%          %plot FFT
%          %          datatest = linebuffer_1(1:510);
%          %          Fs =
str2double(getCurrentPopupString(handles.PopSPS)); %
Sampling frequency
%          %          T = 1/Fs; % Sample time
%          %          L = length(datatest); % Length of signal
%          %          t = (0:L-1)*T; % Time vector
%
%          %          NFFT = 2^nextpow2(L); % Next power of 2 from
length of y
%          %          Y = fft(datatest,NFFT)/L;
%          %          f = Fs/2*linspace(0,1,NFFT/2+1);
%          %          % Plot single-sided amplitude spectrum.
%          %          y=2*abs(Y(1:NFFT/2+1));
%          %          %y=smooth(y);
%
%          %          %Plotting
%          %          interestY = y(6:82); %f value from 5 to 40Hz
%          %          interestF = f(6:82);
% %          %          %plotfft
%          %          set(plotHandle6,'Xdata',interestF);
%          %          set(plotHandle6,'Ydata',interestY);
%
%          %          %          plot(interestF,interestY);
%          %          %          title('FFT of 25Hz Stimulus - Single-Sided
Amplitude Spectrum of y(t) - smoothed')
```



```
%           %           xlabel('Frequency (Hz)')
%           %           ylabel('|Y(f)|')
%           %           %xlim([5 40]);
%
%
%           % Detect peak
%           %
%           indexmax = find(max(interestY) == interestY);
%find index of peak y
%           xmax = interestF(indexmax);
%           ymax = interestY(indexmax);
%
%           %annotate
%
%           %%SNR
%           meanY = mean(interestY);
%           snr = ymax/meanY;
%
%           %snr = 20 * log10((ymax/meanY)^2);
%           strmax = ['Max= ',num2str(xmax),' SNR=
',num2str(snr)];
%           textcolor='rbkymg';
%           %change color based on algo decision
%           if ((snr) > 1.06) & ( (xmax) >9 ) & (
(xmax) <11))
%           tc = 1; %assign red if it is a detection
%           %beep;
%           else
%           tc = 2;
%           end
%           %
text(xmax,ymax,strmax,'HorizontalAlignment','left','color',
textcolor(tc),'Parent',handles.axesFFT1);
%           %           disp(tc);
%           set(handles.textFFT2, 'String', strmax,
'ForegroundColor',textcolor(tc));
%
%           %end
%           %frequency update
```



```
%      %drawnow;

end

% --- Executes on button press in ButttonExit.
function ButttonExit_Callback(hObject, eventdata, handles)
% hObject      handle to ButttonExit (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      structure with handles and user data (see
GUIDATA)

global datac;      %number of data send: count
packets/frames
global data1;      % 24 bit data from channel 1 to 8 raw data
extracted from the serial stream
global data2;      %
global data3;
global data4;
global data5;
global data6;
global data7;
global data8;

global  dataPointIndex;

assignin('base','data1',data1(1:dataPointIndex));
assignin('base','data2',data2(1:dataPointIndex));
assignin('base','data3',data3(1:dataPointIndex));
assignin('base','data4',data4(1:dataPointIndex));
assignin('base','data5',data5(1:dataPointIndex));
assignin('base','data6',data6(1:dataPointIndex));
assignin('base','data7',data7(1:dataPointIndex));
assignin('base','data8',data8(1:dataPointIndex));
%% marker ouput
global markertext;
global markerindex;
%global markerstruct %store all marker note data
```



```
assignin('base','MarkerText',markertext);
assignin('base','MarkerIndex',markerindex);

stiProtocol =
struct('Latency',markerindex,'Note',markertext);
assignin('base','MarkerStruct',stiProtocol);

close all;

%%%%%%%%%%%%% SUPPORT FUNCTION [1]
function str = getCurrentPopupString(hh)
% getCurrentPopupString returns the currently selected
string in the popupmenu with handle hh

% could test input here
if ~ishandle(hh) || strcmp(get(hh,'Type'),'popupmenu')
    error('getCurrentPopupString needs a handle to a
popupmenu as input')
end

% get the string - do it the readable way
list = get(hh,'String');
val = get(hh,'Value');
if iscell(list)
    str = list{val};
else
    str = list(val,:);
end

%--- Executes on button press in ButtonSave.
function ButtonSave_Callback(hObject, eventdata, handles)
% hObject      handle to ButtonSave (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      structure with handles and user data (see
GUIDATA)
```



```
% evalin('base','dataall');
load handles;
global dataall;
uisave('dataall.mat','dataall');

% --- Executes on selection change in ListChan.
function ListChan_Callback(hObject, eventdata, handles)
% hObject      handle to ListChan (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      structure with handles and user data (see
GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
ListChan contents as cell array
%           contents{get(hObject,'Value')} returns selected
item from ListChan

% --- Executes during object creation, after setting all
properties.
function ListChan_CreateFcn(hObject, eventdata, handles)
% hObject      handle to ListChan (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      empty - handles not created until after all
CreateFcns called

% Hint: popupmenu controls usually have a white background
on Windows.
%           See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in PopSPS.
```




```
function PopSPS_Callback(hObject, eventdata, handles)
% hObject      handle to PopSPS (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      structure with handles and user data (see
GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
PopSPS contents as cell array
%           contents{get(hObject,'Value')} returns selected
item from PopSPS

% --- Executes during object creation, after setting all
properties.
function PopSPS_CreateFcn(hObject, eventdata, handles)
% hObject      handle to PopSPS (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      empty - handles not created until after all
CreateFcns called

% Hint: popupmenu controls usually have a white background
on Windows.
%           See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUiControlBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbutton5.
function pushbutton5_Callback(hObject, eventdata, handles)
% hObject      handle to pushbutton5 (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      structure with handles and user data (see
GUIDATA)
```



```
% --- Executes on selection change in PopFilType1.
function PopFilType1_Callback(hObject, eventdata, handles)
% hObject      handle to PopFilType1 (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      structure with handles and user data (see
GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
PopFilType1 contents as cell array
%           contents{get(hObject,'Value')} returns selected
item from PopFilType1

% --- Executes during object creation, after setting all
properties.
function PopFilType1_CreateFcn(hObject, eventdata, handles)
% hObject      handle to PopFilType1 (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      empty - handles not created until after all
CreateFcns called

% Hint: popupmenu controls usually have a white background
on Windows.
%           See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function TextFcl_Callback(hObject, eventdata, handles)
% hObject      handle to TextFcl (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
```



```
% handles      structure with handles and user data (see
GUIDATA)

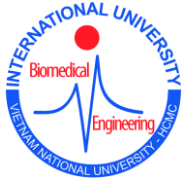
% Hints: get(hObject,'String') returns contents of TextFc1
as text
%      str2double(get(hObject,'String')) returns contents
of TextFc1 as a double

% --- Executes during object creation, after setting all
properties.
function TextFc1_CreateFcn(hObject, eventdata, handles)
% hObject      handle to TextFc1 (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%      See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in PopFilType2.
function PopFilType2_Callback(hObject, eventdata, handles)
% hObject      handle to PopFilType2 (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      structure with handles and user data (see
GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
PopFilType2 contents as cell array
%      contents{get(hObject,'Value')} returns selected
item from PopFilType2
```



```
% --- Executes during object creation, after setting all
properties.
function PopFilType2_CreateFcn(hObject, eventdata, handles)
% hObject    handle to PopFilType2 (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called

% Hint: popupmenu controls usually have a white background
on Windows.
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function TextOrder_Callback(hObject, eventdata, handles)
% hObject    handle to TextOrder (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of
TextOrder as text
%         str2double(get(hObject,'String')) returns contents
of TextOrder as a double

% --- Executes during object creation, after setting all
properties.
function TextOrder_CreateFcn(hObject, eventdata, handles)
% hObject    handle to TextOrder (see GCBO)
```



```
% eventdata reserved - to be defined in a future version
of MATLAB
% handles empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function TextFc2_Callback(hObject, eventdata, handles)
% hObject handle to TextFc2 (see GCBO)
% eventdata reserved - to be defined in a future version
of MATLAB
% handles structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of TextFc2
as text
% str2double(get(hObject,'String')) returns contents
of TextFc2 as a double

% --- Executes during object creation, after setting all
properties.
function TextFc2_CreateFcn(hObject, eventdata, handles)
% hObject handle to TextFc2 (see GCBO)
% eventdata reserved - to be defined in a future version
of MATLAB
% handles empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
% See ISPC and COMPUTER.
```



```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in cboxIIRfilter.
function cboxIIRfilter_Callback(hObject, eventdata,
handles)
% hObject      handle to cboxIIRfilter (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      structure with handles and user data (see
GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
cboxIIRfilter
global IIRfilter;
global af_IIR, global bf_IIR;

IIRfilter = get(handles.cboxIIRfilter, 'Value'); %update
IIR filter status
    if(IIRfilter)                                %generate new parameter
        drawnow;
        Order =
str2double(get(handles.editIIRorder, 'String'));

fs=str2double(getCurrentPopupString(handles.PopSPS));
    fc = str2double(get(handles.editFciir, 'String'));
    [bf_IIR,af_IIR]=newbutter(Order,2*fc/fs,'high');

    end

% --- Executes on button press in cboxFIRfilter.
function cboxFIRfilter_Callback(hObject, eventdata,
handles)
% hObject      handle to cboxFIRfilter (see GCBO)
```



```
% eventdata reserved - to be defined in a future version
of MATLAB
% handles structure with handles and user data (see
GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
cboxFIRfilter
global FIRfilter; %global value to indicate FIR
filter state
global h_FIR; %FIR rely only on current and
previous inputs

FIRfilter = get(handles.cboxFIRfilter, 'Value'); %update
IIR filter status
if(FIRfilter) %generate new parameter
    drawnow;
    Order=str2double(get(handles.editFIRorder, 'String'));
    wHam = hamming(Order+1);
    fs=str2double(getCurrentPopupString(handles.PopSPS));
    fc = str2double(get(handles.editFcfir, 'String'));
    h_FIR = fir1(Order, 2*fc/fs, wHam); %create FIR
    lowpass filter based on given values
end

% --- Executes on button press in cboxCustomFilter.
function cboxCustomFilter_Callback(hObject, eventdata,
handles)
% hObject handle to cboxCustomFilter (see GCBO)
% eventdata reserved - to be defined in a future version
of MATLAB
% handles structure with handles and user data (see
GUIDATA)

% Hint: get(hObject,'Value') returns toggle state of
cboxCustomFilter

% --- Executes on selection change in popupmenuMultiplier.
```



```
function popupmenuMultiplier_Callback(hObject, eventdata, handles)
% hObject      handle to popupmenuMultiplier (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      structure with handles and user data (see
GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
popupmenuMultiplier contents as cell array
%           contents{get(hObject,'Value')} returns selected
item from popupmenuMultiplier

% --- Executes during object creation, after setting all
properties.
function popupmenuMultiplier_CreateFcn(hObject, eventdata, handles)
% hObject      handle to popupmenuMultiplier (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      empty - handles not created until after all
CreateFcns called

% Hint: popupmenu controls usually have a white background
on Windows.
%           See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on selection change in popupmenuMultiplier.
function popupmenu15_Callback(hObject, eventdata, handles)
% hObject      handle to popupmenuMultiplier (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
```




```
% handles      structure with handles and user data (see
GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
popupmenuMultiplier contents as cell array
%      contents{get(hObject,'Value')} returns selected
item from popupmenuMultiplier

% --- Executes during object creation, after setting all
properties.
function popupmenu15_CreateFcn(hObject, eventdata, handles)
% hObject      handle to popupmenuMultiplier (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      empty - handles not created until after all
CreateFcns called

% Hint: popupmenu controls usually have a white background
on Windows.
%      See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

% --- Executes on button press in pushbuttonMarker.
function pushbuttonMarker_Callback(hObject, eventdata,
handles)
% hObject      handle to pushbuttonMarker (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      structure with handles and user data (see
GUIDATA)
global markertext;
global markerindex;
global dataPointIndex;
%getCurrentPopupString(handles.popupMark)
```



```
markerindex = [markerindex, {dataPointIndex}];  
markertext = [markertext,  
{getCurrentPopupString(handles.popupMark)}];
```

```
function editMarker_Callback(hObject, eventdata, handles)  
% hObject      handle to editMarker (see GCBO)  
% eventdata    reserved - to be defined in a future version  
of MATLAB  
% handles      structure with handles and user data (see  
GUIDATA)  
  
% Hints: get(hObject,'String') returns contents of  
editMarker as text  
%          str2double(get(hObject,'String')) returns contents  
of editMarker as a double  
  
% --- Executes during object creation, after setting all  
properties.  
function editMarker_CreateFcn(hObject, eventdata, handles)  
% hObject      handle to editMarker (see GCBO)  
% eventdata    reserved - to be defined in a future version  
of MATLAB  
% handles      empty - handles not created until after all  
CreateFcns called  
  
% Hint: edit controls usually have a white background on  
Windows.  
%          See ISPC and COMPUTER.  
if ispc && isequal(get(hObject,'BackgroundColor'),  
get(0,'defaultUicontrolBackgroundColor'))  
    set(hObject,'BackgroundColor','white');  
end  
%test
```



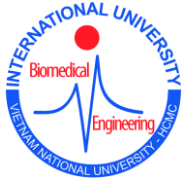
```
% --- Executes on selection change in popupMark.
function popupMark_Callback(hObject, eventdata, handles)
% hObject      handle to popupMark (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      structure with handles and user data (see
GUIDATA)

% Hints: contents = cellstr(get(hObject,'String')) returns
popupMark contents as cell array
%           contents{get(hObject,'Value')} returns selected
item from popupMark

% --- Executes during object creation, after setting all
properties.
function popupMark_CreateFcn(hObject, eventdata, handles)
% hObject      handle to popupMark (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      empty - handles not created until after all
CreateFcns called

% Hint: popupmenu controls usually have a white background
on Windows.
%           See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function editFciir_Callback(hObject, eventdata, handles)
% hObject      handle to editFciir (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
```



```
% handles      structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of
editFciir as text
%          str2double(get(hObject,'String')) returns contents
of editFciir as a double

% --- Executes during object creation, after setting all
properties.
function editFciir_CreateFcn(hObject, eventdata, handles)
% hObject      handle to editFciir (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
%          See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function editFcfir_Callback(hObject, eventdata, handles)
% hObject      handle to editFcfir (see GCBO)
% eventdata    reserved - to be defined in a future version
of MATLAB
% handles      structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of
editFcfir as text
%          str2double(get(hObject,'String')) returns contents
of editFcfir as a double
```



```
% --- Executes during object creation, after setting all
properties.
function editFcfir_CreateFcn(hObject, eventdata, handles)
% hObject    handle to editFcfir (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    empty - handles not created until after all
CreateFcns called
```

```
% Hint: edit controls usually have a white background on
Windows.
```

```
%         See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end
```

```
function editIIRorder_Callback(hObject, eventdata, handles)
% hObject    handle to editIIRorder (see GCBO)
% eventdata  reserved - to be defined in a future version
of MATLAB
% handles    structure with handles and user data (see
GUIDATA)
```

```
% Hints: get(hObject,'String') returns contents of
editIIRorder as text
%         str2double(get(hObject,'String')) returns contents
of editIIRorder as a double
```

```
% --- Executes during object creation, after setting all
properties.
```

```
function editIIRorder_CreateFcn(hObject, eventdata,
handles)
% hObject    handle to editIIRorder (see GCBO)
```



```
% eventdata reserved - to be defined in a future version
of MATLAB
% handles empty - handles not created until after all
CreateFcns called

% Hint: edit controls usually have a white background on
Windows.
% See ISPC and COMPUTER.
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function editFIRorder_Callback(hObject, eventdata, handles)
% hObject handle to editFIRorder (see GCBO)
% eventdata reserved - to be defined in a future version
of MATLAB
% handles structure with handles and user data (see
GUIDATA)

% Hints: get(hObject,'String') returns contents of
editFIRorder as text
% str2double(get(hObject,'String')) returns contents
of editFIRorder as a double

% --- Executes during object creation, after setting all
properties.
function editFIRorder_CreateFcn(hObject, eventdata,
handles)
% hObject handle to editFIRorder (see GCBO)
% eventdata reserved - to be defined in a future version
of MATLAB
% handles empty - handles not created until after all
CreateFcns called
```



Vietnam National Universities – HCMC

International University

Biomedical Engineering Department

```
% Hint: edit controls usually have a white background on  
Windows.  
%      See ISPC and COMPUTER.  
if ispc && isequal(get(hObject,'BackgroundColor'),  
get(0,'defaultUicontrolBackgroundColor'))  
    set(hObject,'BackgroundColor','white');  
end
```