VIRGINIA COMMONWEALTH UNIVERSITY

Virginia Commonwealth University
VCU Scholars Compass

Theses and Dissertations | Graduate School

2014

Development of a compact, low-cost wireless device for biopotential acquisition

Graham Kelly
Virginia Commonwealth University

Follow this and additional works at: http://scholarscompass.vcu.edu/etd

Part of the Biomedical Engineering and Bioengineering Commons

© The Author

Downloaded from
http://scholarscompass.vcu.edu/etd/3559

This Thesis is brought to you for free and open access by the Graduate School at VCU Scholars Compass. It has been accepted for inclusion in Theses and Dissertations by an authorized administrator of VCU Scholars Compass. For more information, please contact libcompass@vcu.edu.

DEVELOPMENT OF A COMPACT, LOW-COST WIRELESS DEVICE FOR BIOPOTENTIAL ACQUISITION

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at Virginia Commonwealth University

by

GRAHAM KELLY
B.S., Virginia Commonwealth University, 2011

Director: OU BAI, PH.D.
Assistant Professor, Department of Biomedical Engineering

Virginia Commonwealth University

Richmond, VA

August, 2014

Acknowledgements

I want to thank my advisor, Dr. Ou Bai, for providing me with the opportunity to work on a subject that I enjoy, and for his good humor even when I made stupid PCB layout mistakes that rendered a batch of boards mostly useless (and I only wish that had just happened once). Thank you for being the jolliest advisor I could possibly have. I also want to thank Dr. Fei for providing the equipment and workspace for all my soldering needs.

I also thank my parents, Charles Kelly and Logan Smith, for their love and support, and for letting me mooch off of them for 24 years and counting. Without their continuous input of effort and generosity, I would be either homeless and uneducated, or dead (probably both). Thanks also to my brother, Jared Kelly, for his invaluably entertaining company in the times I just wanted to hang out and completely forget about "doing science."

I would like to thank my friends and lab-mates (and former lab-mates) Tyler Ferro, Chris Hagerty-Hoff, Plamen Nikolov, Jay Freitas, Natalie Vazquez, and Tareq al-Shargabi, for their company and their valuable input. Thanks also to my girlfriend, Marissa Shaffer, for her encouragement and for keeping me on track.

Finally, I would like to thank Charles Taylor, for his willingness to take me under his wing way back in my just-finished-undergrad days, and for teaching me so much about the practicalities of engineering. If it weren't for him, I would never have developed my interest in electronics in the first place, nor would I have been introduced to the vast maker community that encourages me to continue growing that interest. I learned as much about hands-on engineering work during my year-long stay in the Artificial Heart Lab as I had in four years of undergraduate training, and there are no words or estimation for how valuable that time was.

Table of Contents

List of Tables v
List of Figures vi
List of Abbreviations and Symbols vii
Abstract x
Introduction 1
Background 3
EEG and its Origins 3
EEG Signal Acquisition 4
Wireless EEG Acquisition Devices 5
Design Process 10
Considerations and Parameters 10
Openness 10
Beaglebone and Beaglebone Black 12
Initial Component Selection and Evaluation 14
First Prototype: Brainboard R0 19
Assembly Process 21
Software and Firmware 22
Discussion 27
Second Prototype: Brainboard R1 28
Assembly Process 30
Software and Firmware 31
Design Differentiation 37
Final Design: Brainboard LW 38

Results and Discussion 41
Signal Quality 41
Power Consumption 57
Appendices 72

List of Tables

Table 1: Part selections for initial prototype..... 15
Table 2: ADS1299 command definitions..... 23
Table 3: RMS error between Brainboard LW and Brain Products V-Amp 16. 43
Table 4: 60-Hz noise comparison between Brainboard LW and V-Amp 16..... 44
Table 5: List of common device states and corresponding current consumption 60
Table 6: Component bill of materials..... 62

List of Figures

Figure 1: Brainboard R0 22
Figure 2: Brainboard R1 31
Figure 3: bq24074 lithium-ion/polymer battery charger, typical charge cycle. V_{LOWV} indicates the battery voltage at which the constant-current fast charge phase begins; $I_{O(CHG)}$ is synonymous with the larger of I_{CHG} and $I_{IN(MAX)}$. 35
Figure 4: Battery management expansion board 37
Figure 5: LTC1998 configured as low battery threshold detection with hysteresis 39
Figure 6: Brainboard LW 41
Figure 7: Magnitude response to swept sinusoidal input 44
Figure 8: Noise spectrum for shorted inputs 45
Figure 9: Noise spectrum at ADS1299 specified range 46
Figure 10: Unfiltered noise histogram 47
Figure 11: Noise histogram after 0-65 Hz low-pass filter 48
Figure 12: Channel 3 amplitude spectrum comparison 49
Figure 13: Channel 4 amplitude spectrum comparison 50
Figure 14: Channel 5 amplitude spectrum comparison 51
Figure 15: Channel 6 amplitude spectrum comparison 52
Figure 16: Channel 3 time-domain comparison 53
Figure 17: Channel 4 time-domain comparison 54
Figure 18: Channel 5 time-domain comparison 55
Figure 19: Channel 6 time-domain comparison 56

List of Abbreviations and Symbols

ADC	Analog-to-digital converter
AFE	Analog front-end
ALS	Amyotrophic lateral sclerosis
ASF	Atmel Software Framework
ASIC	Application-specific integrated circuit
BBB	Beaglebone Black
BCI	Brain-computer interface
BOM	Bill of materials
CMOS	Complementary metal-oxide-semiconductor
CAD	Computer-aided design
DMA	Direct memory access
DMM	Digital multimeter
DSP	Digital signal processing
ECG	Electrocardiogram
EEG	Electroencephalogram
EEPROM	Electrically erasable programmable read-only memory
EMG	Electromyogram
EOG	Electrooculogram
ESD	Electrostatic discharge

FCC	Federal Communications Commission
FIR	Finite impulse response
FPGA	Field-programmable gate array
GUI	Graphical user interface
HDL	Hardware description language
I^2C	Inter-Integrated Circuit
IC	Integrated circuit
IC	Integrated circuit
IDE	Integrated development environment
JTAG	Joint Test Action Group
LDO	Low-dropout (linear regulator)
LVTTL	Low-voltage transistor-transistor logic
MCU	Microcontroller unit
MEMS	Micro-electromechanical system
MS	Multiple sclerosis
PCB	Printed circuit board
PRUSS	Programmable real-time unit sub-system
RAM	Random access memory
RF	Radio frequency
SPI	Serial Peripheral Interface
SRAM	Static random-access memory
TI	Texas Instruments

TVS	Transient voltage suppression
TWI	Two-Wire Interface
TWIM	Two-Wire Interface Master
UART	Universal asynchronous receiver/transmitter
VLSI	Very-large-scale integration

Abstract

DEVELOPMENT OF A COMPACT, LOW-COST WIRELESS DEVICE FOR BIOPOTENTIAL ACQUISITION

By Graham S. Kelly, B.S.

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at Virginia Commonwealth University.

Virginia Commonwealth University, 2014.

Major Director: Ou Bai, Ph.D., Assistant Professor, Department of Biomedical Engineering

A low-cost circuit board design is presented, which in one embodiment is smaller than a credit card, for biopotential (EMG, ECG, or EEG) data acquisition, with a focus on EEG for brain-computer interface applications. The device combines signal conditioning, low-noise and high-resolution analog-to-digital conversion of biopotentials, user motion detection via accelerometer and gyroscope, user-programmable digital pre-processing, and data transmission via Bluetooth communications. The full development of the device to date is presented, spanning three embodiments. The device is presented both as a functional data acquisition system and as a template for further development based

on its publicly-available schematics and computer-aided design (CAD) files. The design will be made available at the GitHub repository https://github.com/kellygs/eeg.

Introduction

A brain-computer interface, or BCI, is a modality for human-computer interaction whereby a person may control an external device using brain signals without muscular intervention. The brain signals of interest may be time-varying electrical potentials generated by firing neurons, variations in the brain's magnetic field, or changes in local blood flow, all of which are correlated with different mental states. One of the most widely used signals is the electroencephalogram (EEG), since it has excellent temporal resolution, can be measured non-invasively, and is comparatively inexpensive and simple to acquire.

BCIs are often the best, or even the only, option for a variety of diseases and traumas. They are of the few options available for victims of so-called "locked-in syndrome," a state in which voluntary muscle control is lost over the entire body. This can be a unique disease or a symptom of advanced amyotrophic lateral sclerosis (ALS). It was estimated that approximately 30,000 Americans had ALS in 1999, with an annual incidence rate of one or two cases per 100,000 people [1]. BCI is also a promising avenue for prosthetic limb technologies, an area of particular interest to the U.S. military due to the increasing problem of limb loss among troops returning from Iraq and Afghanistan. For instance, according to the Armed Force Health Surveillance Center, in 2011, 240 deployed troops required upper or lower limb amputation, higher than any year prior. Furthermore, the total number of amputations occurring during all Iraqi and

Afghan conflicts was reported at 1,599 as of May 3, 2012 [2]. Outside of battle, over 1.7 million Americans live with at least one lost limb [3]. One study examining the prevalence of U.S. limb loss up to 2005 estimated that as many as one in 95 Americans may have had an amputation by the year 2050 [3]. With such need, it is no wonder that BCI is an area of active research.

In the following, I present background on the EEG, especially in forms suitable for use as a practical BCI (i.e. wearable, wireless systems). I then move on to discuss the design rationale behind the prototype device presented herein, along with its design evolution over time and its performance metrics as compared to a commercial system. I conclude with future directions for the project.

Background

EEG and its Origins

The first electroencephalogram acquired from a human being is attributed to Hans Berger, a German psychiatrist, who in 1929 published his findings on electrical activity of the brain recorded from the scalp [4]. Despite initial skepticism from the scientific community, EEG has proven to be one of history's most important contributions to clinical electrophysiology. Analysis of the EEG allows for noninvasive determination of a number of clinically relevant parameters, including seizure localization and classification [5], but also the detection of drowsiness [6], certain affective states [7], and motor imagery [8], among others. It has also become a tool for neurofeedback, which has shown potential as a clinical tool for enhancing self-regulation and inducing neuroplasticity [9].

The EEG originates from the formation of dipoles within the brain as the result of neuronal ion flow. Changes in relative ion concentration, and thus in net charge, between the intracellular and extracellular spaces generate electric fields whose orientations are determined by the orientation and geometry of the neurons from which they originate. These electric fields cause wavelike movement of charge, known as volume conduction, through the cranial contents and the cranium itself, ultimately resulting in a distribution of nonzero electrical potentials on the scalp. Although these potentials are present, the varying orientations of the neuronal dipoles, the large

number of different signal patterns being generated, and the distance over which any given signal must propagate through volume conduction to reach the scalp invariably produce a greatly attenuated and spatially smeared representation of the brain's electrical activity. Despite this, sufficiently sensitive equipment can detect resultant changes in voltage across the scalp, albeit with poor spatial resolution.

EEG Signal Acquisition

The EEG is acquired through the placement of electrodes on the scalp, which follow the scalp's change in electrical potential. The potential difference between electrodes can then be amplified to a workable level, at which point it is typically digitized and stored on a computer for analysis.

Because the EEG is a tiny signal on the order of microvolts in amplitude, owing to the summing and blurring effects of volume conduction, acquisition of good-quality EEG signals has stringent requirements on the electrical impedance of the body-electrode contact and the noise resistance of the transmission line between the electrode and the amplifier. Minimization of the scalp-electrode impedance is typically acquired through mechanical abrasion to remove the stratum corneum, the high-impedance upper layer of the epidermis consisting of dead skin cells, followed by application of an ionic gel between the electrode and the skin. Hairy areas such as the head pose a challenge to this impedance minimization process, usually necessitating more gel than would otherwise be used; in cases where maximum signal quality is required, the hair is typically shaved. Although research is ongoing into dry, through-hair

electrodes that match the interface impedance (and thus signal quality) of the traditional type, and some have even made it to market, wet electrodes remain the gold standard for clinical EEGs.

The amplifier itself must also contribute as little noise as possible. Although past amplifiers and analog-to-digital converters (ADCs) tended to run on high-voltage supplies—e.g. ±12V or ±5V—necessitating very high gain, modern amplifiers are capable of rail-to-rail operation from a single-ended voltage supply as low as 5V, 3.3V, or 1.8V, and digital logic levels continue to fall. As a result, much lower gains can be used. This effect is compounded by the advent of ultra-low-noise amplifiers and modern analog-to-digital (A/D) techniques that allow acquisition with up to 24-bit nominal precision. Sigma-delta ADCs, for example, drastically oversample the acquired signal and then use noise-shaping filters and decimation to produce a lower-rate signal with high precision; their inherent low-pass filtering mean that analog anti-aliasing requirements are far less strict than in traditional ADCs. These innovations, combined with improvements in wireless communication technology, mean that EEG amplifiers—once bulky and high-powered devices—can now be miniaturized to the point of being wearable devices.

Wireless EEG Acquisition Devices

The benefits of a small, wireless EEG system are numerous, both medical and non-medical. An ambulatory EEG device either with wireless connectivity to a host computer or the Internet, or with sufficient local storage to record large amounts of data in a home setting, would be extremely useful for clinicians in that they could learn about

a patient's EEG activity in a natural setting, improving patient outcomes. For instance, sleep activity could be monitored at home, or pre-ictal EEG could be analyzed for a patient who experienced a seizure during their everyday routine rather than at a hospital. A wearable EEG system could also enable a more practical non-invasive brain-computer interface in disabled patients, such as those with ALS or MS, than the traditional bulkier designs. On the other side of the coin, a wearable EEG system could enable a wide array of recreational applications, from simple focusing games to neurofeedback for relaxation or simply as a novel control method for external devices. Such devices would need to be as easy as possible to use, which means light weight and long battery life.

Some commercial devices are already available to meet this market. One is the B-Alert X systems from Advanced Brain Monitoring [10], which feature 3, 9, or 23 channels of EEG and 1 auxiliary differential channel, along with an accelerometer for head movement detection. The X10, the midline model with 10 channels, claims power consumption of 40 mA at 3.7V, with a standard battery life of 12 hours and an optional battery extension giving 24 hours of battery life. Advanced Brain Monitoring also sells the Stat-X series with the same features as B-Alert X, but approved for clinical use. All the devices use a proprietary 2.4 GHz RF link to a host computer, however, which limits their interoperability or use for device control. They give no price on their website, suggesting a very expensive system.

Another such commercial device is the EPOC from Emotiv [11], which uses 14 electrodes placed according to the international 10-20 system on an easily donned headset, along with a MEMS gyroscope for head position information. The intent is to

interpret the EEG (transmitted to a host computer via proprietary 2.4 GHz RF) using a suite of affective classification algorithms as a novel form of human-computer interaction. The EPOC retails for only $299, making it one of the few affordable wireless EEG systems on the market. By default, however, the raw EEG signals are not exposed to the user; this requires purchasing a license and SDK to develop novel uses or do research with the device. This costs an additional $451, making such activities a bit further outside the reach of the average consumer. There is some question as to the quality of the EPOC's data compared to a medical-grade EEG system [12] but an increasing number of researchers have used it for BCI experiments [13].

NeuroSky manufactures the MindWave Mobile headset, which centers around an EEG application-specific integrated circuit (ASIC) that compactly acquires and interprets EEG with low power consumption [14]. This ASIC, called TGAT, includes on-chip bandpass filtering down to the EEG band, mains noise notch filtering, and individual frequency band extraction. It also performs an unknown proprietary algorithm for quantifying attention and relaxation. The headset is made easier to use by the fact that it uses Bluetooth as its wireless protocol, and thus it is potentially more compatible with existing wireless devices. However, the MindWave Mobile's single dry electrode is located on the forehead, resulting in extreme signal contamination by EMG, and the frequency band output variation of the device was found in one instance to be the same whether or not the headset was being worn [15].

Lin et al. conducted a survey of published work on wireless/wearable EEG systems in 2010, with a focus on BCI applications [16]. They report a design published in 2002 by Cheng et al. for acquiring EEG and transmitting it to a computer for BCI

applications, but they give no details on the implementation of their device, focusing instead on the processing paradigm and its results [17]. In a 2007 paper, Matthews et al. developed a low-power data acquisition system and wireless transceiver to operate with an experimental capacitive dry electrode design [18]. They report that their design operated off of two AA batteries for 72 hours. However, the design was proprietary (a result of R&D by Quantum Applied Science and Research) and no details on its construction, weight, or potential cost were given. Lin et al. reported in 2008 on their embedded BCI that used a Texas Instruments applications processor including both an ARM9 core and a 16-bit low-power DSP core [19]. Although they give a reasonable amount of detail regarding what components went into their system, the processor module on which it is based is no longer available from TI [20].

One notable aspect of all the wireless EEG systems observed in the Lin review, as well as in the literature at large, is the lack of detailed descriptions of the devices, their performance, and their construction. This is understandable, given the all-too-certain desire of the authors to preserve their intellectual property for later lucrative patenting. However, this tends to result in multiple concurrent efforts being expended redundantly across multiple research teams; one need only examine the volume of independent groups represented in a Google Scholar search of “wireless EEG BCI” to ascertain this fact. This restricts progress to the teams with prior knowledge of how to build these designs and/or the funds to quickly get up and running with outside assistance. At the time this thesis was begun, only one EEG system was fully open, in the sense that all hardware schematics, software, and firmware were freely available online: openEEG [21]. This project, first released around 2002, allows one to build a

low-cost 2-channel desktop EEG system for $200-$400, depending on part sourcing and whether or not electrodes are bought or made oneself. However, the system is not at all wearable and has no wireless component. This is due in large part to the now-antiquated electronics used in the design. This thesis thus set out to fill in the wireless/wearable open-source EEG gap.

One note before continuing: in parallel with this thesis, a group planned and successfully Kickstarted a very similar system known as OpenBCI [22]. This system uses much of the very same hardware applied in this thesis, works as an add-on board for the popular Arduino open-source microcontroller platform [23], and promises to be very successful, given the amount of interest it has generated and the significant funding it has behind it. Although OpenBCI in its current incarnation does not have inbuilt wireless communications, the group states that they have a wireless and battery-powered system in the works. It is very likely that OpenBCI will become the future platform of choice for open-source EEG experiments and homebrew development.

Design Process

Considerations and Parameters

The initial design parameters were that the device should be *wireless*, to enable user mobility; that it should be as *small and lightweight* as possible, implying a high level of component integration and low power consumption to enable a relatively small battery; and that it should be capable of performing at least some basic amount of *digital processing* on the signal it acquires, such as band power extraction, to reduce or eliminate the device's dependency on a host computer and thus further enhance user mobility. As will become clear over the course of this chapter, not all of these parameters were met in a single design, resulting in differentiated designs separately optimizing size and processing capability.

Openness

One aspect of this design that evolved over the course of reviewing the literature was the importance of making the design specifications freely available. Although many researchers have previously developed wireless EEG systems for BCI applications, and some were even commercially available, no finished designs at the time this project was begun had been made truly "open"—no circuit schematics, board layouts, or source

code were posted anywhere for general perusal. This made accurately assessing the state of the art much more difficult, but presented a gap in the field that could be met. Although the basic functionality of the final design isn't completely novel, its near-uniqueness lies in the fact that its hardware and software components are available for free download by anybody. I made the decision to make my design as "open" as possible in the following ways:

- Use only of off-the-shelf components that are openly documented and accessible to hobbyists. This also means that programmable components (e.g. microcontrollers) must have free or low-cost development tools. This has the added benefit of making my own development process much cheaper and easier, but its primary purpose was to ease collaborative modifications of the design in the future.
- Publishing of the entire design—hardware schematics, board artwork, and firmware—to a GitHub repository for public perusal. This would greatly increase the potential value of the design simply by allowing more people to build upon it, if they so choose.

Although use of VLSI hardware design (i.e. custom chip design) doesn't strictly preclude openness with regards to sharing results—the design could, for example, be implemented on an FPGA and the HDL code could be made freely available—full-featured FPGA development environments are quite expensive and not quite in the spirit of "open source" design. It was thus determined that signal processing would take place wholly in software, which necessitated selection of a suitable microprocessor, as described in the following section.

Beaglebone and Beaglebone Black

To allow for sophisticated signal processing in software, it was desirable that a high-speed microprocessor capable of running a full-featured operating system be included in the design. With a desktop or laptop PC out of the equation due to the requirement for mobility, some kind of embedded processor was desired. Smartphones were considered as a platform, but ultimately rejected for a number of reasons:

- A smartphone platform excludes users who do not already own smartphones (although this number is admittedly dwindling every day)
- A smartphone platform would most likely exclude iPhone users during early development, since iOS presents relatively more obstacles versus Android to app development and Bluetooth communications (excepting Bluetooth Low Energy, which was much less mature at the time of initial design planning)
- A local, wired connection to the signal acquisition hardware would be ideal, since it would contain all the components of a BCI system, minus the electrodes, in a single package, and would save energy by eliminating wireless data streaming

Since embedded processors and their associated memory chips are typically in difficult-to-solder packages and can be difficult to work with, requiring close attention to board layout for high-speed signals, a preexisting single-board computer was sought as a host processor. Popular single-board computers included the Raspberry Pi [24] and the Beaglebone (along with its upgrade, the Beaglebone Black, which was released

during design planning [25]). Although the Raspberry Pi was, and remains, the most popular single-board computer within the maker community, it was outmatched in compactness and expandability by the Beaglebone devices.

The Beaglebone Black (BBB) was ultimately selected as the computational engine for the design. It is a 3.4” x 2.15” single-board computer with HDMI, USB host, and Ethernet capability. Its processor is a 1 GHz ARM Cortex-A8 and it has 512 MB of RAM, along with 4 GB of non-volatile flash memory and microSD card support. It has 92 I/O pins for external interfaces and is capable of interacting with devices that would normally require a deterministic microcontroller, due to its two independent on-chip Programmable Real-time Unit Subsystems (PRUSSs).

The Beaglebone and BBB are specifically meant to work with expansion boards, called “capes,” which plug into their I/O sockets and provide additional functionality. For example, there are capes containing Wi-Fi radios, motor control circuitry, and AA battery slots [26]. Capes are recognized by the host using configuration settings loaded onto an EEPROM chip, which is thus required for all capes. This project’s design could therefore easily couple with the BBB simply by designing it as a cape.

The BBB’s current consumption, rated at a typical 210-460 mA depending on activity, would pose an obstacle to wearability if battery life were to be maximized. Thus a dual-mode PCB was envisioned: the device could have its own low-power microcontroller onboard so that it was capable of operating standalone with limited or absent processing of acquired data, or it could plug onto the BBB and gain additional processing capacity at the expense of battery life.

It is important to note that, due to time constraints, BBB software was not developed as part of this project, which instead focused on refining the standalone aspect of the design (i.e., offloading the signal processing to a computer via wireless communication). However since the BBB runs Linux, existing BCI or EEG acquisition software may be adapted for this purpose. For example, BCI2000 [27] in its original incarnation is known to work on a 1.4 GHz single-core machine with only 256 MB of RAM [28]. Open Ephys, an open-source hardware/software solution for electrophysiology experiments such as intracellular recordings, includes a lightweight customizable GUI with built-in support for data processing blocks [29]. As part of a side project, a rudimentary data engine was established by Jay Freitas for reading from the BBB's UART port and displaying 8 channels of data; however, it has not yet been expanded upon, and the display GUI provides no means of bidirectional communication with the ADS1299. Robust Beaglebone software would be a clear direction for further development.

Initial Component Selection and Evaluation

Before any custom circuit design was attempted, the major individual components had to be selected. Once these major components were known, evaluation boards for each could be purchased and linked together to build a prototype system. This process was carried out using the design criteria previously listed, as presented in Table 1.

Function	Part Selected	Justification
Signal acquisition	Texas Instruments ADS1299 analog front-end	Signal conditioning and ADC on single chip: eases design and consumes less board space than discrete solution
System control	Atmel AT32UC3L064 32-bit AVR microcontroller	Low power consumption, high speed (up to 48 MHz), small package size, direct memory access engine for fast peripheral communications, native DSP instructions, free development environment (Atmel Studio)
Wireless communications	Microchip RN-42 Bluetooth module	Widely used and popular among hobbyists, simple interface, pre-certified by FCC, no RF design needed, Bluetooth protocol allows simple interface with common hardware like PCs and Android phones

Table 1: Part selections for initial prototype

The ADS1299 is an 8-channel, 24-bit, low-noise analog front-end for biopotential measurements, specialized for scalp EEG applications. It provides both analog signal conditioning (i.e. low-noise programmable-gain amplifiers) and analog-to-digital conversion in a single package. All 8 channels are fully differential and sampled simultaneously, and the device may be daisy-chained and synchronized with additional ADS1299s to expand the number of channels almost arbitrarily. Its high bit resolution gives it both precision and dynamic range, allowing it to capture signals as high as 4.5 V and as low as 0.5 μV. This high input range means that it can easily be repurposed to capture various electrophysiological signals other than EEG, such as EOG, EMG, or ECG. It can also sample at rates from 250 Hz to 16 kHz and includes on-chip circuitry for setting the patient bias voltage, along with lead-off/lead-impedance detection. Finally, it incorporates a sophisticated multiplexer that allows for dynamic selection of

reference and bias electrodes in the event of lead-off or excessive lead impedance, along with an array of diagnostic modes for (e.g.) measuring the bias voltage, measuring the die temperature, or performing calibration and internal noise tests.

Although the ADS1299 is expensive ($58.14/chip in quantities below 10 from Digi-Key), it more than makes up for this in board space reduction due to the sheer amount of features. Only passive components and two external op amps per ADS1299 (if multiple AFEs are used) are necessary for buffering the reference electrode and driving the cable shield, respectively, and single-package dual op amps are widely available.

Selection of the system microprocessor/microcontroller was the most time-consuming decision. The processor needed to consume as little power as possible and yet have enough processing capability to handle, at the very least, real-time FIR filtering of a reasonable-length signal. It also needed to be both reasonably easy to program (high-level languages like C/C++ were a must) and inexpensive in both time and money to develop for (low-cost or free development tools, freely available libraries for on-board peripherals and, if possible, for signal processing).

The initial desire to make the design open-source led toward an Arduino-compatible microcontroller, but the lack of computational power in the 8-bit Arduinos and the excessive chip size and power consumption of the 32-bit Arduino Due ruled it out as a useful candidate for a battery-powered wearable system. I leaned toward 32-bit microcontrollers since they would be able to handle the 24-bit data from the ADS1299 more quickly. Of those I could find during my initial search, only Atmel’s 32-bit

microcontrollers had a completely free development environment. Atmel offered two families of MCUs that had specialized DSP instructions: those with the AVR UC3 core, and those with the ARM Cortex-M4 core. Of these, the former was older and more well-established, which I assumed would be a benefit in terms of support and chip availability.

It should be noted at the outset that the selected microcontroller may not have been the best for a number of reasons. The AT32UC3L064 has only 16 KB of SRAM, which limits the size of its data buffers and thus the amount of signal processing that can be done on-chip. More importantly, however, its documentation and support (both professional and from the user base) are lacking compared to comparable but more popular chips such as the wide variety of ARM Cortex-M microcontrollers. Although I initially supposed that the chips' being slightly older than Atmel's Cortex-M devices would make them more established and thus give them a wider support base, this was discovered after a long and frustrating period of development not to be the case.

Future modifications of this design should upgrade to something with a wider following, perhaps something based on the Teensy 3.1 (a Cortex-M4 device with support for the Arduino IDE). Although the Teensy designs are not completely open-source, they are very popular within the hobbyist and open-source hardware community due to their Arduino IDE support, and thus this drawback might be acceptable.

The RN-42 is a Bluetooth pre-certified module for drop-in replacement of serial (RS-232) cables. Because it is pre-certified, it can be used in end devices without additional certification by radio emissions regulatory bodies such as the FCC. It

interfaces with the Universal Asynchronous Receiver/Transmitter (UART) of a host microcontroller and converts data bidirectionally between the wired CMOS/LVTTL-level RS-232 protocol and the Bluetooth 2.1 SPP (Serial Port Protocol). The module contains on-board flash memory for settings and Bluetooth stack storage, along with a Bluetooth system-on-chip that executes the stack and interprets data from the UART. The RN-42 is a Bluetooth Class 2 device, meaning that it emits up to 2.5 mW of RF power and thus has a typical range of about 10 meters. The device is well established within the hobbyist community due to its ready availability on SparkFun, a popular DIY electronics website.

Evaluation boards were purchased for each of the core parts. For the microcontroller, the specific board selected was the UC3L Xplained, along with an AVR Dragon programmer board for programming and debugging. The RN-42 was evaluated using the popular BlueSMiRF Silver board from SparkFun. In the case of the ADS1299, only the official evaluation module from TI was available, and so that was used. All additional parts of the later custom-PCB prototypes (voltage regulators, passive components, etc.) were selected using the reference schematics for these evaluation boards, which greatly eased the design effort.

Because the device is coupled through a low-impedance path to the user's scalp, some measure of protection is needed to ensure user safety. In addition, the circuitry should be protected against electrostatic discharge (ESD).

The device is battery-powered, which reduces the need for on-board protection circuitry. For the initial prototype, the problem of user isolation during battery charging

was sidestepped by simply not using the device while the battery charges. To prevent taking up extra room on the device's PCB, isolation should ultimately be implemented using an isolated USB cable. This type of cable contains integrated isolation circuitry.

All analog lines connecting to the user are current-limited as an inherent aspect of the passive antialiasing filters. This provides two-way protection: both the user and the device benefit from the limited current. However, ESD can generate voltages too great to allow the use of current-limiting resistors alone for device protection, since larger resistors produce more thermal noise and must be matched with impractical precision between differential inputs. As an additional safeguard, ESD protection diodes were used. In general, an ESD solution for precision analog circuitry should have the lowest possible leakage current and capacitance in order to prevent signal distortion. Texas Instruments' TPD4E001 is a quad-channel ESD-protection diode array that meets these requirements, with less than 1 nA of leakage current and 1.5 pF I/O capacitance. It is available in a number of small packages, down to 1.6x1.6 mm. Their layout causes excessive voltage spikes to be harmlessly redirected to one of the power rails, where they are suppressed by a decoupling capacitor.

First Prototype: Brainboard R0

The first prototype, the Brainboard R0 (revision 0), was meant as a proof of concept and also to test the feasibility of assembling a surface-mount PCB in-house. It used the standard Beaglebone cape form factor, with the intention that it would be able to operate in two modes: either as a standalone board capable of transmitting EEG data

over a Bluetooth link, or as a cape for the BBB capable of sending and receiving data to/from that device over one UART link, then relaying commands from the BBB to the Bluetooth module over a second UART.

The R0 was not intended as a finished design, and thus it did not include all the necessary components to generate every power supply on-board. The ADS1299's analog circuitry was powered by an off-board 5V supply, which was post-regulated using an LDO linear regulator to reduce switching noise that would be introduced from the BBB. All digital circuitry, which comprised the remainder of the circuitry on the board, was powered by an off-board 3.3V supply. Both of these supplies were broken out on the BBB I/O headers, so the design was essentially dependent on either the BBB or some other external power supply in order to run. All evaluation was done using a bench-top power supply and jumper wire.

Initially, it was planned that schematic creation and PCB layout would be done in Cadsoft's EAGLE program, which is a program I was familiar with and which has a freeware version that is very popular in the maker community. However, it quickly became apparent that the complexity of this design and the requirement for proper shielding of analog signals was going to require at least four copper layers, and EAGLE's freeware edition can only generate two. Since the non-freeware versions of EAGLE that could be purchased by the university are expensive (the lower-cost hobbyist version is for individual use only), an alternative had to be pursued. This came in the form of KiCAD, an open-source suite of schematic capture and PCB layout software.

One key decision that had to be made in this design was whether to break out both the positive and negative inputs to the ADS1299 for every channel. Referencing all the positive inputs to a single negative input (the “reference channel”) is standard procedure for most EEG systems, and it would also greatly simplify board layout. This was therefore the option chosen for R0.

The onboard microcontroller was to be programmed and debugged using a 10-pin JTAG port. This would be connected to the AVR Dragon, which in turn would be connected to a PC via USB cable.

Assembly Process

The PCBs were ordered from OSHPark, a low-cost board fabrication service. Because the lab did not have a means of reflowing the surface-mount components, it was necessary to affix them by hand using a soldering iron. Although this process was time-consuming and error-prone (initial assembly, plus finding and repairing hidden solder bridges and cold joints, took over a week) it was ultimate successful in producing a functional, if aesthetically imperfect, board.

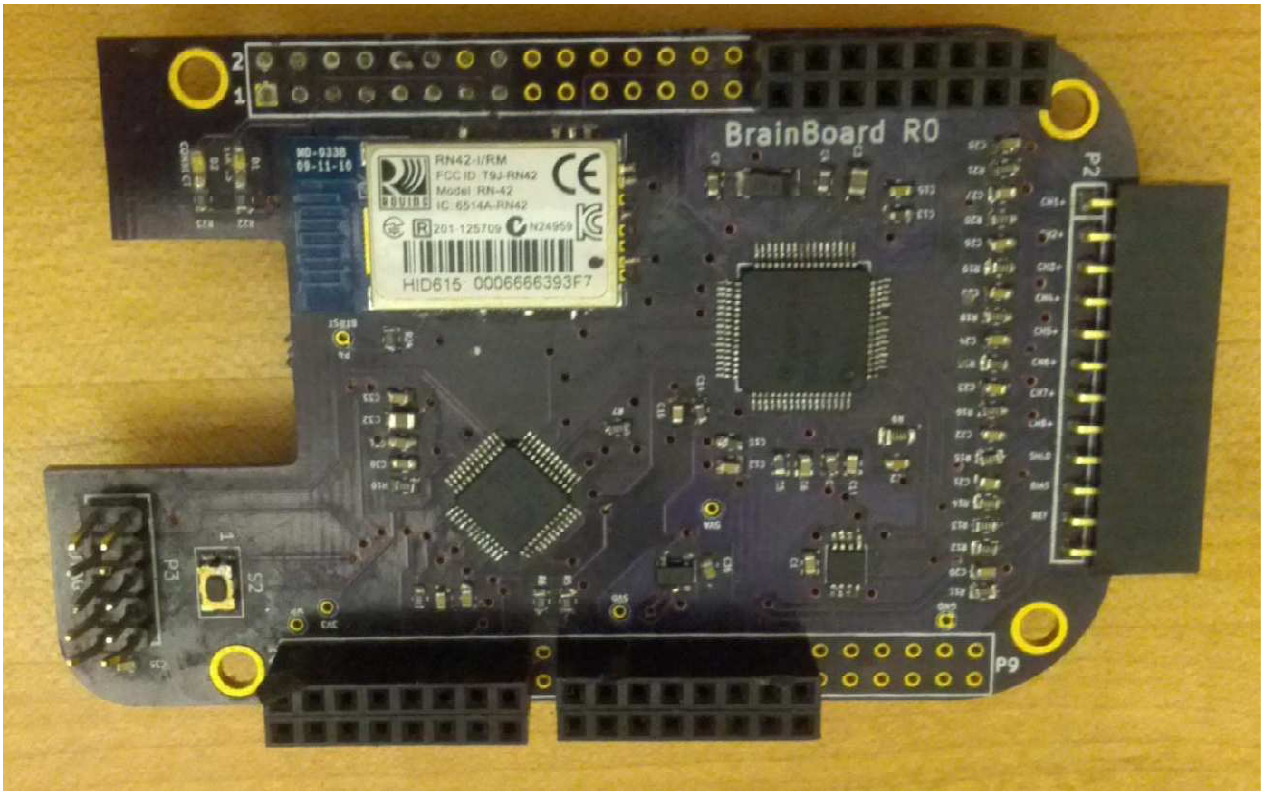


Figure 1: Brainboard R0

Software and Firmware

Firmware was developed in C using the free Atmel Studio IDE, and debugged/flashed onto the board using the AVR Dragon connected to the 10-pin JTAG

port. Atmel's freely available set of libraries, known as the Atmel Studio Framework (ASF), was used as much as possible in order to reduce development time and effort, both for me and for future developers. Using the ASF was also an attempt to ensure maximum cross-platform code compatibility with other Atmel products, in the event that another microcontroller was used in a future revision. All firmware for the final design is given in **Error! Reference source not found.**.

The ADS1299 is controlled via the SPI protocol and supports 10 commands. These are given in Table 2, which is derived from page 35 of the device datasheet [30].

Command	Description	First Byte	Second Byte
System Commands			
WAKEUP	Wake up from standby mode	02h	-
STANDBY	Enter standby mode	04h	-
RESET	Reset the device	06h	-
START	Start and restart (synchronize) conversions	08h	-
STOP	Stop conversion	0Ah	-
Data Read Commands			
RDATAC	Enable continuous data read mode	10h	-
SDATAC	Disable continuous data read mode	11h	-
RDATA	Read data by command	12h	-
Register Read Commands			
RREG	Read from register(s)	2Rh	0Nh
WREG	Write to register(s)	4Rh	0Nh

Table 2: ADS1299 command definitions

The register read commands differ from the rest. They are 2-byte commands; the first byte signifies dataflow direction (read or write) with its 6th and 5th bits, while bits 4 through 0 (indicated by "R" in Table 2) indicate the register address of the first register to be accessed. The second byte's lower 5 bits then indicate how many registers to access sequentially after the first register. In this way a single command can be used to initiate multiple register reads/writes. The WREG command is always followed by at

least one more byte, which contains the data to be written to the selected register. The firmware and software implemented in the Brainboard designs does not support multiple register accesses in this fashion from the host computer; the maximum number of bytes in a received command is 3. Writing or reading multiple sequential registers must be done with separate WREG/RREG commands.

The code is entirely event-driven. On power-up, the MCU initializes its on-chip peripherals (clocks, DMA controller, data and auxiliary UART, and SPI), and then verifies that the ADS1299 is working properly by requesting its chip identifier byte, which is specified in the device datasheet. If this byte is wrong, the MCU sends a message to this effect on UART and goes into a low-power idle state. Otherwise, it initializes the ADS1299 registers such that, upon receipt of the START command, it will begin sending data continuously at a rate of 250 Hz. The MCU then goes to sleep and waits for receipt of a three-byte sequence on the data UART, corresponding to the maximum number of bytes in an ADS1299 SPI command. These three-byte commands come from the controlling computer and are instantly relayed over the SPI bus, allowing the computer to reprogram the ADS1299 if necessary. More importantly, the controlling computer sends the START command to begin acquiring data.

Since the ADS1299 is put in continuous read mode, only one START command needs to be sent. The MCU sleeps until the ADS1299 asserts its DRDY (data ready) pin, which triggers an interrupt. After receiving this interrupt, the MCU reads a 3-byte status word and eight 3-byte data words from the ADS1299, stores them in a buffer to be transferred over the data UART, increments a counter, and then compares the counter to a threshold. If the counter has reached the threshold, it initiates DMA transfer

of the buffer over the data UART and goes to sleep. Otherwise, it simply goes to sleep and waits for the next DRDY assertion.

Importantly, the UART connection between the MCU and the RN-42 must use some form of hardware flow control; otherwise, the RN-42 can quickly become overwhelmed by data coming in at a baud rate greater than 9600. Hardware flow control, in this and most cases, refers to the use of two out-of-band signals, designated RTS and CTS, as the hardware layer of a handshaking protocol. These are crossed over between endpoints, meaning that RTS of one endpoint connects to CTS of the other, and vice versa. In the null state, each endpoint holds its CTS line low, meaning that it is ready to receive data. When an endpoint's input buffer is almost full, it asserts its CTS line, which the other endpoint uses to gate its transmissions. Asserting CTS on a completely full buffer is likely to drop data, since it is very possible for the other endpoint to have sent some bytes at the same time the control signal was sent. Transmission to the overwhelmed endpoint will thus halt until its buffer has cleared, at which point it will again clear its CTS line. From the viewpoint of the microcontroller communicating with the RN-42, the program must halt all transmissions upon assertion of the MCU's RTS line (connected to the RN-42's CTS line), and then resume them once RTS is cleared.

Since the MCU is much faster than the RN-42 at data handling, and since only very small amounts of data move from the RN-42 to the MCU, unidirectional flow control is all that is really needed. This is simpler to implement, since it doesn't require an estimation of what level constitutes an "almost full" UART buffer. This is a useful point to keep in mind for implementation on simpler microcontrollers without hardware support

for RTS/CTS handshaking; however, the UC3L has such support, and thus all flow control is handled transparently.

Data is transmitted in 42-byte packets, and the UART baud rate is the RN-42's default 115,200 bits/s. To minimize power consumption, data is sampled from the ADS1299 at the minimum rate of 250 Hz. Although this can be increased up to 16 kHz, doing so provides no clear advantage for BCI applications, since all the relevant frequency bands are below 30 Hz, giving plenty of Nyquist headroom at even the minimum data rate.

On the host computer side, the BCI2VR MATLAB program was used [31]. Acquisition in this environment consists of starting the Data Acquisition module and loading a "setup file," which is an m-file that contains the information needed to open a connection with the Brainboard: the MAC and channel numbers for the RN-42 Bluetooth module, the number of channels being used (up to 8), and the scaling factor to convert the ADS1299 output integers to microvolts. An example setup file is given in **Error! Reference source not found.**. Once the setup file is loaded, BCI2VR will attempt to open a connection and program the ADS1299 with its default settings: gain of 24 and 250 Hz sampling rate. If this fails, an error message will be thrown and the process will have to be restarted. Otherwise, the connection is opened and the user may proceed by pressing the green triangle ("play") button on the BCI2VR GUI, which begins display of the data. Actual acquisition streaming is begun as soon as the connection is successfully opened.

An additional button is presented in the GUI for turning on the ADS1299's internal calibration signal on all channels. This is a square wave of amplitude 1.875 mV

and frequency of approximately 1 Hz (2.048 MHz / 2^{21}). Since the signal is at known amplitude and also is generated inside the chip, it can be used to simultaneously calibrate the device and determine if a fault in the signal chain is rooted in firmware or in a faulty hardware connection. Pressing this button once resets the ADS1299 and programs the MUX bits for all channels such that they are internally connected to the test signal generator; pressing it again resets the chip and returns it to normal operation.

The ADS1299 contains a huge array of other features that could be conveniently controlled using the BCI2VR Data Acquisition GUI, such as lead-off detection and impedance monitoring. Although these have not been fully implemented in the current version, it would be trivial to do so using the calibration button implementation as a reference.

Discussion

As a proof of concept, Brainboard R0 was successful. However, because it lacked a way to power itself from a battery, it was incapable of running in standalone mode without extra circuitry. This was done for the first prototype as a temporary measure, so as to avoid designing a complex circuit that could coexist with the BBB's power management system (including battery detection and charging). The difficulty of reconciling these two power management systems without introducing expensive redundancy was a strong impetus for the later design differentiation (see Design Differentiation below).

Furthermore, the design had a serious layout mistake that rendered the ESD protection diodes useless. The particular package selected for these diodes was chosen

because it was the largest package with external leads, and thus it would be easiest to work with. Unfortunately, the pinout differed from package to package, and when I created the schematic symbol, I accidentally used the pinout from a similar-looking package. As a result, the diodes were connected incorrectly to the remainder of the circuit, with the effect that shorting all the signals together (as was done in the externally shorted noise test) actually shorted the 5V power and ground rails together. This destroyed most of the ICs on the board, and they had to be replaced. Further tests of the board had to proceed with the diodes uninstalled.

Once basic functionality of the EEG acquisition system was confirmed, further revisions were undertaken in order to correct the above issues.

Second Prototype: Brainboard R1

The second design improved on the layout issues from R0, and expanded its functionality by adding inertial sensors in the form of the MPU-6050, a 16-bit, 6-axis digital accelerometer/gyroscope with an I^2C interface. It was further improved by altering the way the ADS1299 inputs were broken out. Two of the channels were made differential; i.e., for channels 1 and 2, both the positive and negative inputs were broken out for electrode connection. The rest of the negative inputs were hard-wired to the common reference electrode input pin, SRB1. This allowed the ADS1299 to be placed in differential mode overall, producing the effect of two differential channels and six single-ended channels. The two differential channels were for use as EOG electrodes, which would allow simpler removal of eye-movement artifacts from recordings. If the

differential capability was undesired, the negative inputs could easily be overridden simply by programming the ADS1299 to operate in single-ended mode.

The reason for including the MPU-6050 was to allow the device to track head/body movements, both for primary measurement and for decorrelating the EEG signals to eliminate motion artifacts. As primary measurements, linear and rotational acceleration of the device could be used as additional inputs or as fallbacks when the EEG signals could not be successfully analyzed, creating a hybrid BCI system.

This design also broke out the SPI signals (and the MPU-6050's I^2C signals) to the BBB headers. This was done primarily for debugging reasons, but it also opened the possibility of controlling those chips directly from the BBB without going through the microcontroller. In this case, a board intended purely for use as a BBB cape could be made cheaper by not populating the microcontroller.

The decision on how to reconcile the need for an independent power supply for standalone mode with the BBB's onboard power management unit and regulators was to continue to leave off the power regulation from the R1, and develop a simple add-on board with a battery charger and regulator. The 5V supply, previously taken from the BBB's 5V pins, was be replaced on the R1 by a 2.5V regulator, a switched-capacitor voltage inverter, and a -2.5V regulator, producing a split ±2.5V supply for the ADS1299. These are all derived from the 3.3V regulated supply generated off-board. This split-supply configuration means that the mid-supply bias will keep the patient at zero potential relative to the Brainboard's electronics, adding an additional measure of safety in comparison to the single-supply version's 2.5V bias.

Assembly Process

Assembly of R1 was greatly accelerated with respect to R0 due to the availability of a toaster oven, solder paste, and a paste stencil for the top side of the board. The greatest difficulty came in aligning the stencil over the board's exposed pads, since the lack of a paste printing machine meant this alignment had to be made by hand. This was still far easier than the hand-soldering process, and once the board was aligned, it could be carefully taped in place under the stencil, making the paste application process easy. Solder paste was scraped across the surface of the stencil using an old magnetic keycard, then excess paste was scraped away and saved in a small storage container for reuse. The wet paste could be easily wiped away with isopropyl alcohol and replaced in the event of a mistake, although none occurred. Each component was hand-placed onto the paste-covered pads, and then the board was heated at 450º F for approximately 15 minutes, or until inspection revealed that the paste had silvered and reflowed. Fine-pitch components like the UC3L and ADS1299 required some minor touch-up work with the iron due to bridging, but overall this process was significantly easier and faster than before.

The bottom side of the board contained only the TVS diodes and their attendant capacitors, along with a few solder jumpers. It was thus not cost-effective to order a stencil for this side, and it was soldered by hand in the same manner as R0. The larger pitch of these components and their small number meant that this process took only a few minutes.

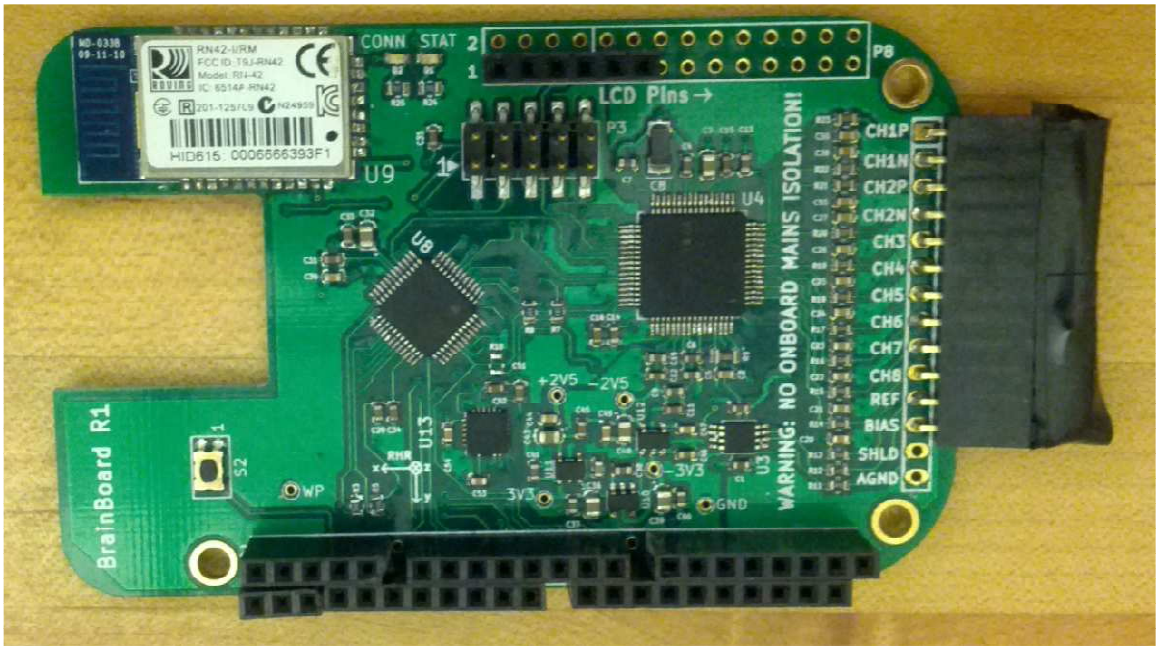


Figure 2: Brainboard R1

Software and Firmware

Firmware development to accommodate the MPU-6050 in R1 was most easily accomplished using Atmel's TWI[1] library for the UC3L, but this introduced some setbacks. The library was hard-coded to use a blocking interrupt setup: TWI interrupts would be enabled to priority level 1 at the beginning of each function in the library (and initialized in its initialization function), and each function would wait for the interrupt to trigger before returning. This meant firstly that any lower-priority interrupts would be masked by the TWI functions, and secondly that any higher-priority interrupts would themselves mask the TWI functions, causing them to hang as they waited for the interrupt flag. This had to be avoided by (1) ensuring that higher-priority interrupt code sections were rapidly executed entirely within their respective interrupt service routines, and (2) setting no interrupts to any priority lower than 1.

Since inertial readings from the MPU-6050 need only be sampled at a low rate relative to EEG signals (e.g. 10 Hz vs. 250+ Hz), it was unnecessary to code a separate interrupt for the MPU-6050's own "data ready" signal (INT). Instead, this signal is simply polled at each ADS1299 DRDY interrupt. Incidentally, this made routing the INT signal much simpler on the PCB, since it could connect to any general purpose I/O instead of one of the UC3L's limited number of external interrupt pins. This had the disadvantage of preventing the inertial data from being read independently of EEG data; however, for this application, such operation was considered unnecessary.

Battery Adapter Board

[1] , TWI, or Two-Wire Interface, Atmel's I^2C-compatible bus protocol. Its modules are implemented as two discrete entities: TWIM, or Two-Wire Interface Master, and TWIS, or Two-Wire Interface Slave. In the Brainboard designs, the MCU functions as a TWI/I^2C master device only.

Although there does exist a cape for adapting the Beaglebone and BBB to use 4 AA batteries, such a solution is not ideal in terms of size and weight. In order to accommodate a prismatic lithium-polymer battery, a minimal second board was designed. This board incorporated a battery charging IC (Texas Instruments bq24074), 3.3V linear regulator (Texas Instruments TPS73601 adjustable LDO with appropriate feedback resistors), and pushbutton on/off controller (Linear Technology LTC2951-1). A lithium-ion/polymer battery fuel gauge IC (Maxim Integrated MAX17048) was added to the board layout, but not populated due to the relative priority of getting a functional power supply versus writing firmware to detect the battery's state of charge.

The bq24074 is a load-sharing battery charger, meaning that the device could potentially be used while charging (although this would pose a shock risk unless an isolated power supply was employed). It can be pin-configured to comply with USB current limits (100 or 500 mA) and also includes resistor-programmable fast-charge current, safety cutoff time, and input current limit for non-USB-compliant applications. The equations for calculating the necessary resistor values are given below [32]:

$$R_{ISET} = \frac{K_{ISET}}{I_{CHG}}$$

$$R_{ILIM} = \frac{K_{ILIM}}{I_{IN(MAX)}}$$

$$R_{ITERM} = \frac{I_{TERM} \times R_{ISET}}{0.030}$$

$$R_{TMR} = \frac{t_{MAXCHG}}{10 \times K_{TMR}}$$

In the above, R terms represent resistances, I terms represent currents, and K terms represent empirical constants given in the device datasheet. ISET indicates variables related to the fast charge current I_{CHG}, the highest current supplied to the battery and the primary current used to return the battery to full charge; ILIM indicates variables related to the input current limit $I_{IN(MAX)}$, the limit on the amount of current drawn from the external power supply used to charge the battery, which supersedes I_{CHG}; ITERM indicates variables related to the charge termination current I_{TERM}, the current threshold below which the charger turns off during the constant-voltage phase of charging, indicating charge completion; and TMR indicates variables related to the safety cutoff time t_{MAXCHG}, after which charging ceases even if the battery voltage hasn't been restored and/or charge current hasn't yet fallen below I_{TERM}. A typical charge cycle is represented in Figure 3 below, taken from Figure 39 in the device datasheet [32].

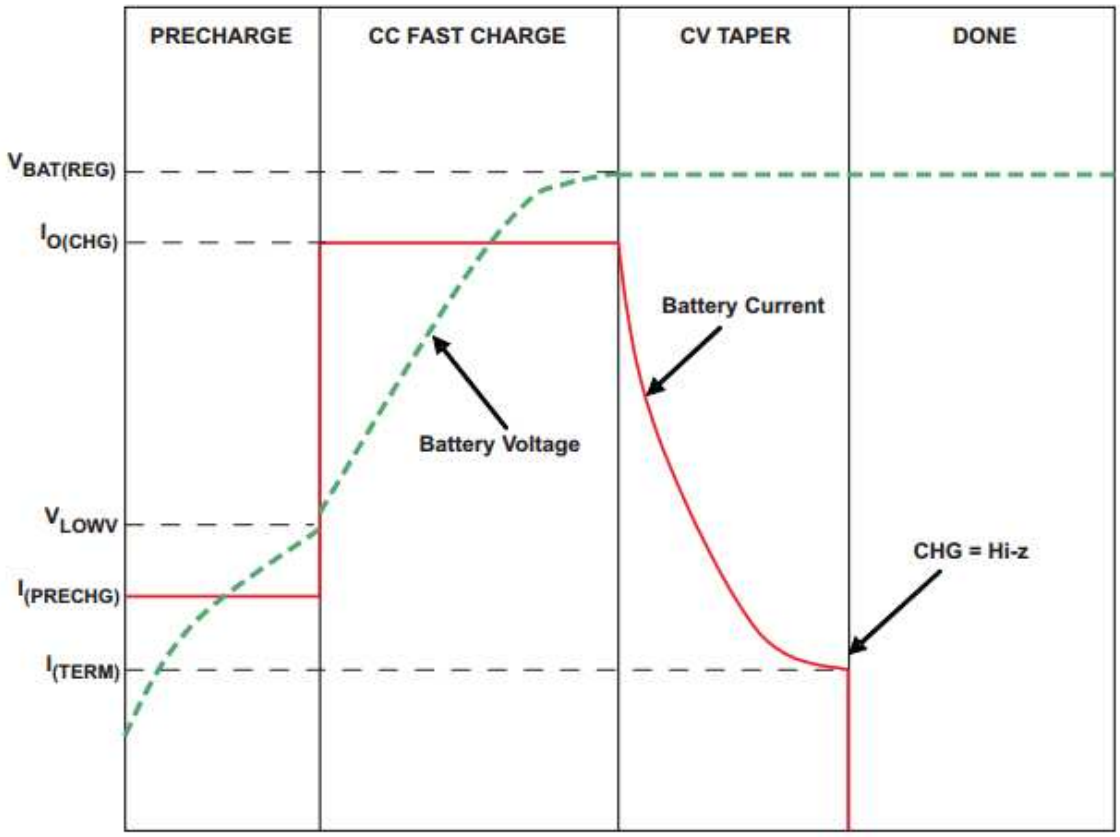


Figure 3: bq24074 lithium-ion/polymer battery charger, typical charge cycle. V_{LOWV} indicates the battery voltage at which the constant-current fast charge phase begins; $I_{O(CHG)}$ is synonymous with the larger of I_{CHG} and $I_{IN(MAX)}$.

The LTC2951 is designed to respond to a falling edge on its input, created by a momentary pushbutton, in two different ways. On power-up, the device waits for a single (debounced) falling edge on its input pin. This is considered the “power-on” press and it causes the EN (enable) pin to go high-impedance; a pull-up resistor brings the line high and enabled the system voltage regulator. When it detects a falling edge while EN is high-impedance, the device waits for a given amount of time, determined by an external capacitor (see below), and then checks the level of the input. If the input remains low, then the LTC2951 sends an interrupt signal on its ~INT pin, which may be

connected to a microcontroller or back to the ~KILL pin on the LTC2951. Regardless of the origin of the signal, a low level on the ~KILL pin causes the EN pin to switch back to a low state, disabling the voltage regulator and powering down the system. The equation for the external capacitor determining the amount of time the pushbutton must be held to turn off the system is given below [33]:

$$C_{OFFT} = 0.000156\ \mu F/ms\ \times (t_{OFFT} - 1\ ms)$$

The TPS73601 is an adjustable voltage regulator capable of sourcing up to 400 mA of current. The equation for determining the values of the external resistor network for a given regulated voltage is given as [34]:

$$V_{OUT} = \frac{R_1 + R_2}{R_2} \times 1.204$$

The resistance R_1 is connected between the output voltage and the regulator's feedback (FB) pin, while the resistance R_2 is connected from the FB pin to ground. The reason for not simply using a 3.3V regulator was to allow the board to potentially supply other voltages for different contexts besides the Brainboard R1 design.

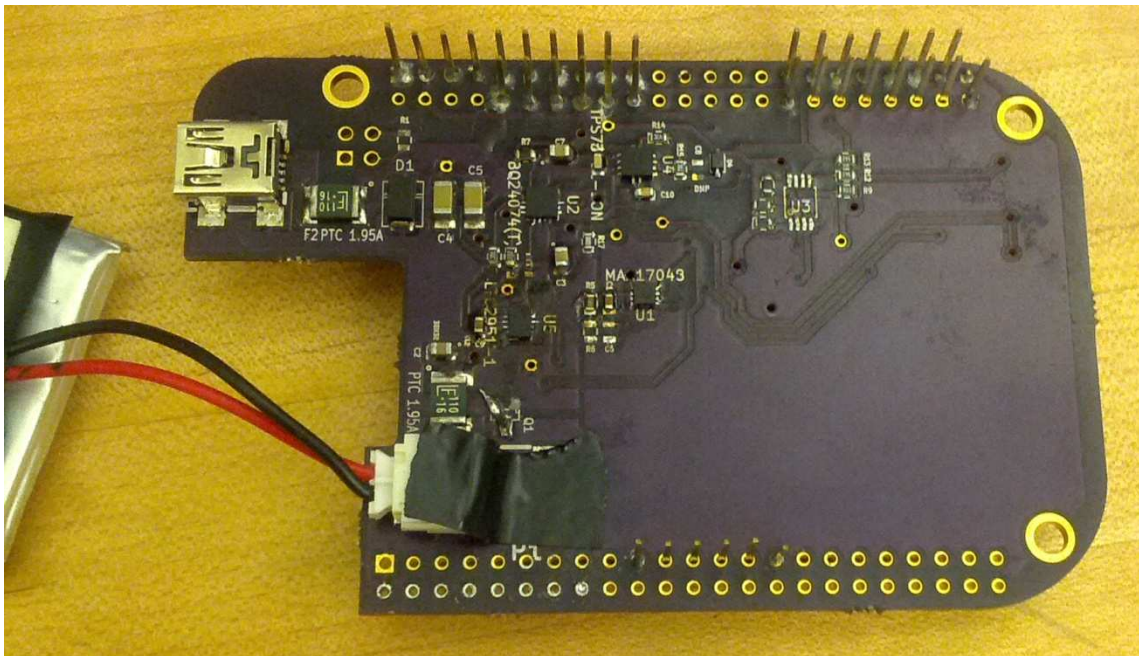


Figure 4: Battery management expansion board

Design Differentiation

Although the BBB form factor is significantly smaller than most EEG amplifiers, it still took up more space than necessary when the Brainboard was acting in a standalone capacity. In order to miniaturize the design even further, the Brainboard was differentiated into two separate versions: Brainboard R2, which was a pure BBB cape with no on-board microcontroller or Bluetooth radio, instead relying on the BBB and a Bluetooth dongle for these functions; and Brainboard LW (Lite-Wireless), which was approximately 2/3 the size of the original but could no longer plug directly onto the BBB. Since the lab's work had become more focused at this point on developing the smallest possible device, the remaining effort in this thesis focused on the Brainboard LW. Validation of the Brainboard R2 will be left for future work.

Final Design: Brainboard LW

The original BBB cape form factor was modified to take up less space. Its original 3.4” length was reduced to 2.4”, and both the notch for the Ethernet jack and the side headers were removed to allow more space for components. More components were also placed on the bottom side of the PCB.

Since there was more room on the board now, battery charging circuitry originally included on the battery adapter board was incorporated into LW. The pushbutton controller and its associated power button were replaced with a slide switch, and the fuel gauge replaced with a comparator (Linear Technology LTC1998), to simplify the implementation and reduce component count. The LTC1998 is specifically designed for low-battery detection on lithium-ion batteries and includes an on-chip reference voltage. It uses an external resistor network to set threshold voltage and hysteresis as given below [35]:

$$R_{TOTAL} = R_1 + R_2 + R_3 = \frac{4.2V}{I_R}$$

$$R_1 = R_{TOTAL}\left(\frac{5V}{V_{BATT.Th} + V_{HYST}} - 1\right)$$

$$R_2 = R_{TOTAL}\left(\frac{5V}{V_{BATT.Th}} - 1\right) - R_1$$

$$R_3 = R_{TOTAL} - R_1 - R_2$$

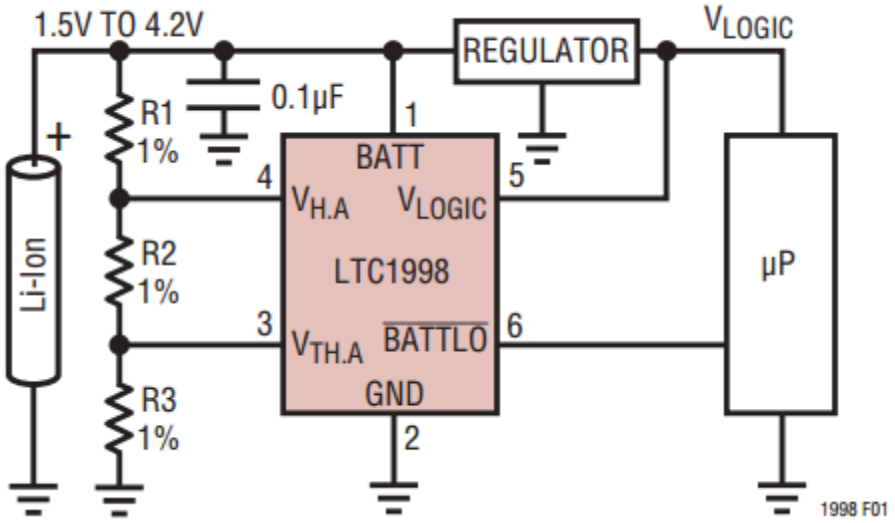


Figure 5: LTC1998 configured as low battery threshold detection with hysteresis

The network was designed such that the comparator had a threshold $V_{BATT.Th}$ of 3.1V with hysteresis V_{HYST} of 100mV, giving an upper (rising) trigger level of 3.2V and a lower (falling) trigger of 3.0V. The maximum allowable resistor current I_R (which drains the battery) was selected as 1 µA. Tripping the comparator causes its ~BATTLO pin to sink current from an LED, providing a visual indication of the low battery status. At this point the 3.3V regulator will already be sagging; however, all on-board devices support operation down to 1.8V except for the RN-42. The 3.0V trigger provides adequate time to plug in the device before battery voltage falls below the 3.0V minimum recommended level for the RN-42 power supply. The ~BATTLO signal was also connected to an external interrupt pin on the UC3L, providing the possibility of a controlled power-down procedure or a message to the host computer indicating a low battery (however, this was not implemented in firmware as of this writing).

One minor design change was changing the MPU-6050's INT pin to an actual interrupt-capable pin on the UC3L. Although not necessary for the application at hand, it might prove useful in the future to rewrite the firmware such that the inertial measurements can be taken independently from the EEG measurements (i.e., with the ADS1299 powered down or otherwise not sampling).

This design necessitated double-sided reflow, which brought up concerns about already-soldered parts falling off the board during the second pass, and I was prepared to do repairs by hand after the reflow process was complete. However, the solder surface tension was strong enough that these worries proved unfounded.

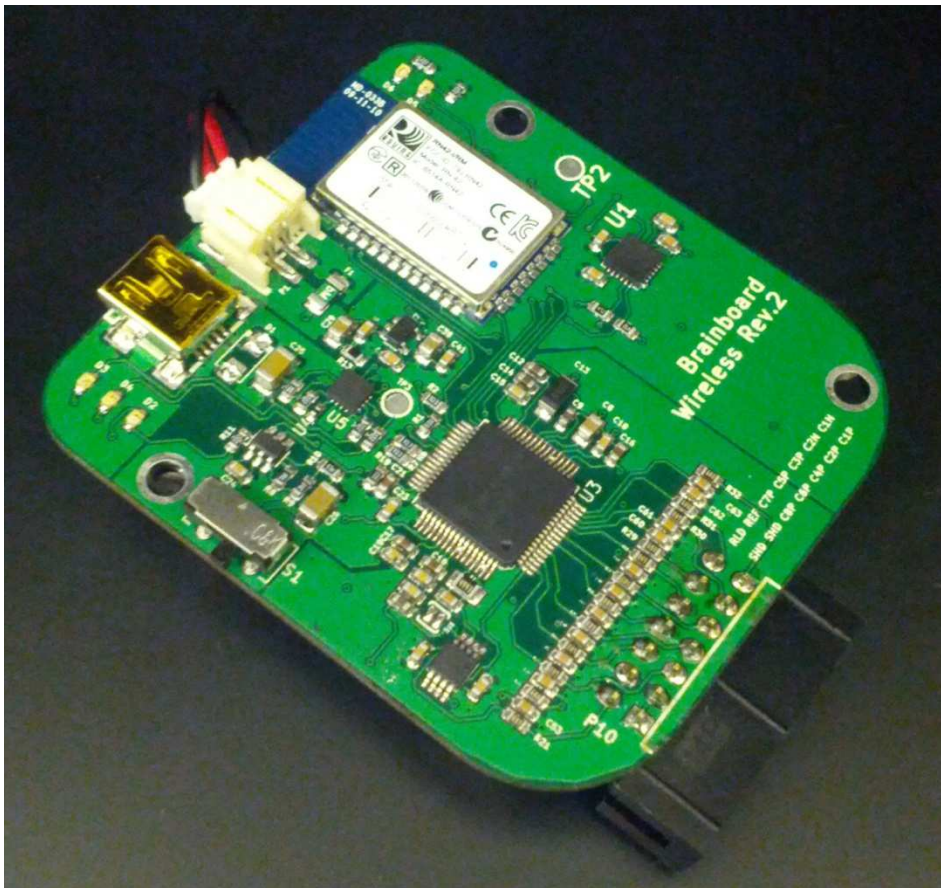


Figure 6: Brainboard LW

Results and Discussion

Signal Quality

The 0.1” pin socket electrode input to the Brainboard LW was broken out to 1.5 mm touch-proof connectors for compatibility with the lab’s electrodes. The electrodes used for signal quality comparison with a professional amplifier were actiCAP active electrodes (Brain Products GmbH, Munich, Germany), chosen to minimize potential signal degradation effects due to electrode problems. Electrodes were placed onto the subject’s scalp at positions Fz, Cz, Pz, Oz, C3, C4, and P3; however, inputs were recorded simply as channels 1-7 and their actual scalp locations were not tracked for

subsequent data analysis, since the only analysis being done was on differential signal quality between the two amplifiers.

EEG data were collected simultaneously with both the Brainboard LW and a professional DC-coupled amplifier, the V-Amp 16 (Brain Products GmbH, Munich, Germany) using a set of compatible signal splitters for the touch-proof connectors. The V-Amp was connected via USB to a laptop running BCI2VR, while the Brainboard was wirelessly connected to a separate desktop computer, also running BCI2VR. Due to data collection problems with the V-Amp 16, only channels 3-6 were obtainable; the remaining channels were inexplicably corrupted. The subject was instructed to relax, blink his eyes, and close his eyes, so as to present eye movement artifacts and alpha rhythm for comparison.

After collection, data were post-processed in MATLAB. The two signals, although simultaneously recorded, were time-shifted relative to one another since the “record” button could not be pressed simultaneously on both host PCs. This time shift was empirically determined to be 31.2 ms and was corrected. The spectrum up to 100 Hz (Welch’s modified periodogram, 600-point FFT with 600-sample windows and 50% window overlap) was calculated for both records and compared. Next the mean value was subtracted from each record to eliminate the effect of electrode bias and inherent amplifier offset, which is expected to vary between any two amplifiers and electrode setups. RMS error between the V-Amp and the Brainboard data was then calculated, both with and without a 60-Hz notch filter (8th-order Butterworth, Q = 10). RMS errors for each channel in the time and frequency domains are shown in Table 3. The quantified level of 60-Hz noise pickup is given in Table 4.

The frequency response of the Brainboard LW was estimated by applying sinusoids from 0 to 300 Hz at 9 mV peak-to-peak, sampled at 1000 Hz, and recording the attenuation at the output. Output amplitude was converted to decibels referenced to the 9 mV input. The plot shows a reasonably flat frequency response out to 50 Hz, with a -3 dB point at approximately 153 Hz.

Board-level noise was measured by shorting the input pins together and measuring for 10 seconds. From these data, RMS level was calculated, and then this was converted to a peak-to-peak value using a conversion constant of 6.6 as given in the ADS1299 datasheet. Input-referred noise levels were measured at a gain of 24, the gain at which all other tests besides frequency response were performed. Mean V_{rms} and V_{pp} for the Brainboard were 0.409 µV and 2.67 µV, respectively. These are higher than the V-Amp's reported specs of less than 1 µVpp noise [36], but not egregiously so. The measured noise is likely dependent on ambient conditions, as well, and these were not optimal in the lab, given the large number of active computers and other RF radiators. The noise spectrum is shown below. A close-up of the 0-65 Hz range, which is the range given in the ADS1299 datasheet for specified performance, is also shown. The response is almost flat over this range, indicating acceptably low distortion of acquired EEG.

Channel	Unfiltered time-domain RMSE (µV)	Filtered time-domain RMSE (µV)	Frequency-domain RMSE (dB)
3	23.5	23.2	2.32e-11
4	12.6	10.4	2.32e-11
5	20.3	19.5	1.35e-11
6	26.7	25.9	8.68e-11

Table 3: RMS error between Brainboard LW and Brain Products V-Amp 16.

Device	Channel	Unfiltered 60-Hz noise (dB)
Brainboard LW	3	-106
	4	-111
	5	-108
	6	-106
V-Amp 16	3	-134
	4	-101
	5	-104
	6	-103

Table 4: 60-Hz noise comparison between Brainboard LW and V-Amp 16.

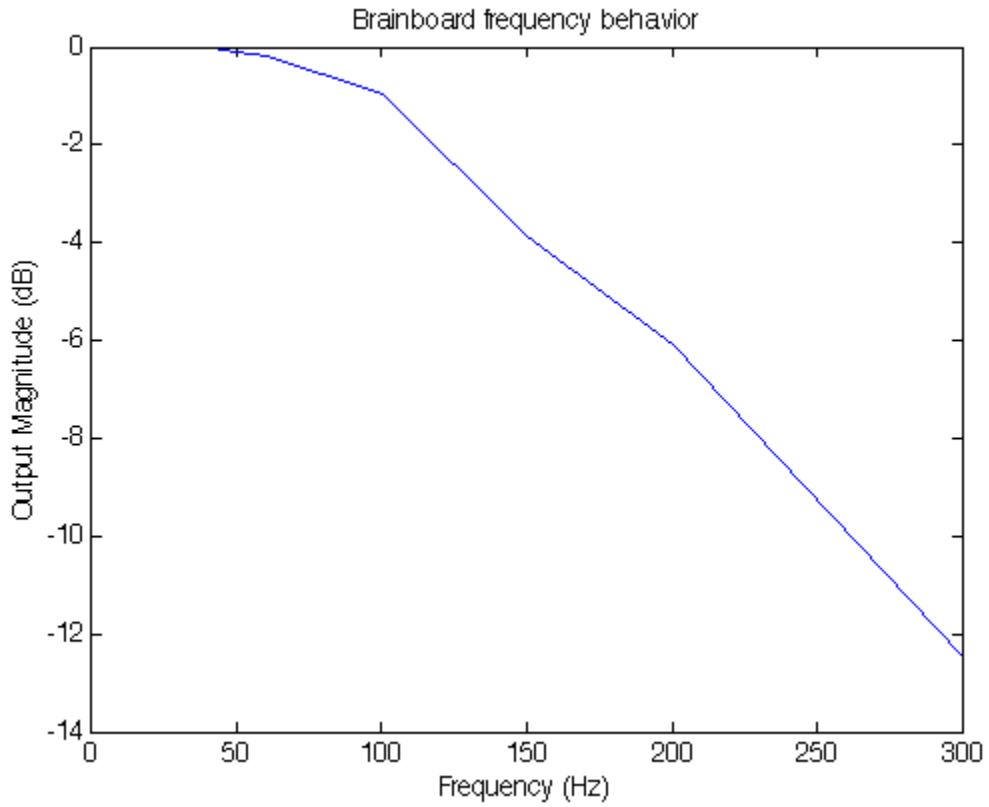


Figure 7: Magnitude response to swept sinusoidal input

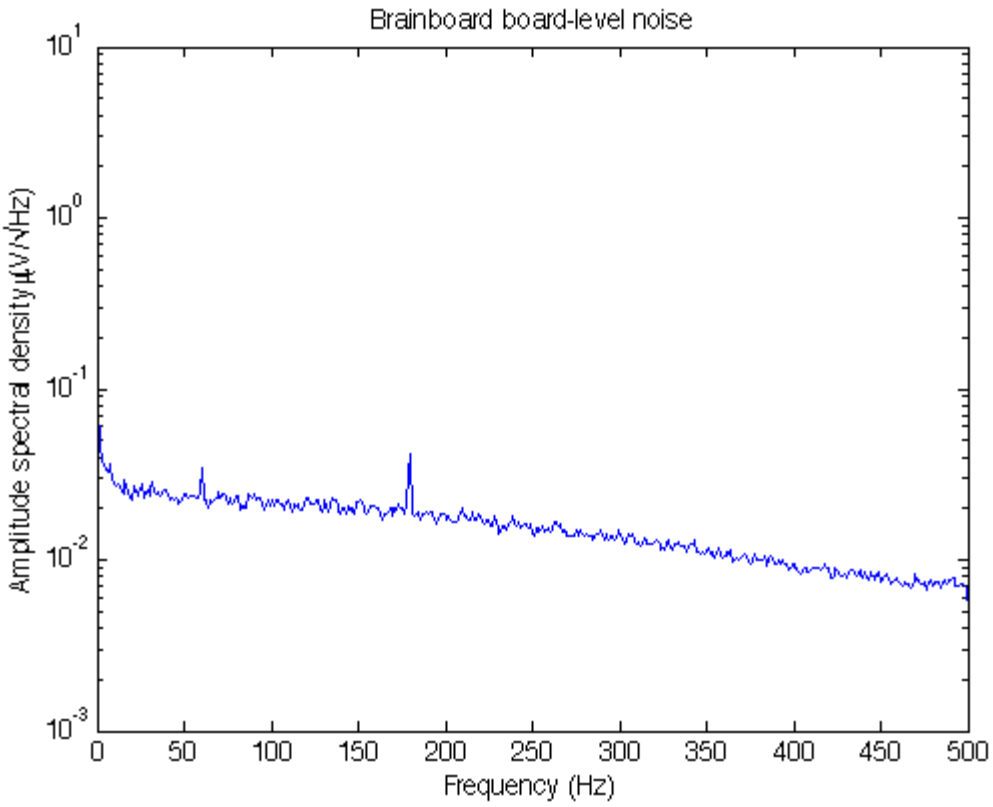


Figure 8: Noise spectrum for shorted inputs

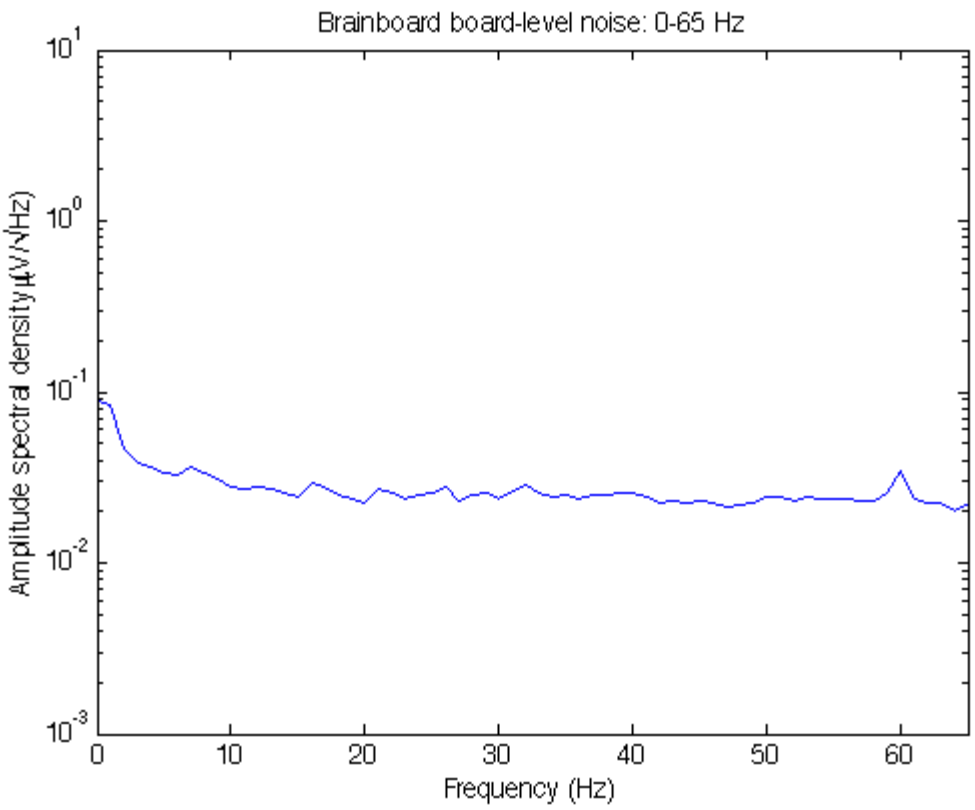


Figure 9: Noise spectrum at ADS1299 specified range

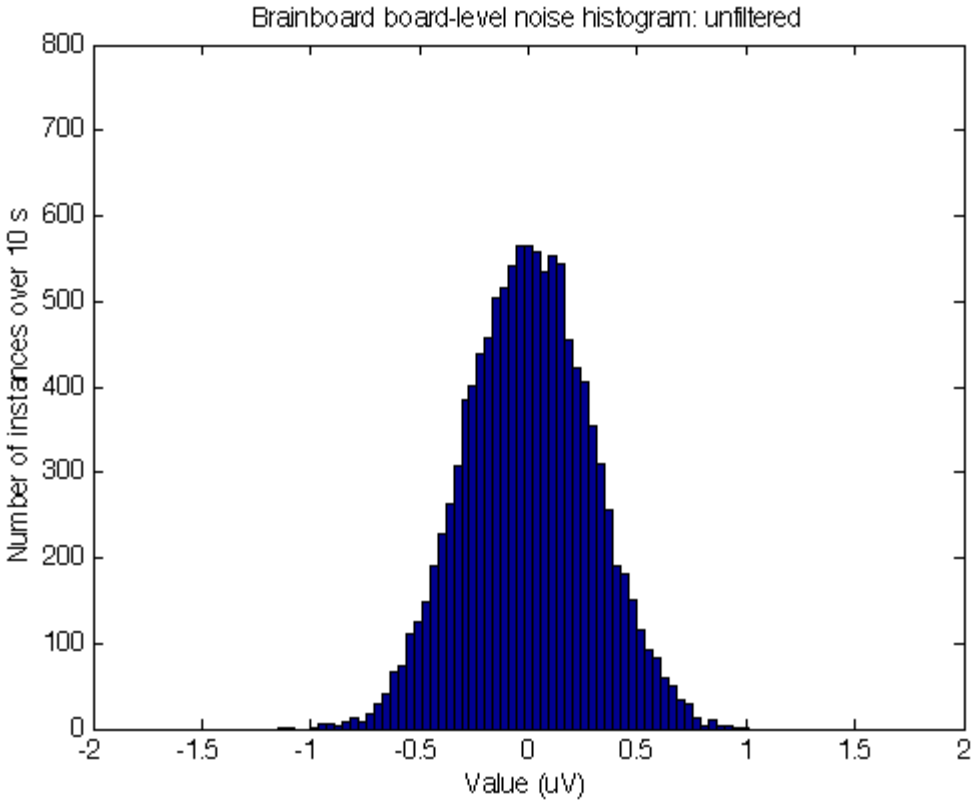


Figure 10: Unfiltered noise histogram

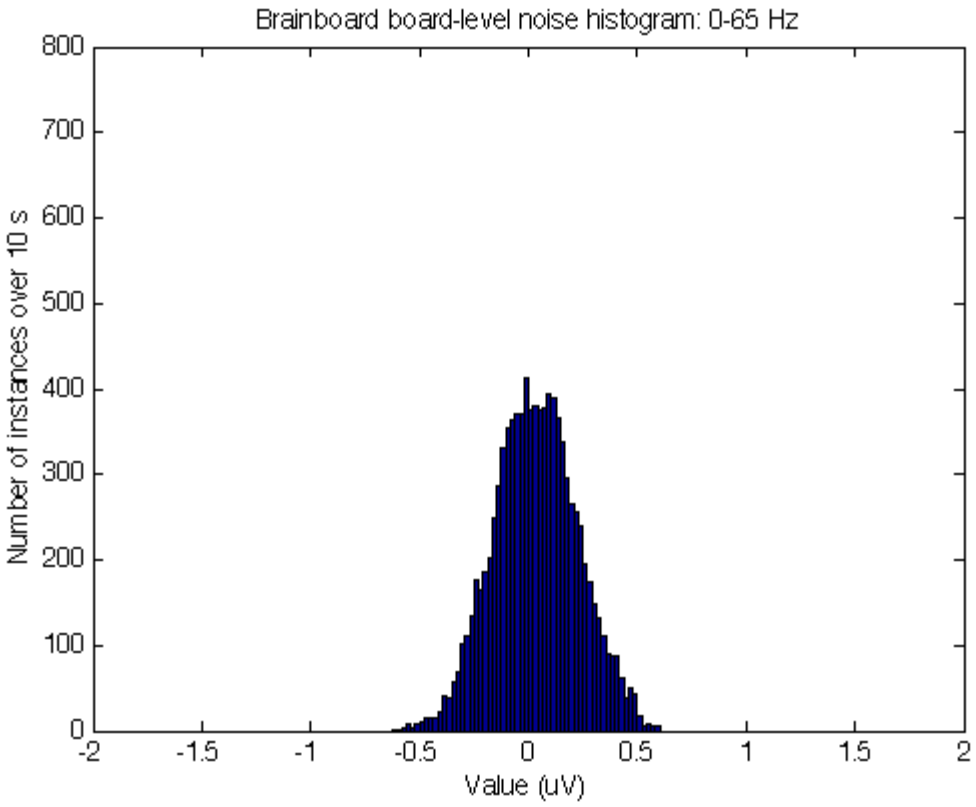


Figure 11: Noise histogram after 0-65 Hz low-pass filter

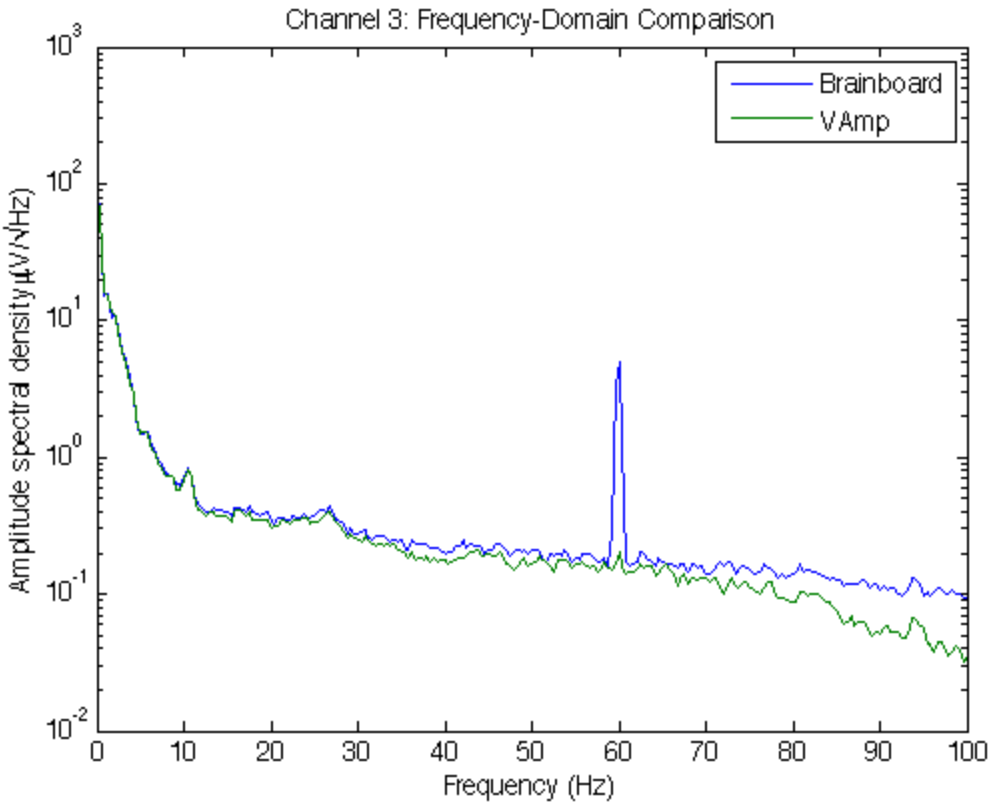


Figure 12: Channel 3 amplitude spectrum comparison

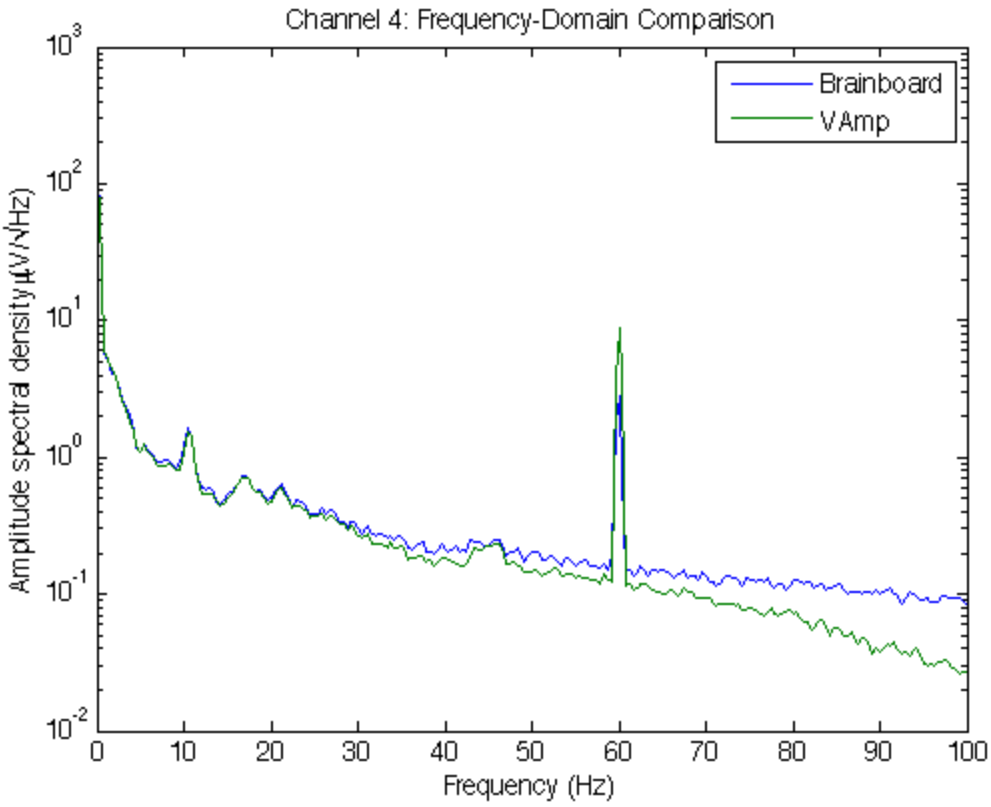


Figure 13: Channel 4 amplitude spectrum comparison

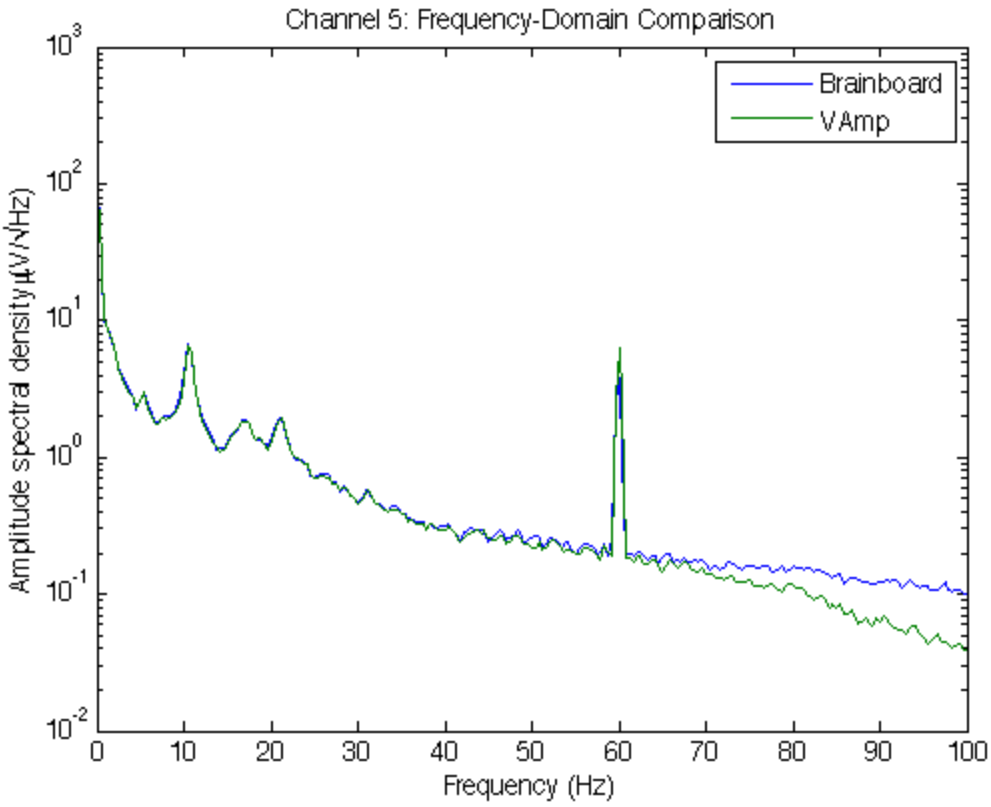


Figure 14: Channel 5 amplitude spectrum comparison

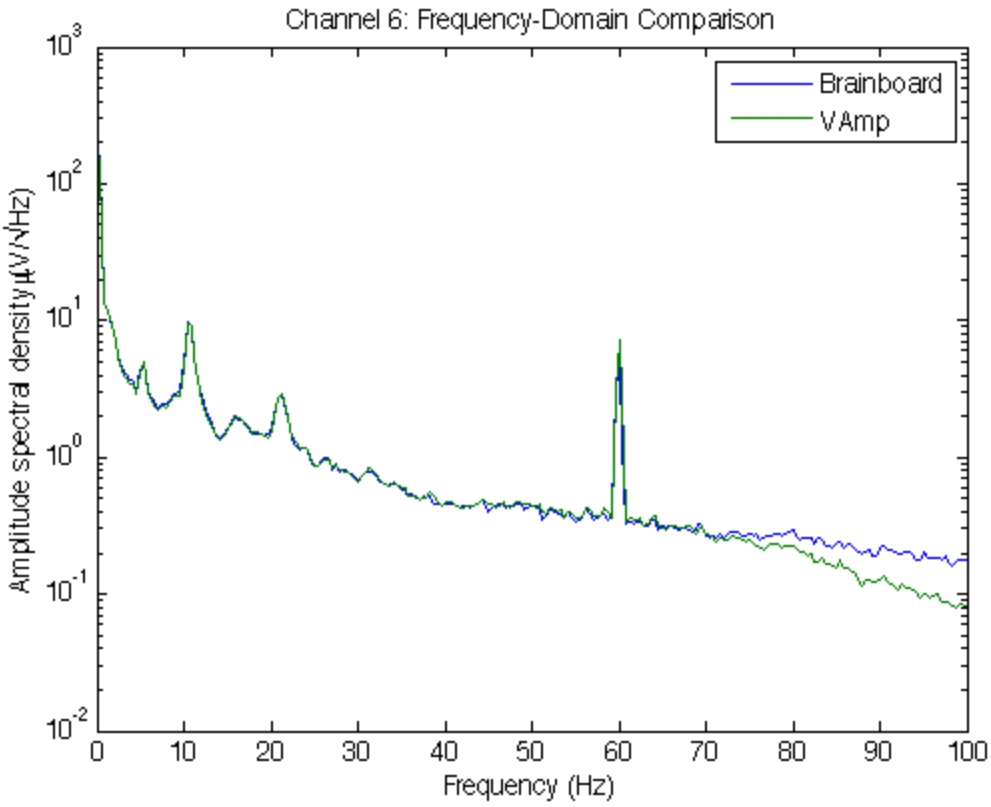


Figure 15: Channel 6 amplitude spectrum comparison

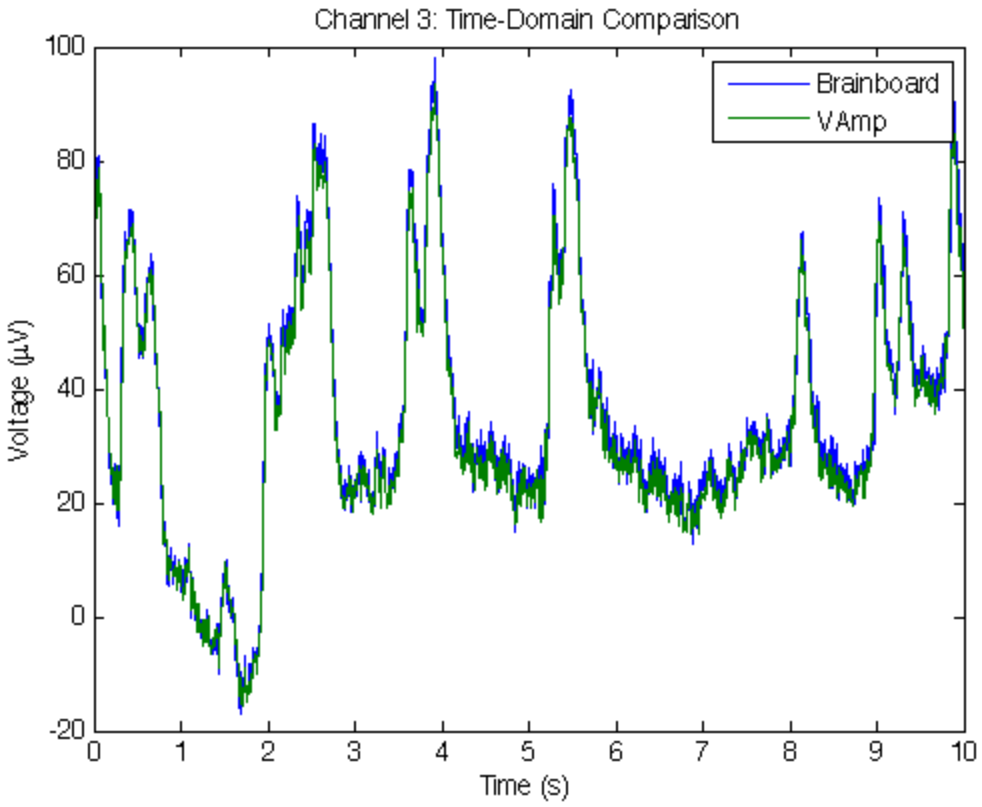


Figure 16: Channel 3 time-domain comparison

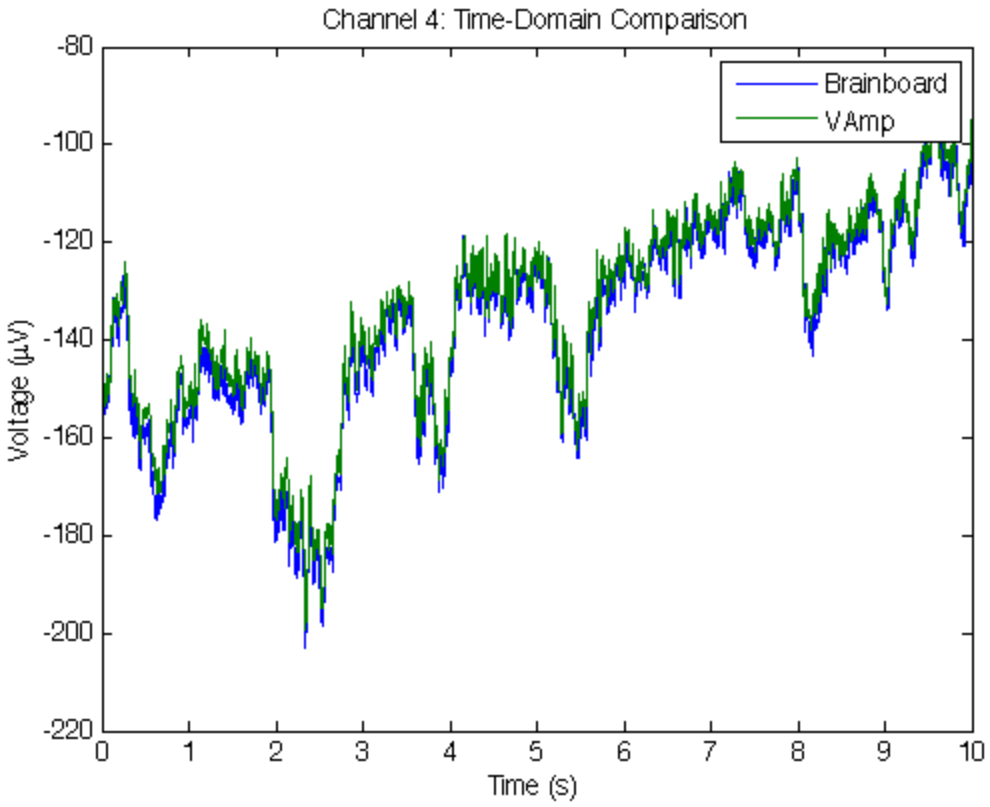


Figure 17: Channel 4 time-domain comparison

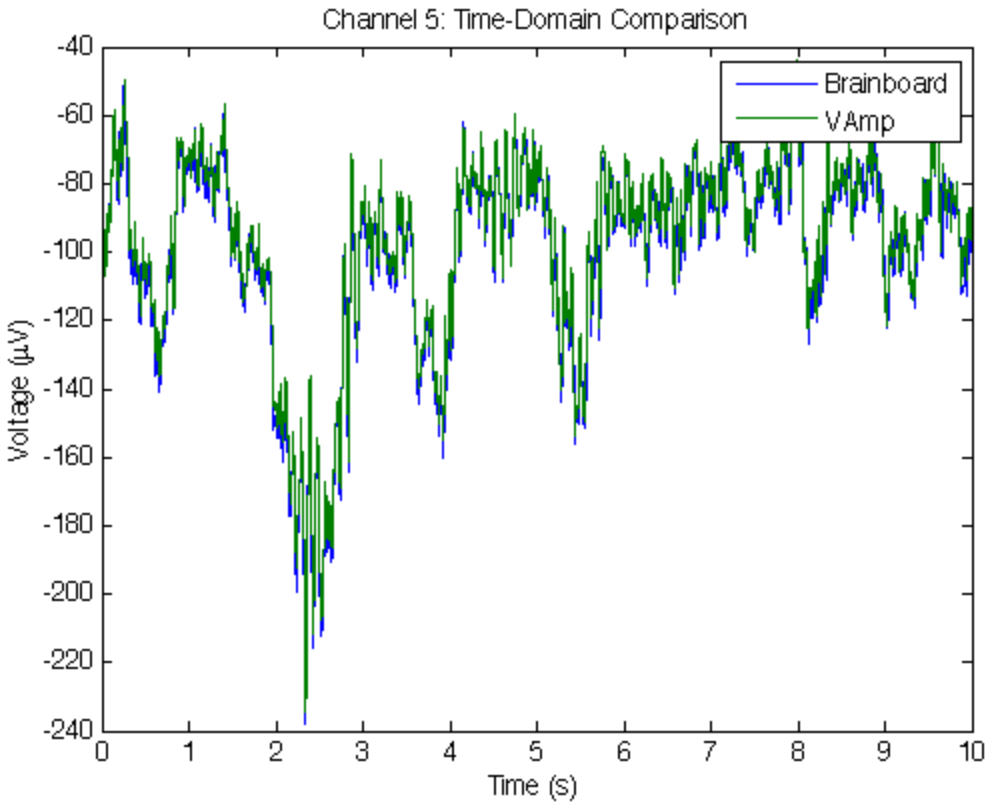


Figure 18: Channel 5 time-domain comparison

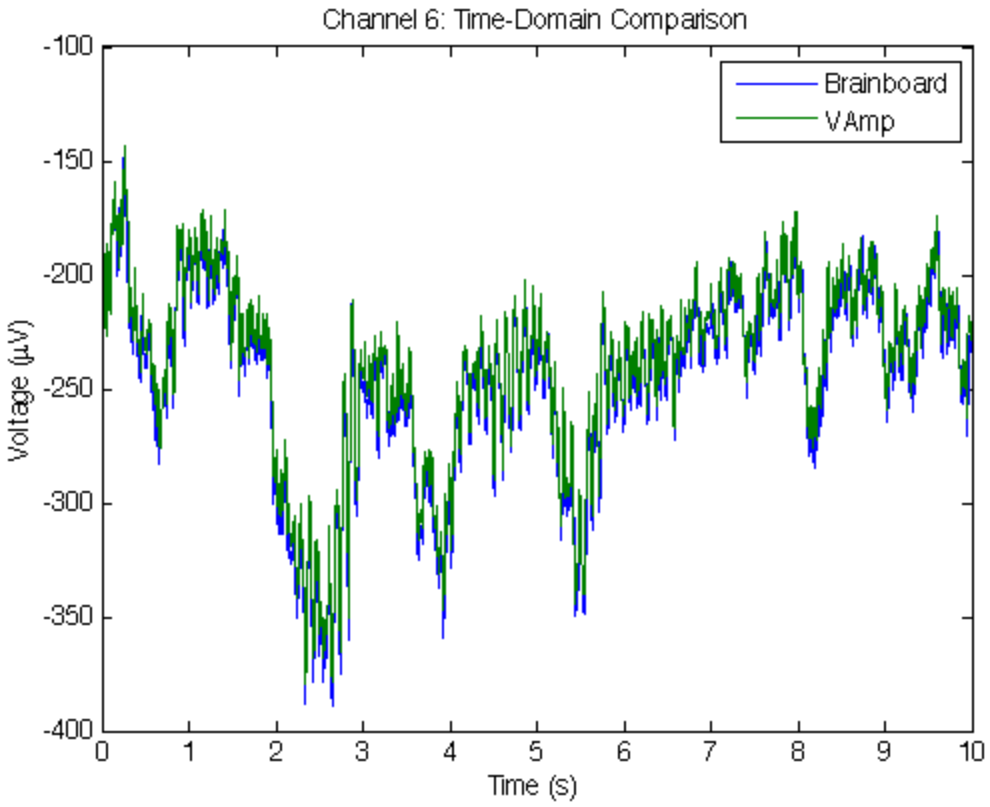


Figure 19: Channel 6 time-domain comparison

The Brainboard LW has identical performance to the V-Amp up to around 70 Hz, at which point its attenuation drops off more shallowly. The V-Amp likely has sharper analog filtering than the Brainboard, which only uses first-order anti-aliasing filters at 42 kHz. Mains pickup is comparable between the two devices. Noise performance might be improved through design of an appropriate metal enclosure for the analog portion of the device. It is unclear what caused the anomalously low 60-Hz power on the V-Amp's channel 3. It's possible that the other values are in fact anomalously high and the electrodes were poorly plugged into the amplifier, or that noise coupled between the Brainboard and the V-Amp due to the simultaneous recording method.

The Brainboard's noise is shown to have an approximately Gaussian distribution, confirming the approximately constant power spectral density observed in the spectra above.

Power Consumption

Current consumption was tested in different device states using a handheld DMM placed in series with the battery through an intermediate plug and jack. This information is shown in Results below. The maximum power consumption of 85 mA took place, as expected, during continuous data streaming at 250 Hz. This figure would theoretically allow just under 12 hours of run time on a 1000 mAh battery such as the one employed for testing; however, this was not verified empirically due to the impracticality of observing the device for such a long period of time. A 12-hour continuous run time should far exceed what is needed for the lab's experiments, but it isn't quite sufficient for, say, an all-day-wear prosthetic. Such an application would necessitate on-board processing of some kind, whether data analysis or compression, in order to reduce the radio's transmission duty cycle.

Reducing the power consumption of the device is possible on multiple levels due to the scalable operation of different components on the board. For example, the MPU-6050 can power-off its gyroscope, its accelerometer, or both (although the accelerometer consumes negligible power); the RN-42 may duty cycle its radio operation according to the Bluetooth SNIFF protocol [37]; the ADS1299 can power-down some of its channels or go into standby mode; the UC3L may go into a number of software-controlled sleep modes. The applicability of these different power consumption reduction methods depends on how the Brainboard is to be used: for example, a pure

data-streaming mode like what was evaluated in this thesis would likely require all of the devices to be operating at full capacity except perhaps for the MPU-6050. Radio power consumption could be reduced by implementing a compression algorithm, although at the expense of some extra power consumed by the MCU. The MCU might also be operated at a lower clock speed, although this might affect its ability to efficiently perform the compression algorithm. One final suggestion is to keep the Bluetooth radio in low-power sleep mode, buffer some of the data, and then send it in a burst transmission over the Bluetooth link before returning to sleep.

As an investigation into the feasibility of this latter option, let us consider the following. The RN-42 has two sleep modes: standard sleep, at which the module consumes about 2 mA, and deep sleep, at which it consumes 300 µA. The module consumes about 30 mA when connected and 45 mA when transmitting, so this is a significant savings. There is a catch to the deep sleep, however: it has a 5 ms worst-case wakeup time [38], corresponding to 1.25 sample periods at 250 Hz. The RN-42 then stays awake for exactly 1 second of inactivity before returning to deep sleep, meaning that the microcontroller must buffer at least 1 second of data, or the module will never return to sleep! Moreover, the RN-42 isn't even asleep during this 1 second, limiting our power savings to 15 mA. Since 1 second of 8-channel sampling corresponds to 8000 bytes of 32-bit-encoded 24-bit data, already half the RAM on the device, we will have to settle for the higher-power sleep mode.

Each packet of sent data consists of 18 non-EEG bytes, along with 24 EEG bytes per sample (the current implementation, with no accumulation of samples in the buffer, thus has 42 bytes per packet). Assuming no overhead and 115,200 bit/s transmission

rate to the RN-42, each packet will take 1.25 ms, plus 1.67 ms/sample, to offload. If only one sample is sent, this will take 2.92 ms, which is already almost as long as the sample period. The current consumption here is:

$$\frac{(30\ \text{mA} \times 1.08\ \text{ms}) + (45\ \text{mA} \times 2.92\ \text{ms})}{4\ \text{ms}} = 40.95\ \text{mA}$$

Now let us presume that we keep the RN-42 asleep for 124 ms while buffering 31 samples, initiate wakeup and wait another 5 ms (buffering another sample in the process), and then send data for (1.25 + 1.67 x 32 = 54.69 ms). This gives an average current of

$$\frac{(2\ \text{mA} \times 124\ \text{ms}) + (45\ \text{mA} \times 59.69\ \text{ms})}{183.69\ \text{ms}} = 14.75\ \text{mA},$$

a 64% reduction in power.

Some common device states with their corresponding current draw from the battery are shown in Table 5. Note that there is actually very little difference in power draw between enabling and disabling the RN-42's LED drivers. This is due to the relatively large resistors placed in series with the LEDs (2.2kΩ). The driving voltage is 3.3V, resulting in only 1.5 mA of LED current.

ADS1299 Status	# Active Channels	RN-42 Status	RN-42 LEDs	Gyro	I (mA)
Power-down	N/A	Reset	N/A	Off	30.1
Standby	N/A	Idle	Disabled	Off	36.8
Normal, bias on	1	TX @ 250 Hz	Disabled	Off	71
Normal, bias on	8	TX @ 250 Hz	Disabled	Off	82
Normal, bias on	8	TX @ 250 Hz	Disabled	On	85
Normal, bias on	8	Idle	Enabled	On	56
Normal, bias on	8	TX @ 250 Hz	Enabled	On	86.5

Table 5: List of common device states and corresponding current consumption

Cost

The final cost of a single Brainboard LW board was $135.01 for components, $17 for the bare PCB, $10 for a stencil, and $12.95 for 50 grams of solder paste. Note that this doesn't include assembly costs (I assembled it myself) or the cost of an enclosure, or electrode costs. These would add to the price of the final product if it were sold. The component BOM is given below:

Item Value	Quantity	Manufacturer	PN	Distributor	Ext Price
Capacitor 1nF	1	Kemet	C0603C102K3RACTU	DigiKey	$ 0.10
Capacitor 2.2nF	1	Kemet	C0603C222K5RACTU	DigiKey	$ 0.10
Capacitor 4.7nF	11	Kemet	C0603C472K5RACTU	DigiKey	$ 0.33
Capacitor 0.01uF	4	Kemet	C0603C103K5RACTU	DigiKey	$ 0.40
Capacitor 0.1uF	21	Kemet	C0603C104K4RACTU	DigiKey	$ 0.46
Capacitor 1uF	12	Kemet	C0603C105K8PACTU	DigiKey	$ 0.52
Capacitor 2.2uF	5	Kemet	C0603C225K8PACTU	DigiKey	$ 0.95
Capacitor 10uF	7	Kemet	C0805C106Z8VACTU	DigiKey	$ 1.12
Capacitor 100uF	1	Vishay Sprague	TL8A0107M010C	DigiKey	$ 1.35
USB Mini-B connector	1	EDAC Inc	690-005-299-043	DigiKey	$ 0.41
Zener diode	1	Littelfuse Inc	SMAJ7.0A	DigiKey	$ 0.43
Battery-low LED	1	Kingbright Company LLC	APT1608SRCPRV	DigiKey	$ 0.17
Power-good LED	1	Kingbright Company LLC	APT1608SGC	DigiKey	$ 0.16
Charge LED	1	Kingbright Company LLC	APT1608SRCPRV	DigiKey	$ 0.17

Status LED	1	Kingbright Company LLC	APT1608SRCPRV	DigiKey	$	0.17
Connected LED	1	Kingbright Company LLC	APT1608SGC	DigiKey	$	0.16
Resettable Fuse	1	Bel-Fuse	0ZCA0050FF2G	DigiKey	$	0.16
Battery connector	1	JST Sales America Inc	S2B-PH-SM4-TB(LF)(SN)	DigiKey	$	0.56
Programming connector	1	FCI	95278-101B10LF	Mouser	$	0.86
Electrode connector	1	Sullins Connector	SBH11-PBPC-D07-RA-BK	DigiKey	$	0.87
Resistor 1.18k	1	Yageo	311-1.18KHRCT-ND	DigiKey	$	0.10
Resistor 2.2k	6	Stackpole Electronics	RMCF0603JT2K20	DigiKey	$	0.12
Resistor 4.7k	2	Yageo	311-4.7KGRCT-ND	DigiKey	$	0.20
Resistor 4.99K	12	Stackpole Electronics	RNCS0603BKE4K99	DigiKey	$	5.70
Resistor 10K	6	Stackpole Electronics	RMCF0603JT10K0	DigiKey	$	0.12
Resistor 51.1k	1	Rohm Semiconductor	RHM51.1KCFCT-ND	DigiKey	$	0.10
Resistor 52.3k	1	Stackpole Electronics	RMCF0603FT52K3	DigiKey	$	0.04
Resistor 383k	1	Rohm Semiconductor	RHM383KCFCT-ND	DigiKey	$	0.10
Resistor 499k	1	Stackpole Electronics	RNCS0805BKE499K	DigiKey	$	0.77
Resistor 523k	1	Rohm Semiconductor	RHM523KCFCT-ND	DigiKey	$	0.10
Power switch	1	C&K Components	JS202011AQN	DigiKey	$	0.39
Thermistor	1	Vishay BC Components	NTCS0402E3103JLT	DigiKey	$	0.25
MPU6050	1	InvenSense		DigiKey	$	14.89
LTC1998	1	Linear Technology		DigiKey	$	2.50
BQ24074(T)	1	Texas Instruments		DigiKey	$	2.98
ATUC3L064	1	Atmel	AT32UC3L064-AUT	DigiKey	$	8.70
BLUETOOTH-RN42	1	Microchip Technology	RN42-I/RM	DigiKey	$	18.32
OPA2376	1	Texas Instruments	OPA2376AIDGKR	DigiKey	$	2.73
ADS1299	1	Texas Instruments	ADS1299IPAG	DigiKey	$	56.70
TPS60403	1	Texas Instruments		DigiKey	$	1.13
TPS73233	1	Texas Instruments		DigiKey	$	1.79
TPS73225	1	Texas Instruments		DigiKey	$	1.79

TPS72325	1	Texas Instruments	DigiKey	$	3.24
TPD4E001DBV	3	Texas Instruments	DigiKey	$	2.80
			Total:	$	135.01

Table 6: Component bill of materials

Conclusion and Future Work

The Brainboard is a functional EEG and acceleration/rotation acquisition board with the advantage over commercial devices of being reprogrammable by anyone with a free copy of Atmel Studio and an AVR Dragon ($49 directly from Atmel). Its components, other than the microcontroller, are all popular among hobbyists and the open-source hardware community and thus have a wide range of free application code and use cases available on the Internet. Atmel provides free code libraries in the form of the ASF, which eases use of the Brainboard's microcontroller. However, it has a way to go before it can achieve the ease of use and low cost needed to make it a widely adopted device.

Outside of electronics development, one simple future project is a case for the PCB. This could be manufactured quite simply using a 3D printer or laser cutter, both of which are available in relatively inexpensive consumer forms.

Inclusion of a USB bootloader would vastly improve the ease of programming, since no awkward cable would need to be plugged into the JTAG port, and no AVR Dragon would need to be purchased. However, more importantly, I would recommend that future development based on the Brainboard design begin with porting to a different microcontroller. The UC3L, while capable of supporting the application for which it was used, is underpowered in many respects, such as RAM and available libraries, not to mention user support. Adoption of a microcontroller with Arduino IDE compatibility, or a

non-Arduino device with a large user support base such as the STM32 series, would be a huge step in the right direction. Since most of the Arduino-compatible platforms are 8-bit devices with drastically reduced memory capacity and clock speed versus the UC3L, the latter option would be preferable (although the recently announced Arduino Zero, which introduces support for an ARM Cortex-M0+ MCU running at up to 48 MHz and having 32 KB of SRAM, would be an excellent candidate once it is released [39]. A Cortex-M4 device, which has powerful DSP specializations and inherent support for ARM's own core-optimized DSP library, would be an even more attractive option; the Teensy 3.1 is such a device, and it has support for Arduino-style programming and almost all Arduino libraries, although it is a full PCB system rather than a chip and requires a proprietary bootloader [40]. A possible option would be redesigning the Brainboard as a Teensy expansion board. The Tiva microcontrollers from Texas Instruments also have a strong hobbyist following thanks to TI's low-cost ($12.99) Launchpad development boards, and they can be programmed in an Arduino-like IDE called Energia [41].

Further pursuing of the Beaglebone Black integration is another logical step, since this would drastically improve the Brainboard's BCI processing capability and ease of software development. The first step should be software development so that the already-existing R1 can interface with the BBB; this could come in the form of a modification of BCI2000 or Open-Ephys's GUI. The second step would be refining the R1 into a more streamlined, lower-cost product. This step is already underway with the design of the Brainboard R2, a stripped-down board meant to operate solely in conjunction with the BBB and a Bluetooth dongle. Ultimately, given the fully open-

source nature of the BBB, I would like to develop a completely customized board incorporating components from the Brainboard and BBB. This would also allow bypassing the relative difficulty of running the BBB off of battery power, since unused parts of that board could be removed in this custom design (e.g. Ethernet, USB host, HDMI).

I envision the Brainboard eventually as a multi-embodiment ecosystem of devices with varying capabilities, ranging from simple data streaming to complex BCI processing. The simplest version could work with a basic 8-bit Arduino-compatible MCU, while intermediate versions might have a Cortex-M4 device capable of mid-level processing, and the most advanced ones would have a full-featured microprocessor like the Cortex-A8 on the BBB. The cost to the user of these devices could be minimized by making them interchangeable with a single sensor board containing the ADS1299 and MPU-6050, or whatever their successors might be. These would all be in a form factor small enough to allow them to be worn on the body.

The final roadblock toward mainstreaming EEG-based BCI systems is the inconvenience of traditional wet electrodes. Dry electrodes that can operate through hair remain expensive, and it is unclear whether or not they fully meet the wet gold standard. This would be the final direction to take: research into a user-friendly, easily donned and doffed electrode setup. Once this goal is reached, I believe EEG BCI will have finally entered into its own.

Unfortunately in some respects and fortunately in others, this project was begun on the eve of a flurry of activity regarding open-source neurophysiology and BCI platforms. Such recent projects as OpenBCI and Open Ephys—the former of which

even uses the same ADS1299 chip and acts as an Arduino expansion board—have a time-to-market and manpower advantage over Brainboard, but the success of Brainboard is less important than the community interest it was intended to build. The goal is commonplace, affordable brain-computer interfaces, and I believe this goal is not too far away.

References

[1] A. D. Walling, “Amyotrophic lateral sclerosis: Lou Gehrig’s disease.,” *Am. Fam. Physician*, vol. 59, no. 6, pp. 1489–96, Mar. 1999.

[2] H. Fischer, “U.S. Military Casualty Statistics: Operation New Dawn, Operation Iraqi Freedom, and Operation Enduring Freedom,” 2012.

[3] K. Ziegler-Graham, E. J. MacKenzie, P. L. Ephraim, T. G. Travison, and R. Brookmeyer, “Estimating the prevalence of limb loss in the United States: 2005 to 2050.,” *Arch. Phys. Med. Rehabil.*, vol. 89, no. 3, pp. 422–9, Mar. 2008.

[4] H. Berger, “Über das Elektrenkephalogramm des Menschen,” *Arch. Psychiatr. Nervenkr.*, vol. 87, no. 1, pp. 527–570, Dec. 1929.

[5] A. Chancellor, “Electroencephalography: maturing gracefully.,” *Pract. Neurol.*, vol. 9, no. 3, pp. 130–2, Jun. 2009.

[6] J. Santamaria and K. H. Chiappa, “The EEG of drowsiness in normal adults.,” *J. Clin. Neurophysiol.*, vol. 4, no. 4, pp. 327–82, Oct. 1987.

[7] T. Musha, Y. Terasaki, H. A. Haque, and G. A. Ivamitsky, “Feature extraction from EEGs associated with emotions,” *Artif. Life Robot.*, vol. 1, no. 1, pp. 15–19, Mar. 1997.

[8] G. Pfurtscheller, C. Brunner, A. Schlögl, and F. H. Lopes da Silva, “Mu rhythm (de)synchronization and EEG single-trial classification of different motor imagery tasks.,” *Neuroimage*, vol. 31, no. 1, pp. 153–9, May 2006.

[9] T. Ros, M. A. M. Munneke, D. Ruge, J. H. Gruzelier, and J. C. Rothwell, “Endogenous control of waking brain rhythms induces neuroplasticity in humans.,” *Eur. J. Neurosci.*, vol. 31, no. 4, pp. 770–8, Feb. 2010.

[10] “B-Alert X-Series EEG Systems - Advanced Brain Monitoring,” 2014. [Online]. Available: http://advancedbrainmonitoring.com/xseries/. [Accessed: 29-May-2014].

[11] “Emotiv | EEG System | Electroencephalography.” [Online]. Available: http://emotiv.com/. [Accessed: 05-Aug-2014].

[12] M. Duvinage, T. Castermans, T. Dutoit, M. Petieau, T. Hoellinger, C. De Saedeleer, K. Seetharaman, and G. Cheron, “A P300-based Quantitative Comparison between the Emotiv Epoc Headset and a Medical EEG Device,” in *Biomedical Engineering / 765: Telehealth / 766: Assistive Technologies*, 2012.

[13] H. Ekanayake, “Research Use of Emotiv EPOC.” [Online]. Available: http://neurofeedback.visaduma.info/emotivresearch.htm. [Accessed: 05-Aug-2014].

[14] “EEG Biosensor Solutions | NeuroSky.” [Online]. Available: http://neurosky.com/products-markets/eeg-biosensors/. [Accessed: 05-Aug-2014].

[15] S. Kravitz, “Hackers in Residence - Hacking MindWave Mobile - SparkFun,” 2014. [Online]. Available: https://learn.sparkfun.com/tutorials/hackers-in-residence---hacking-mindwave-mobile/all . [Accessed: 01-Jul-2014].

[16] C.-T. Lin, L.-W. Ko, M.-H. Chang, J.-R. Duann, J.-Y. Chen, T.-P. Su, and T.-P. Jung, “Review of wireless and wearable electroencephalogram systems and brain-computer interfaces--a mini-review.,” *Gerontology*, vol. 56, no. 1, pp. 112–9, Jan. 2010.

[17] M. Cheng, X. Gao, S. Gao, and D. Xu, “Design and implementation of a brain-computer interface with high transfer rates,” *IEEE Trans. Biomed. Eng.*, vol. 49, no. 10, pp. 1181–1186, Oct. 2002.

[18] J. M. P. Robert Matthews, Neil J. McDonald, Harini Anumula, Jamison Woodward, Peter J. Turner, Martin A. Steindorf, Kaichun Chang, “Novel hybrid bioelectrodes for ambulatory zero-prep EEG measurements using multi-channel wireless EEG system,” in *Foundations of Augmented Cognition*, L. M. R. Dylan D. Schmorrow, Ed. Springer Berlin Heidelberg, 2007, pp. 137–146.

[19] C.-T. Lin, L.-W. Ko, J.-C. Chiou, J.-R. Duann, R.-S. Huang, S.-F. Liang, T.-W. Chiu, and T.-P. Jung, “Noninvasive Neural Prostheses Using Mobile and Wireless {EEG},” *Proc. {IEEE}*, vol. 96, no. 7, pp. 1167–1183, Jul. 2008.

[20] “Analog, Embedded Processing, Semiconductor Company, Texas Instruments - TI.com.” [Online]. Available: http://www.ti.com/. [Accessed: 05-Aug-2014].

[21] “The ModularEEG.” [Online]. Available: http://openeeg.sourceforge.net/doc/modeeg/modeeg.html. [Accessed: 05-Aug-2014].

[22] “OpenBCI.” [Online]. Available: http://www.openbci.com/. [Accessed: 05-Aug-2014].

[23] “Arduino - Home.” [Online]. Available: http://www.arduino.cc/. [Accessed: 05-Aug-2014].

[24] “Raspberry Pi.” [Online]. Available: http://www.raspberrypi.org/. [Accessed: 05-Aug-2014].

[25] “BeagleBoard.org - black.” [Online]. Available: http://beagleboard.org/black. [Accessed: 05-Aug-2014].

[26] “Beagleboard:BeagleBone Capes - eLinux.org.” [Online]. Available: http://elinux.org/Beagleboard:BeagleBone_Capes. [Accessed: 05-Aug-2014].

[27] “BCI2000 | Schalk Lab.” [Online]. Available: http://www.schalklab.org/research/bci2000. [Accessed: 05-Aug-2014].

[28] G. Schalk, D. J. McFarland, T. Hinterberger, N. Birbaumer, and J. R. Wolpaw, “BCI2000: a general-purpose brain-computer interface (BCI) system.,” *IEEE Trans. Biomed. Eng.*, vol. 51, no. 6, pp. 1034–43, Jun. 2004.

[29] “Open Ephys — GUI.” [Online]. Available: http://open-ephys.org/#/gui/. [Accessed: 05-Aug-2014].

[30] Texas Instruments, “Low-Noise, 8-Channel, 24-Bit Analog Front-End for Biopotential Measurements.” 2012.

[31] M. Hallett, O. Bai, and C. Bonin, “Predicting Movement: When, Which and Where,” in *2007 IEEE/ICME International Conference on Complex Medical Engineering*, 2007, pp. 5–7.

[32] Texas Instruments, “1.5A USB-FRIENDLY Li-Ion BATTERY CHARGER AND POWER-PATH MANAGEMENT IC.” 2008.

[33] Linear Technology, “Pushbutton On/Off Controller.” 2005.

[34] Texas Instruments, “Cap-Free, NMOS, 400mA Low-Dropout Regulator with Reverse Current Protection.” 2010.

[35] Linear Technology, “2.5µA, 1% Accurate SOT-23 Comparator and Voltage Reference for Battery Monitoring.” 2001.

[36] “Brain Products GmbH / Products & Applications / actiCAP.” [Online]. Available: http://www.brainproducts.com/productdetails.php?id=4. [Accessed: 05-Aug-2014].

[37] Roving Networks Technical Staff, “Bluetooth Data Module Command Reference & Advanced Information User’s Guide.” Roving Networks, 2013.

[38] Roving Networks, “RN42/RN42N Class 2 Bluetooth Module.” 2013.

[39] “Arduino - ArduinoBoardZero.” [Online]. Available: http://arduino.cc/en/Main/ArduinoBoardZero. [Accessed: 05-Aug-2014].

[40] PJRC, “Teensy 3.1,” 2014. [Online]. Available: https://www.pjrc.com/teensy/teensy31.html. [Accessed: 01-Jul-2014].

[41] “TI LaunchPad Evaluation Ecosystem.” [Online]. Available: http://www.ti.com/ww/en/launchpad/launchpad.html?DCMP=TIMDC_in&HQS=TIMDC-launchpad. [Accessed: 05-Aug-2014].

Vita

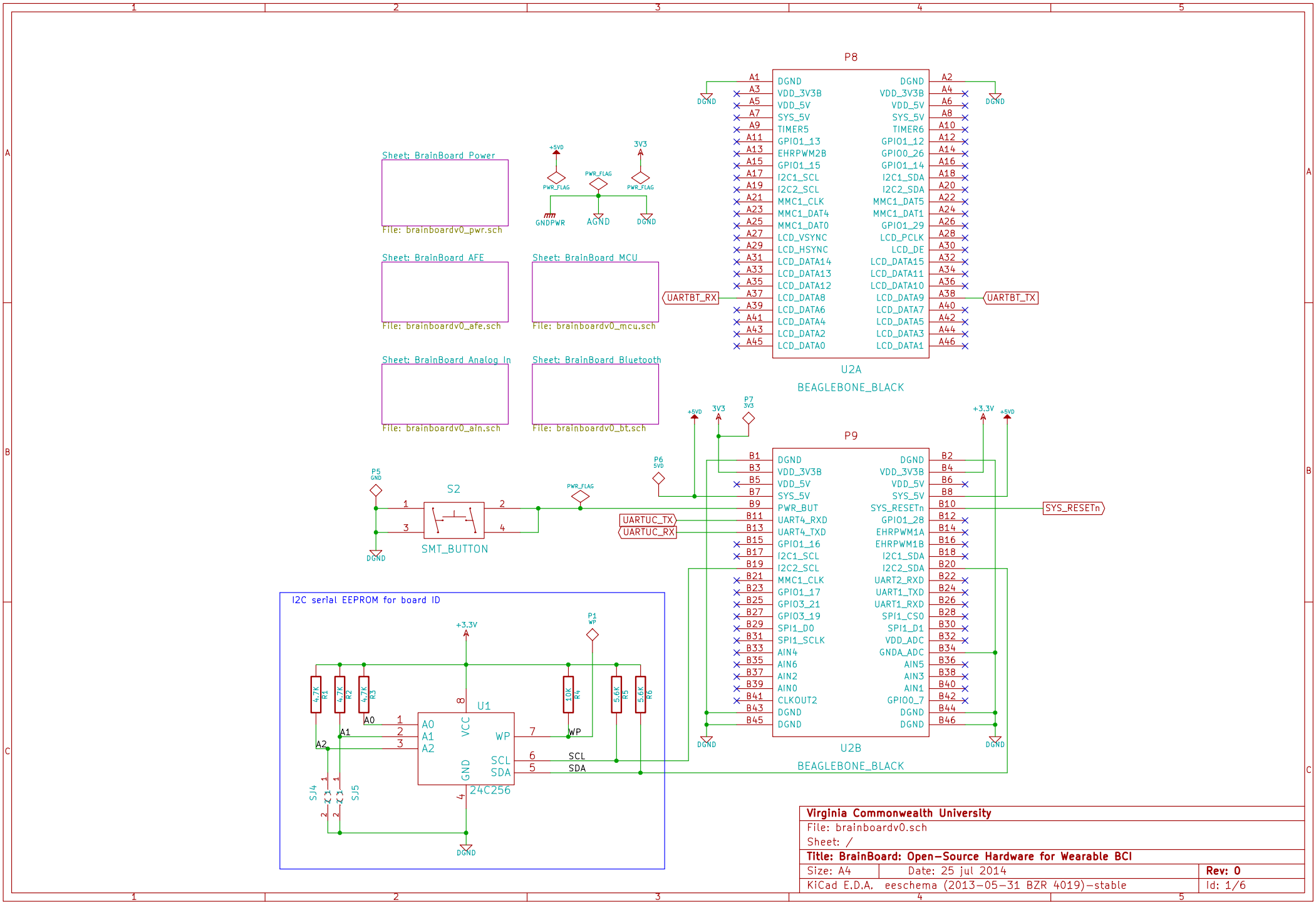
Graham Kelly was born on July 5, 1990 in Greensboro, NC, USA. He received his Bachelor of Science degree in Biomedical Engineering from Virginia Commonwealth University in May of 2011. Afterward, he began working as a lab technician in VCU's Artificial Heart Lab under Dr. Miller and Charles Taylor, where his work included electronic control systems prototyping, mechanical component design and fabrication, ECG signal processing, and image processing of fluid flow. It was in the Heart Lab that Graham was first exposed to Maker culture and the DIY ethos. He joined VCU's Master of Science program in Biomedical Engineering under Dr. Ou Bai to pursue his longtime interest in the brain and neural engineering. His research interests include brain-computer interfaces for advanced prosthetics, wearable electronics, and the application of nanosensors to real-time minimally invasive brain activity recording. He plans to enter the medical device industry before returning to school to get his Ph.D.

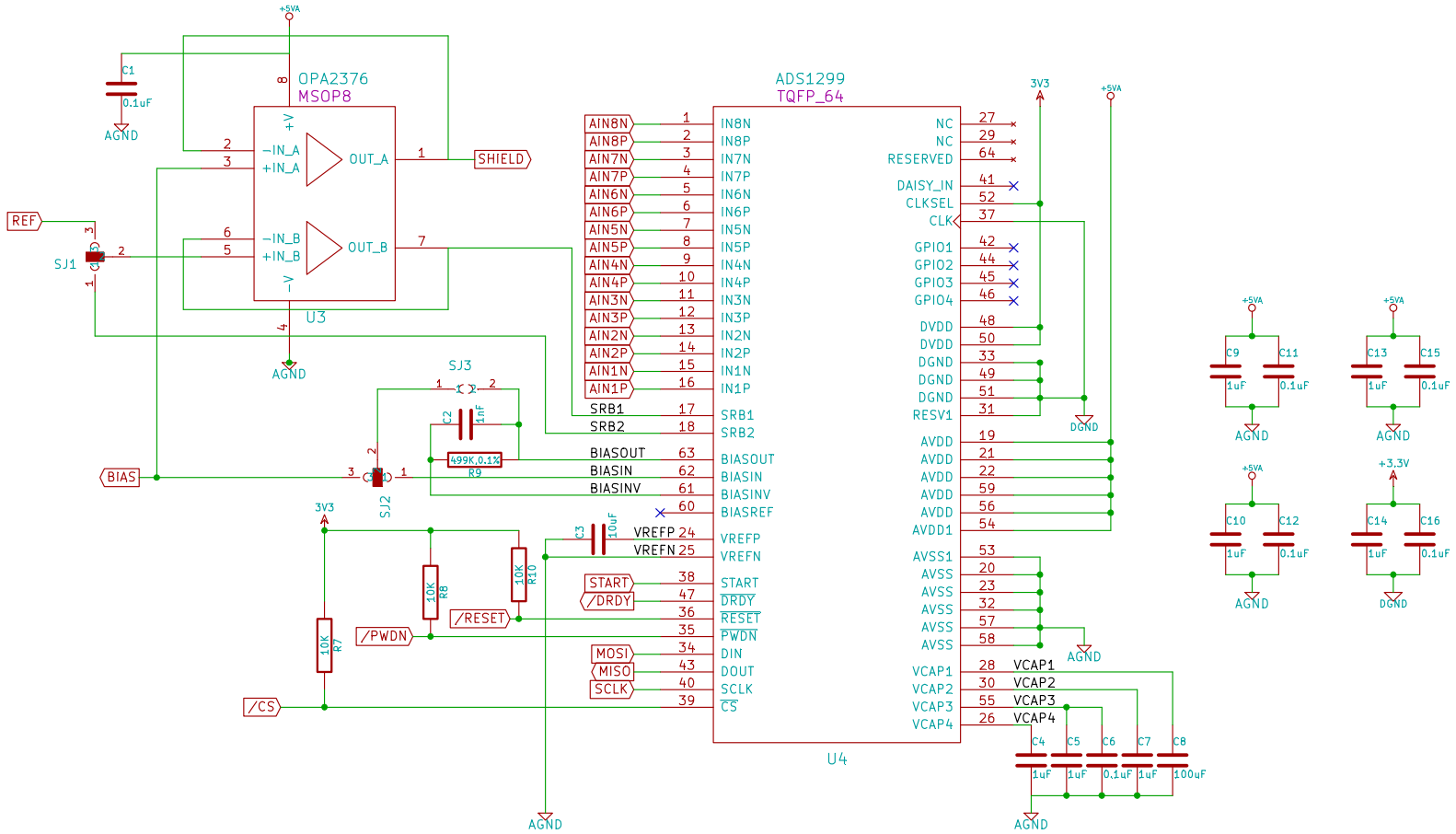
Appendices

Appendix A

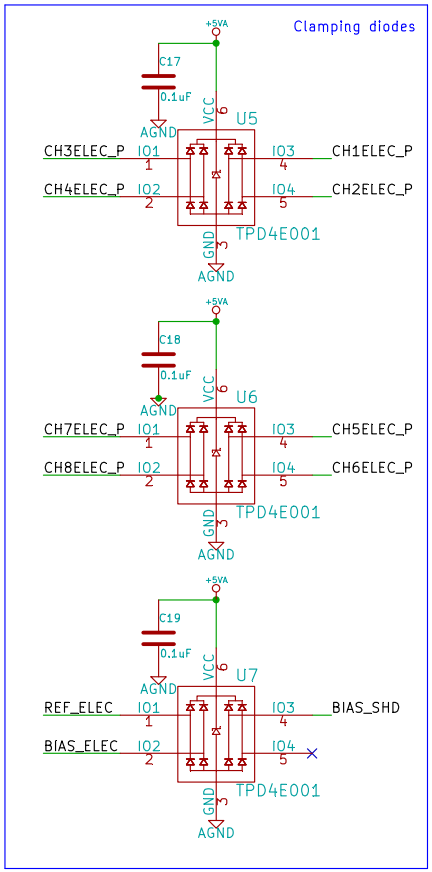
Circuit Schematics

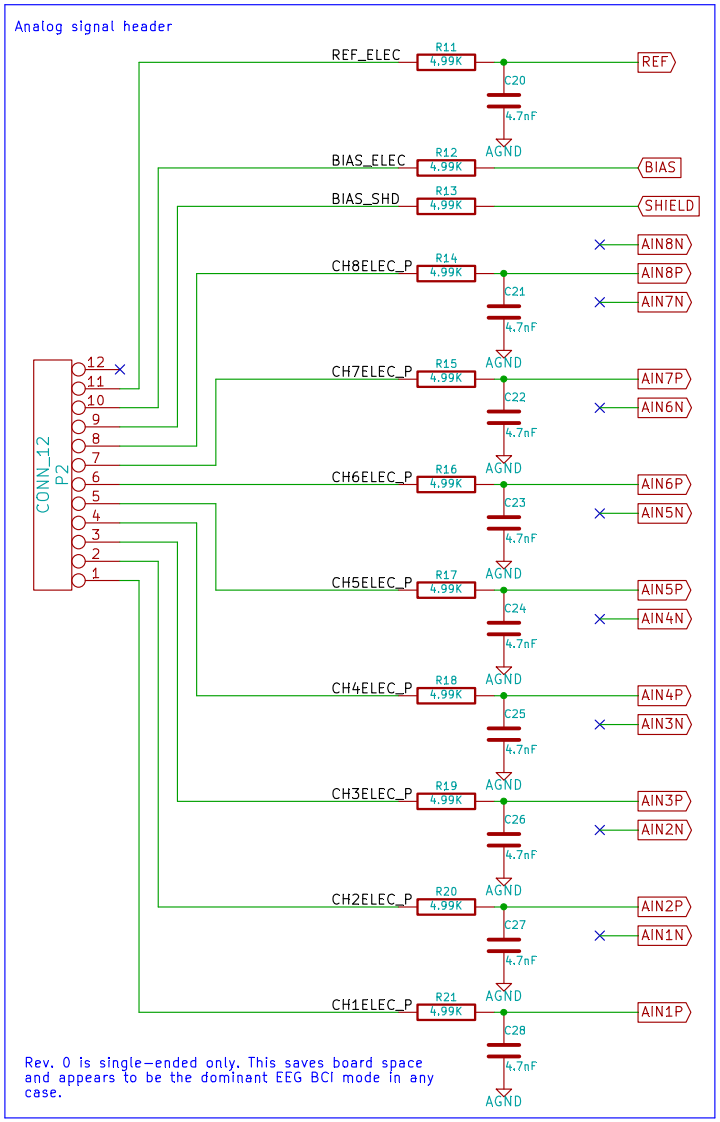
Brainboard R0


P8
U2A
BEAGLEBONE_BLACK
P9
U2B
BEAGLEBONE_BLACK
Sheet: BrainBoard Power
File: brainboardv0_pwr.sch
Sheet: BrainBoard AFE
File: brainboardv0_afe.sch
Sheet: BrainBoard MCU
File: brainboardv0_mcu.sch
Sheet: BrainBoard Analog In
File: brainboardv0_ain.sch
Sheet: BrainBoard Bluetooth
File: brainboardv0_bt.sch
UARTBT_RX
UARTBT_TX
UARTUC_TX
UARTUC_RX
SYS_RESETn
S2
SMT_BUTTON
I2C serial EEPROM for board ID
U1
24C256
SCL
SDA
WP
A0
A1
A2
Virginia Commonwealth University
File: brainboardv0.sch
Sheet: /
Title: BrainBoard: Open-Source Hardware for Wearable BCI
Size: A4
Date: 25 jul 2014
Rev: 0
KiCad E.D.A. eeschema (2013-05-31 BZR 4019)-stable
Id: 1/6

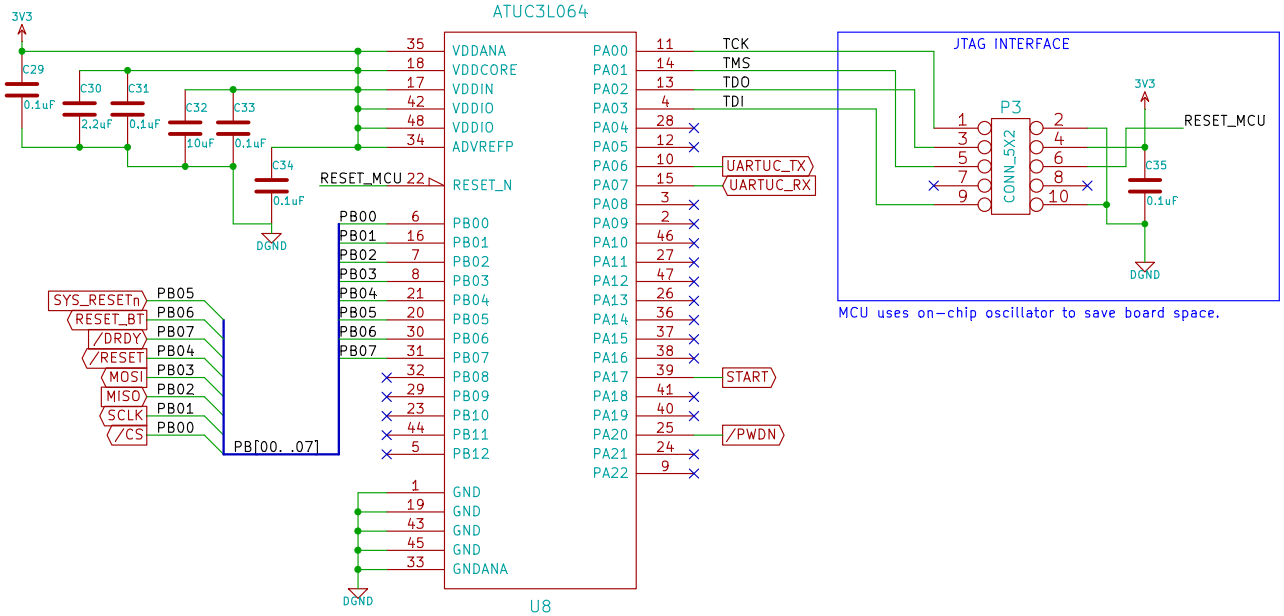


Virginia Commonwealth University		
File: brainboardv0_afe.sch		
Sheet: /BrainBoard AFE/		
Title: BrainBoard: Open-Source Hardware for Wearable BCI		
Size: A4	Date: 25 jul 2014	**Rev: 0**
KiCad E.D.A. eeschema (2013-05-31 BZR 4019)-stable		Id: 2/6

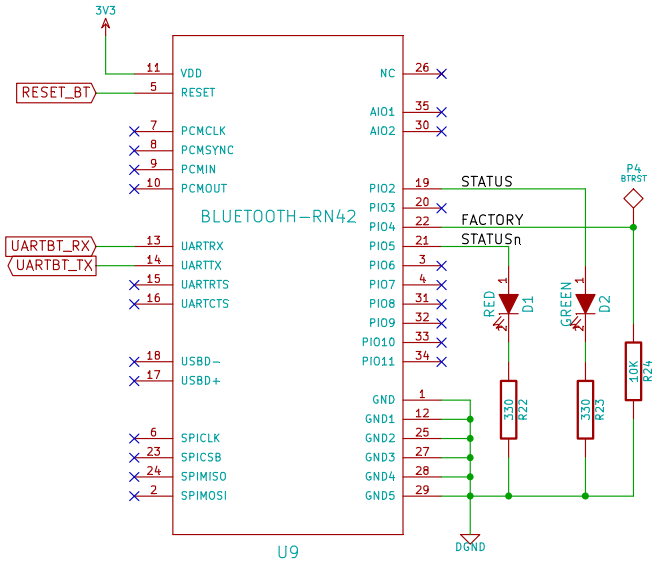




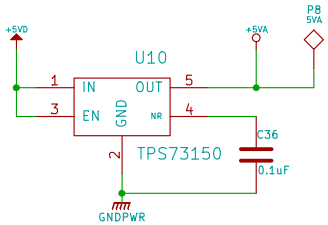
Virginia Commonwealth University		
File: brainboardv0_ain.sch		
Sheet: /BrainBoard Analog In/		
Title: BrainBoard: Open-Source Hardware for Wearable BCI		
Size: A4	Date: 25 jul 2014	**Rev: 0**
KiCad E.D.A. eeschema (2013-05-31 BZR 4019)-stable		Id: 3/6



Virginia Commonwealth University		
File: brainboardv0_mcu.sch		
Sheet: /BrainBoard MCU/		
Title: BrainBoard: Open-Source Hardware for Wearable BCI		
Size: A4	Date: 25 jul 2014	**Rev: 0**
KiCad E.D.A. eeschema (2013-05-31 BZR 4019)-stable		Id: 4/6



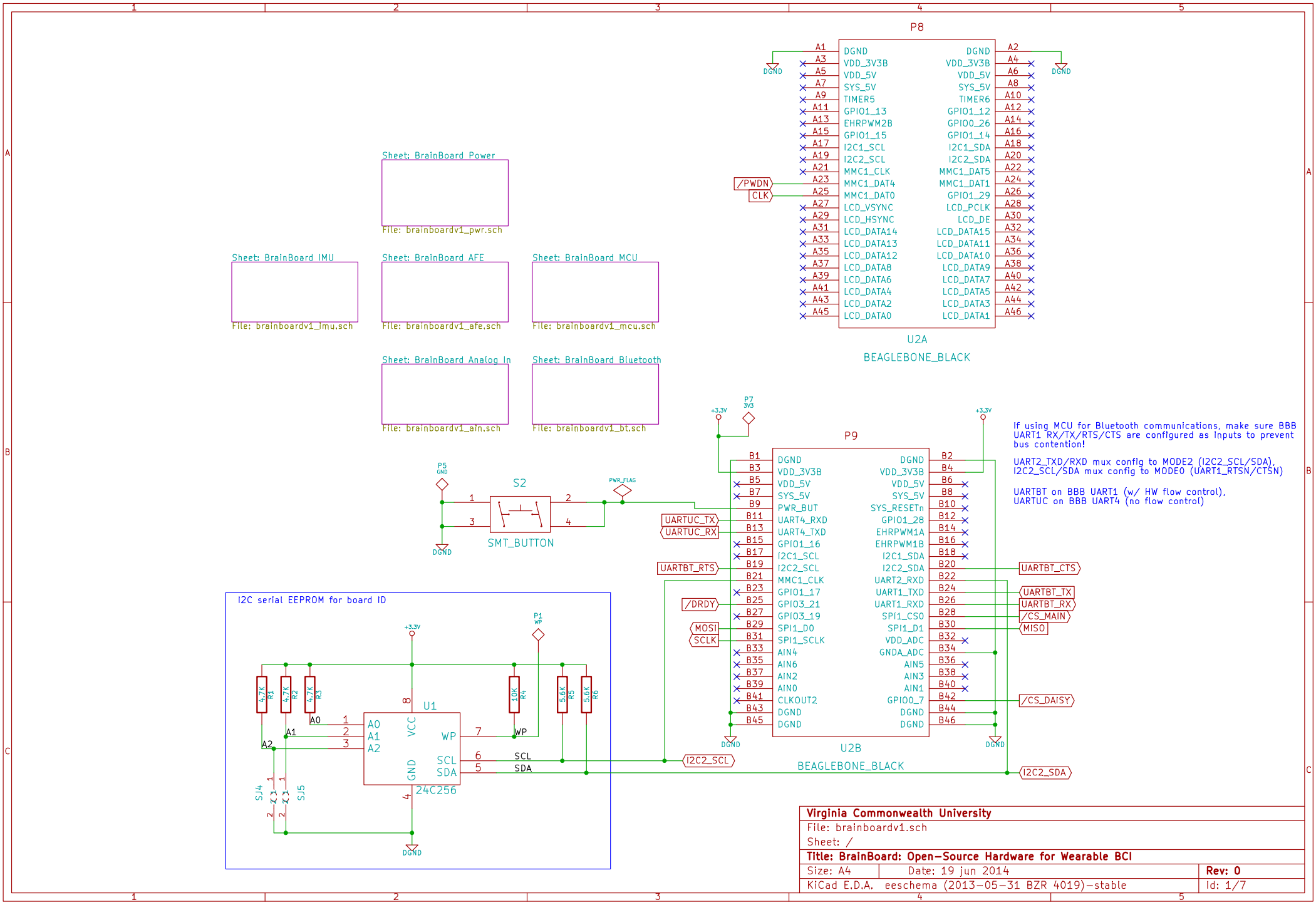
Virginia Commonwealth University		
File: brainboardv0_bt.sch		
Sheet: /BrainBoard Bluetooth/		
Title: BrainBoard: Open-Source Hardware for Wearable BCI		
Size: A4	Date: 25 jul 2014	**Rev: 0**
KiCad E.D.A. eeschema (2013-05-31 BZR 4019)-stable		Id: 5/6

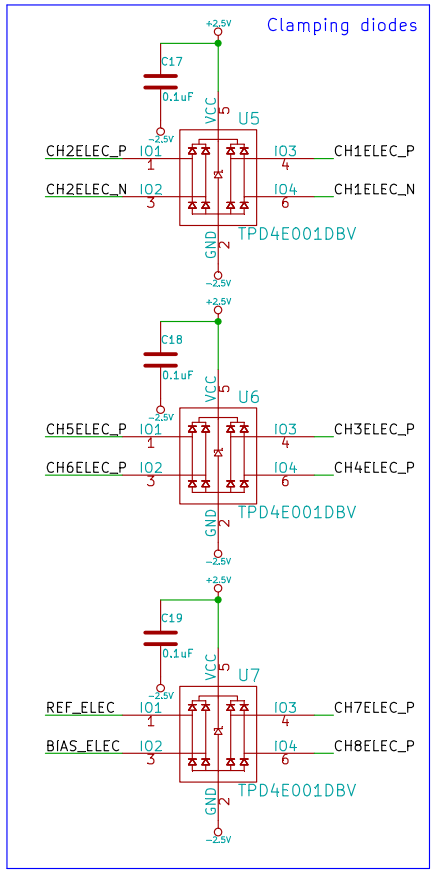


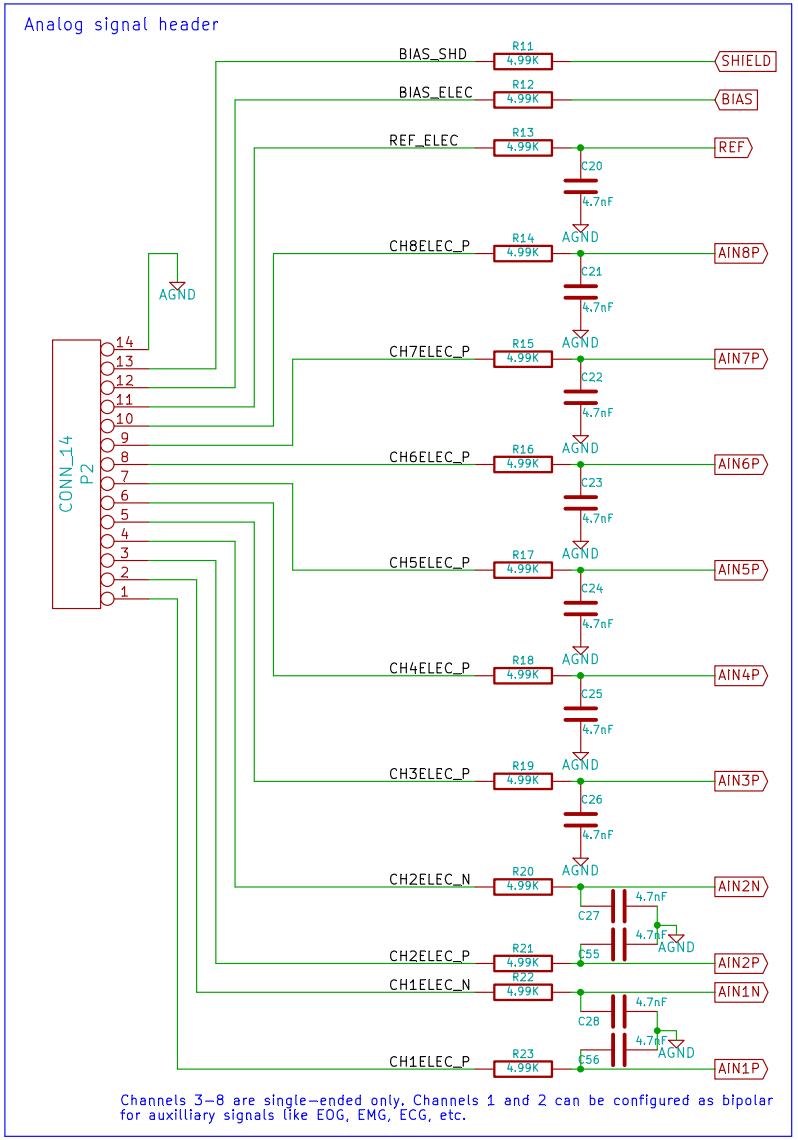
Rev. 0 is meant for demonstration of basic signal acquisition capabilities and does not use battery power. +5VD is the VCC_5V supply from the BeagleBone. The TPS73150 is used for post-regulation.

Virginia Commonwealth University		
File: brainboardv0_pwr.sch		
Sheet: /BrainBoard Power/		
Title: BrainBoard: Open-Source Hardware for Wearable BCI		
Size: A4	Date: 25 jul 2014	**Rev: 0**
KiCad E.D.A. eeschema (2013-05-31 BZR 4019)-stable		Id: 6/6

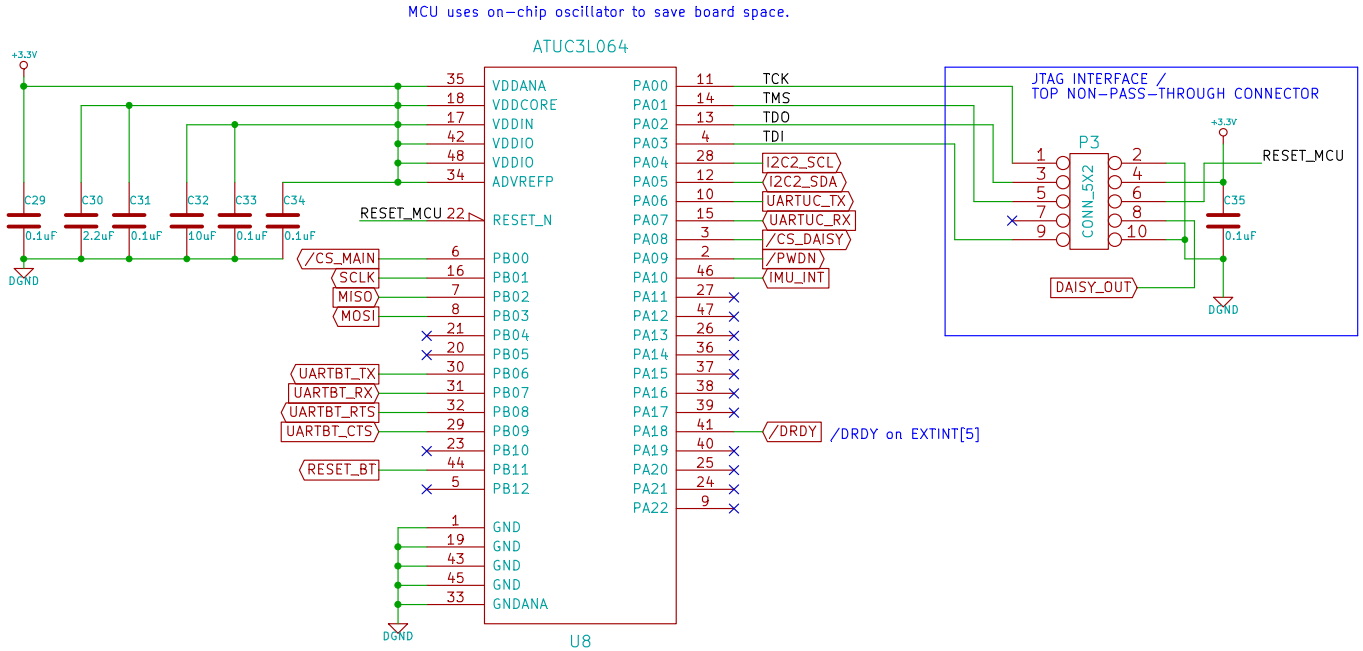
Brainboard R1


P8
U2A
BEAGLEBONE_BLACK
Sheet: BrainBoard Power
File: brainboardv1_pwr.sch
Sheet: BrainBoard IMU
File: brainboardv1_imu.sch
Sheet: BrainBoard AFE
File: brainboardv1_afe.sch
Sheet: BrainBoard MCU
File: brainboardv1_mcu.sch
Sheet: BrainBoard Analog In
File: brainboardv1_ain.sch
Sheet: BrainBoard Bluetooth
File: brainboardv1_bt.sch
P9
U2B
BEAGLEBONE_BLACK
S2
SMT_BUTTON
I2C serial EEPROM for board ID
U1
24C256
If using MCU for Bluetooth communications, make sure BBB UART1 RX/TX/RTS/CTS are configured as inputs to prevent bus contention!
UART2_TXD/RXD mux config to MODE2 (I2C2_SCL/SDA), I2C2_SCL/SDA mux config to MODE0 (UART1_RTSN/CTSN)
UARTBT on BBB UART1 (w/ HW flow control), UARTUC on BBB UART4 (no flow control)
Virginia Commonwealth University
File: brainboardv1.sch
Sheet: /
Title: BrainBoard: Open-Source Hardware for Wearable BCI
Size: A4
Date: 19 jun 2014
Rev: 0
KiCad E.D.A. eeschema (2013-05-31 BZR 4019)-stable
Id: 1/7

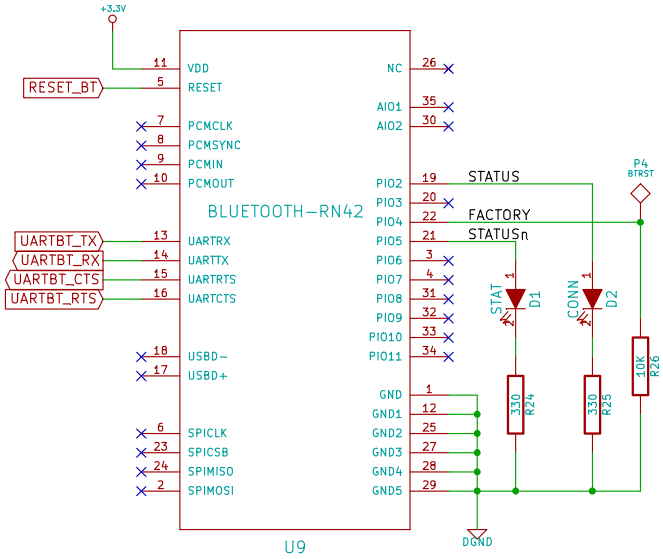




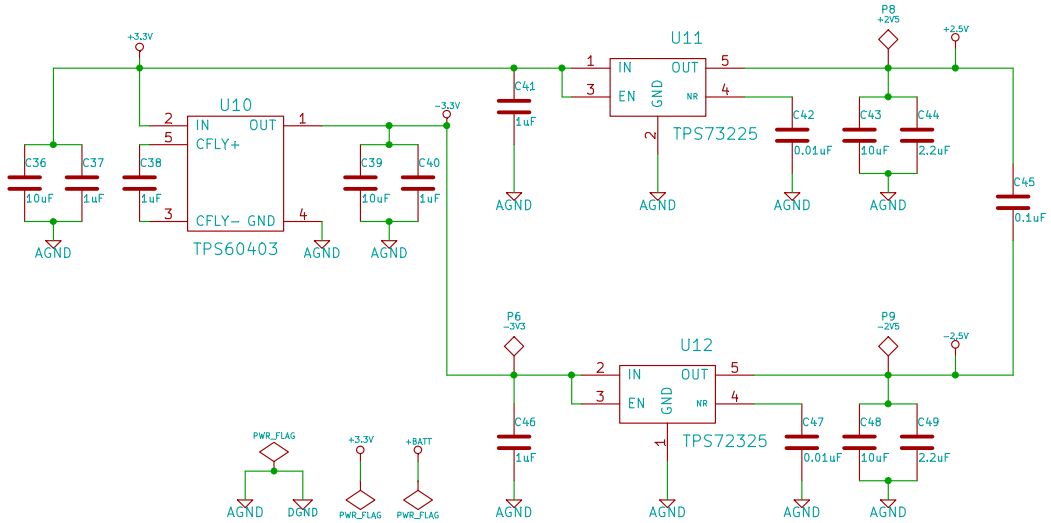
Virginia Commonwealth University		
File: brainboardv1_ain.sch		
Sheet: /BrainBoard Analog In/		
Title: BrainBoard: Open-Source Hardware for Wearable BCI		
Size: A4	Date: 19 jun 2014	**Rev: 0**
KiCad E.D.A. eeschema (2013-05-31 BZR 4019)-stable		Id: 2/7



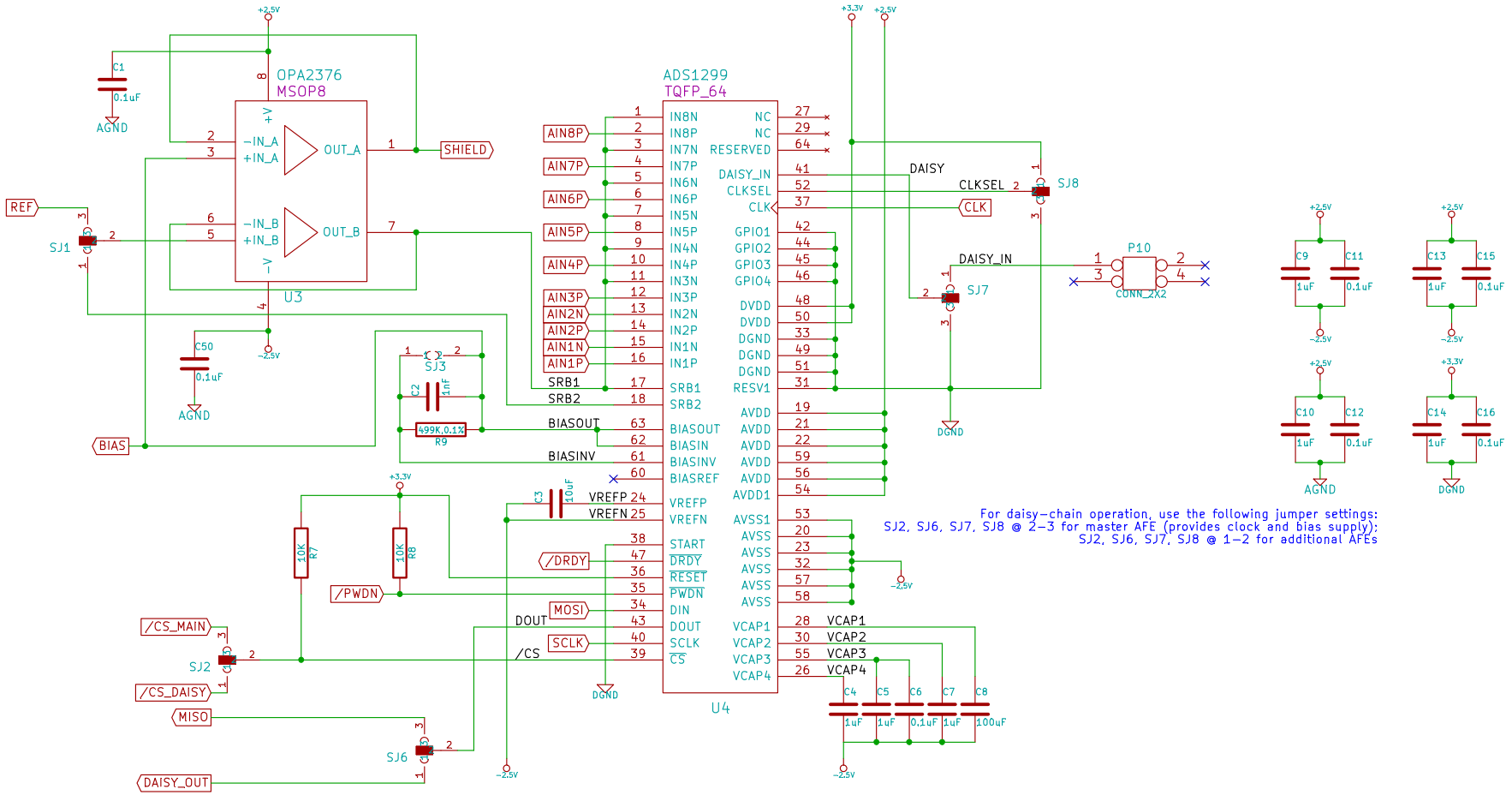
Virginia Commonwealth University		
File: brainboardv1_mcu.sch		
Sheet: /BrainBoard MCU/		
Title: BrainBoard: Open-Source Hardware for Wearable BCI		
Size: A4	Date: 19 jun 2014	**Rev: 0**
KiCad E.D.A. eeschema (2013-05-31 BZR 4019)-stable		Id: 3/7



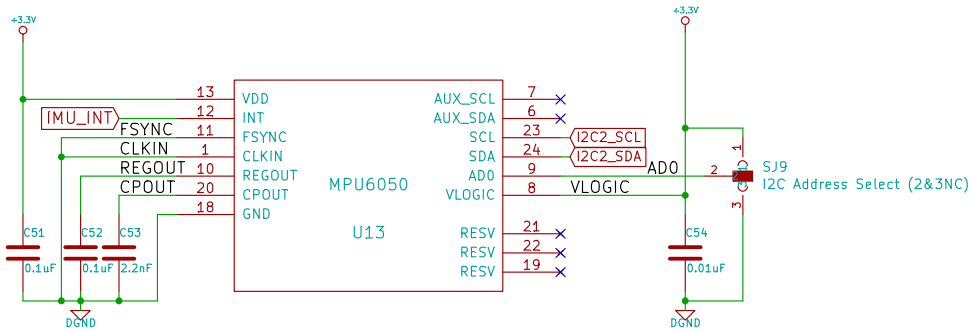
Virginia Commonwealth University		
File: brainboardv1_bt.sch		
Sheet: /BrainBoard Bluetooth/		
Title: BrainBoard: Open-Source Hardware for Wearable BCI		
Size: A4	Date: 19 jun 2014	**Rev: 0**
KiCad E.D.A. eeschema (2013-05-31 BZR 4019)-stable		Id: 4/7



Virginia Commonwealth University		
File: brainboardv1_pwr.sch		
Sheet: /BrainBoard Power/		
Title: BrainBoard: Open-Source Hardware for Wearable BCI		
Size: A4	Date: 19 jun 2014	**Rev: 0**
KiCad E.D.A. eeschema (2013-05-31 BZR 4019)-stable		Id: 5/7

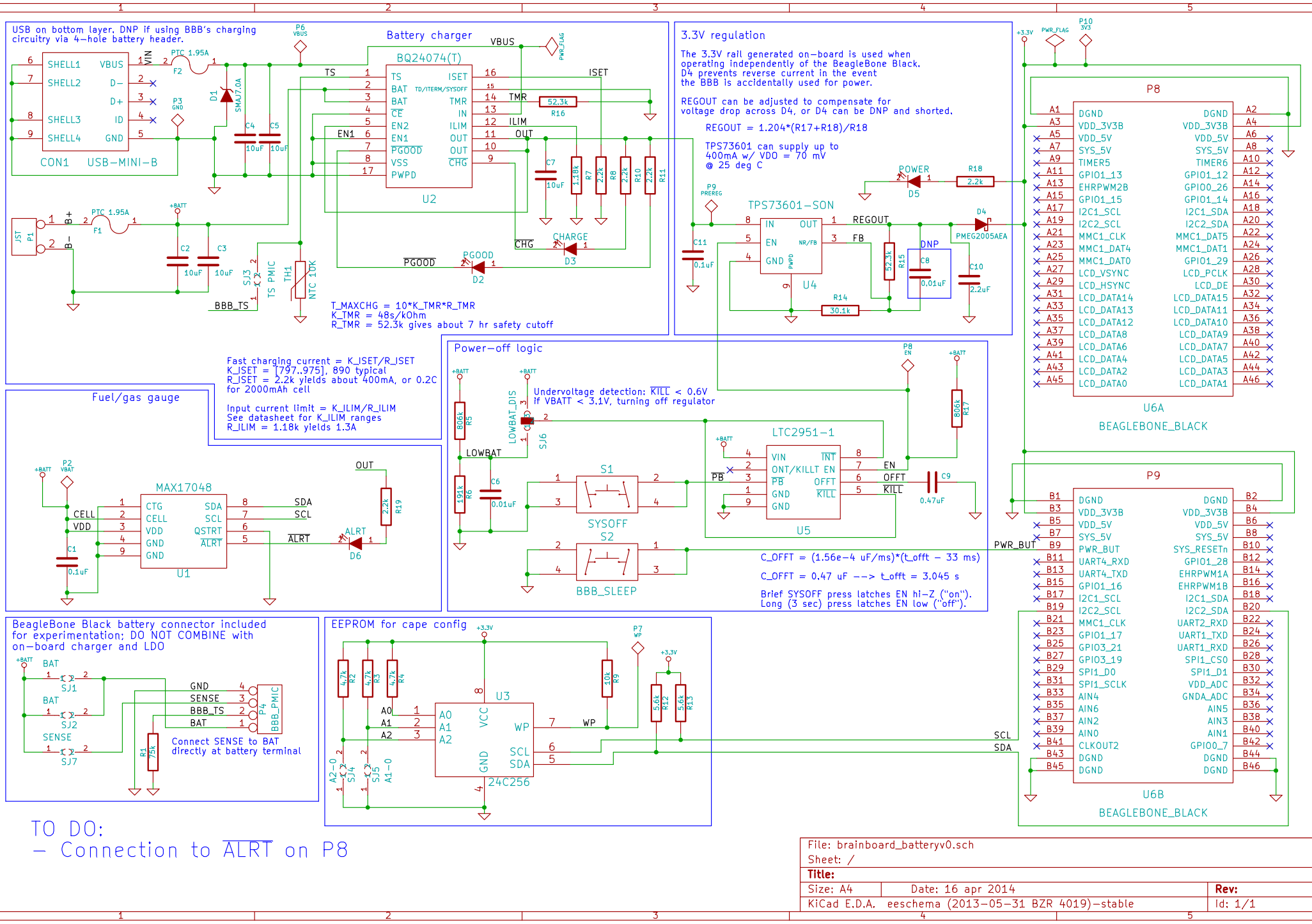


Virginia Commonwealth University		
File: brainboardv1_afe.sch		
Sheet: /BrainBoard AFE/		
Title: BrainBoard: Open-Source Hardware for Wearable BCI		
Size: A4	Date: 19 jun 2014	**Rev: 0**
KiCad E.D.A. eeschema (2013-05-31 BZR 4019)-stable		Id: 6/7



File: brainboardv1_imu.sch		
Sheet: /BrainBoard IMU/		
Title:		
Size: A4	Date: 19 jun 2014	**Rev:**
KiCad E.D.A. eeschema (2013-05-31 BZR 4019)-stable		Id: 7/7

Battery Board


USB on bottom layer. DNP if using BBB's charging circuitry via 4-hole battery header.
CON1 USB-MINI-B
Battery charger
BQ24074(T)
U2
T_MAXCHG = 10*K_TMR*R_TMR
K_TMR = 48s/kOhm
R_TMR = 52.3k gives about 7 hr safety cutoff
3.3V regulation
The 3.3V rail generated on-board is used when operating independently of the BeagleBone Black. D4 prevents reverse current in the event the BBB is accidentally used for power.
REGOUT can be adjusted to compensate for voltage drop across D4, or D4 can be DNP and shorted.
REGOUT = 1.204*(R17+R18)/R18
TPS73601 can supply up to 400mA w/ VDO = 70 mV @ 25 deg C
TPS73601-SON
U4
Fast charging current = K_ISET/R_ISET
K_ISET = [797..975], 890 typical
R_ISET = 2.2k yields about 400mA, or 0.2C for 2000mAh cell
Input current limit = K_ILIM/R_ILIM
See datasheet for K_ILIM ranges
R_ILIM = 1.18k yields 1.3A
Fuel/gas gauge
MAX17048
U1
Power-off logic
Undervoltage detection: KILL < 0.6V if VBATT < 3.1V, turning off regulator
LTC2951-1
U5
SYSOFF
BBB_SLEEP
C_OFFT = (1.56e-4 uF/ms)*(t_offt - 33 ms)
C_OFFT = 0.47 uF --> t_offt = 3.045 s
Brief SYSOFF press latches EN hi-Z ("on"). Long (3 sec) press latches EN low ("off").
BeagleBone Black battery connector included for experimentation; DO NOT COMBINE with on-board charger and LDO
Connect SENSE to BAT directly at battery terminal
EEPROM for cape config
24C256
U3
P8
U6A
BEAGLEBONE_BLACK
P9
U6B
BEAGLEBONE_BLACK
TO DO:
- Connection to ALRT on P8
File: brainboard_batteryv0.sch
Sheet: /
Title:
Size: A4
Date: 16 apr 2014
Rev:
KiCad E.D.A. eeschema (2013-05-31 BZR 4019)-stable
Id: 1/1

Brainboard LW

Sheet: Brainboard Bluetooth

File: brainboard_wearable_v2_bt.sch

Sheet: Brainboard MCU

File: brainboard_wearable_v2_mcu.sch

Sheet: Brainboard Power

File: brainboard_wearable_v2_pwr.sch

Sheet: Brainboard AFE

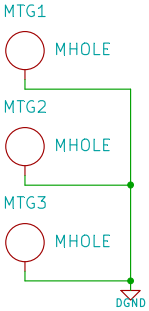
File: brainboard_wearable_v2_afe.sch

Sheet: Brainboard IMU

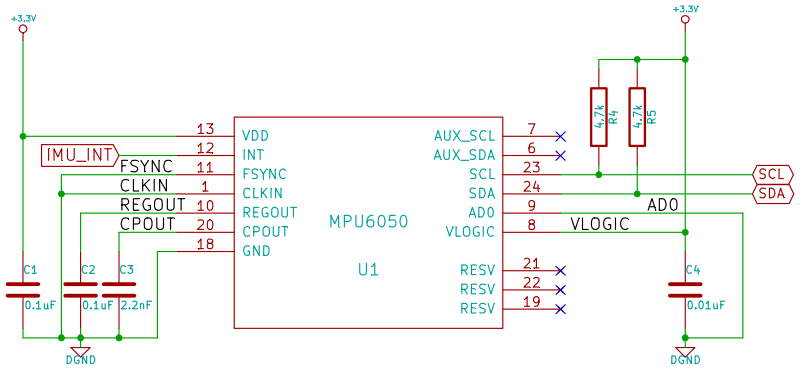
File: brainboard_wearable_v2_imu.sch

Sheet: Brainboard Analog In

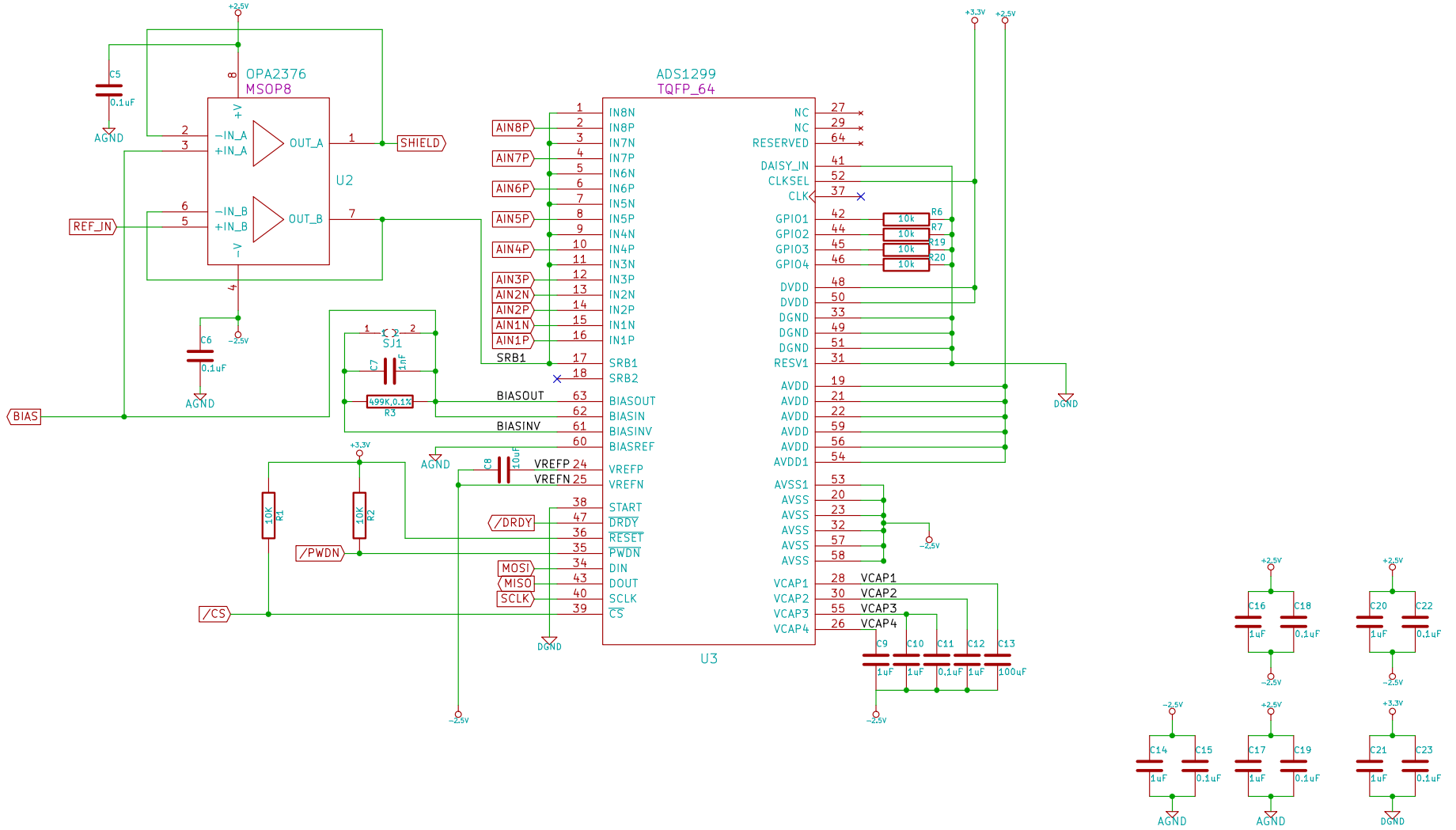
File: brainboard_wearable_v2_ain.sch



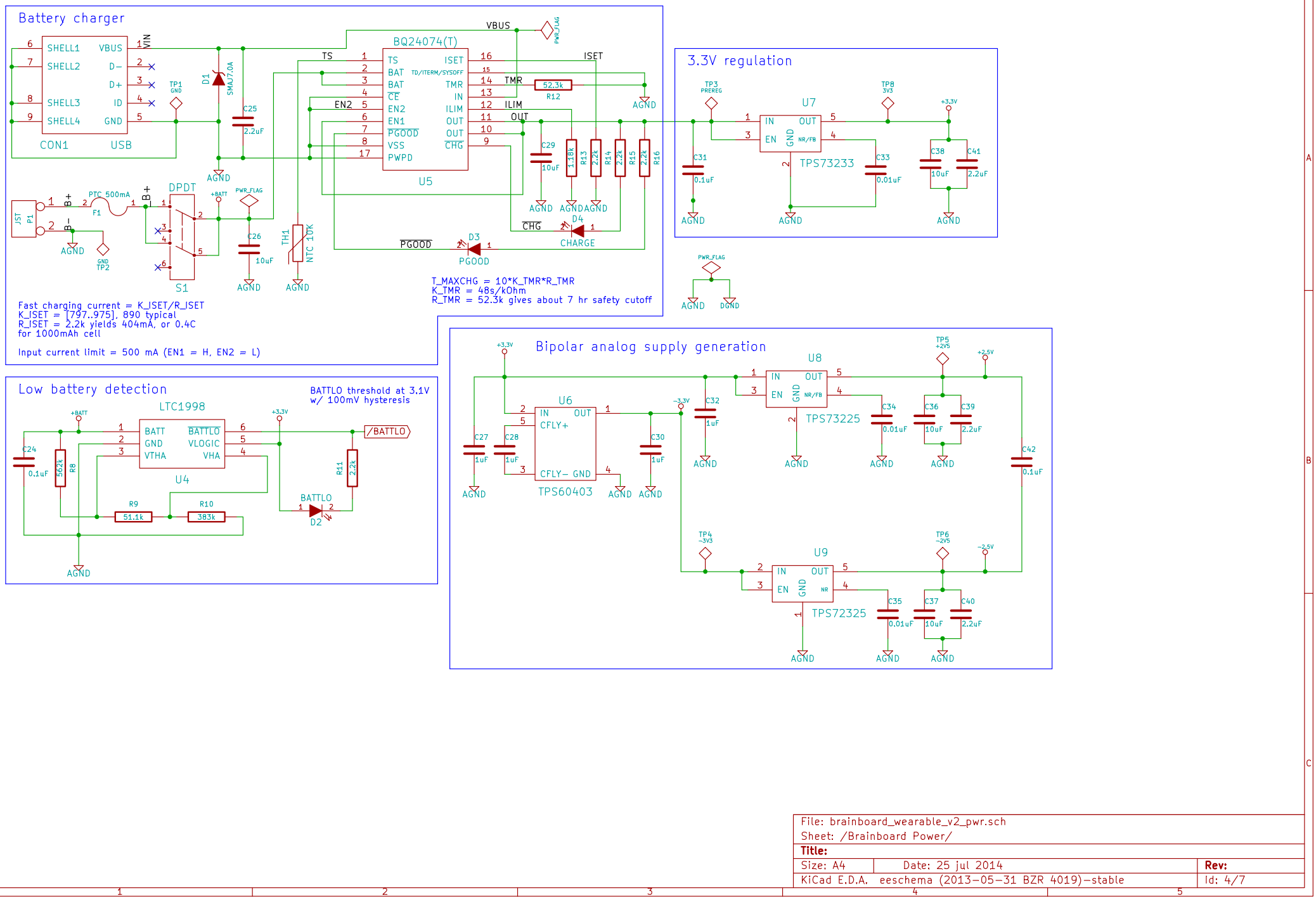
Virginia Commonwealth University		
File: brainboard_wearable_v2.sch		
Sheet: /		
Title: BrainBoard: Open-Source Hardware for Wearable BCI		
Size: A4	Date: 25 jul 2014	**Rev: 0**
KiCad E.D.A. eeschema (2013-05-31 BZR 4019)-stable		Id: 1/7



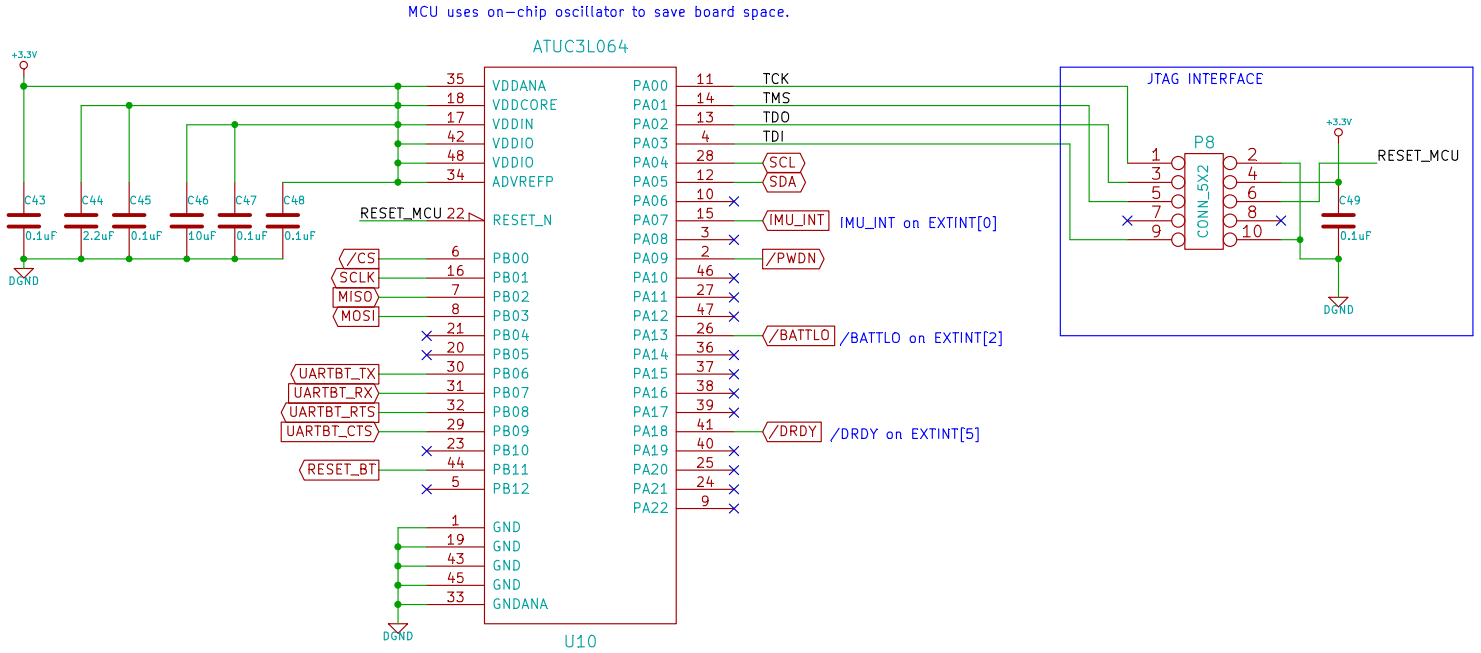
File: brainboard_wearable_v2_imu.sch		
Sheet: /Brainboard IMU/		
Title:		
Size: A4	Date: 25 jul 2014	**Rev:**
KiCad E.D.A. eeschema (2013-05-31 BZR 4019)-stable		Id: 2/7



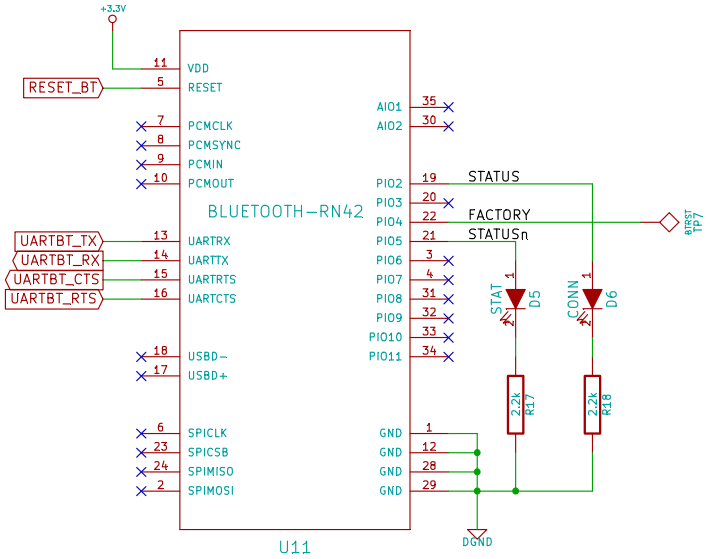
Virginia Commonwealth University		
File: brainboard_wearable_v2_afe.sch		
Sheet: /Brainboard AFE/		
Title: BrainBoard: Open-Source Hardware for Wearable BCI		
Size: A4	Date: 25 jul 2014	**Rev: 0**
KiCad E.D.A. eeschema (2013-05-31 BZR 4019)-stable		Id: 3/7



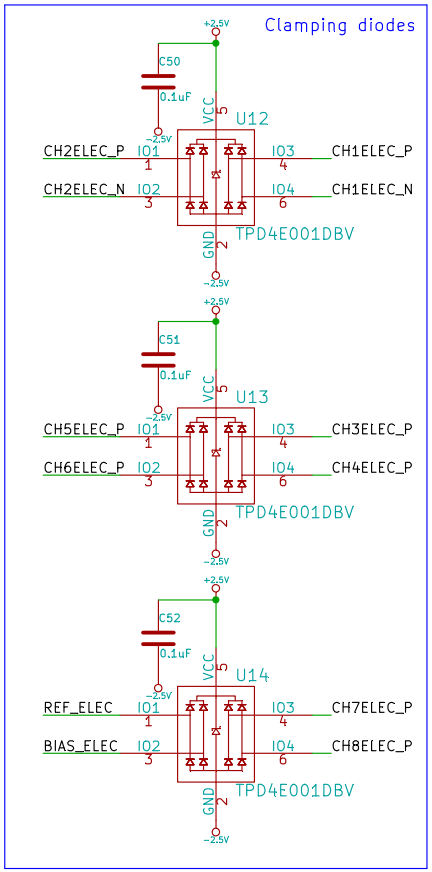
File: brainboard_wearable_v2_pwr.sch		
Sheet: /Brainboard Power/		
Title:		
Size: A4	Date: 25 jul 2014	**Rev:**
KiCad E.D.A.	eeschema (2013−05−31 BZR 4019)−stable	Id: 4/7

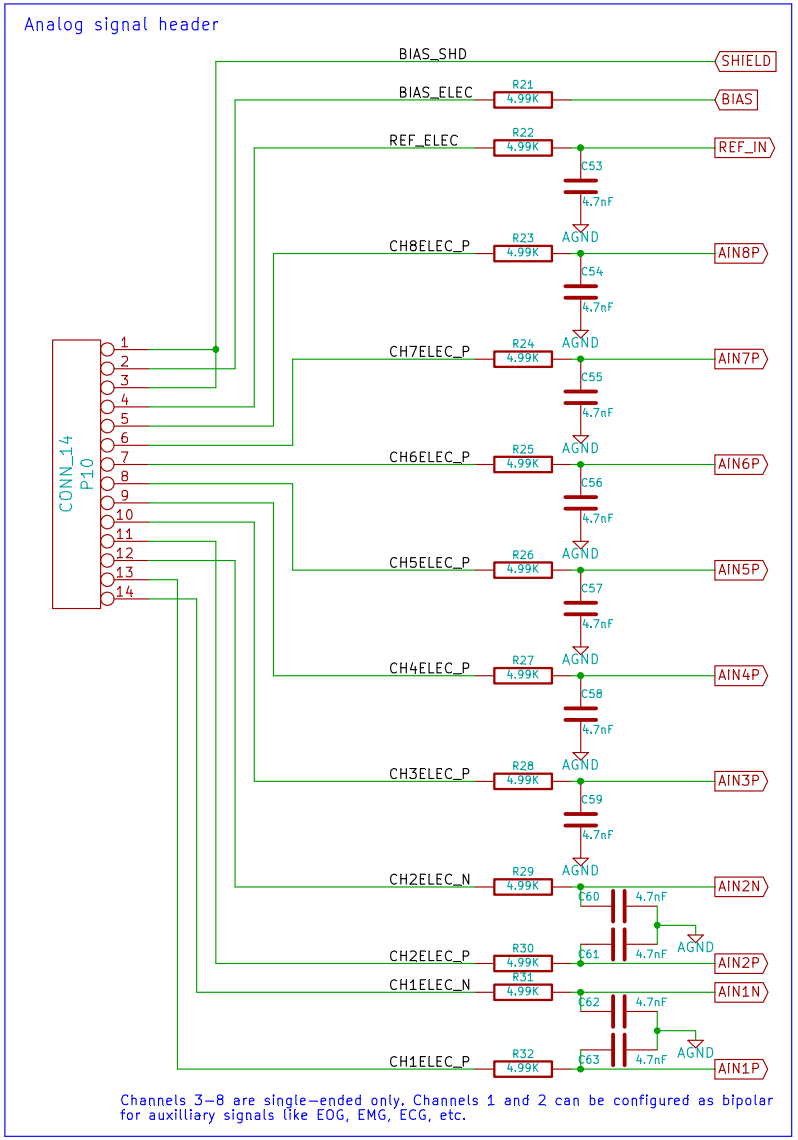


Virginia Commonwealth University		
File: brainboard_wearable_v2_mcu.sch		
Sheet: /Brainboard MCU/		
Title: BrainBoard: Open-Source Hardware for Wearable BCI		
Size: A4	Date: 25 jul 2014	**Rev: 0**
KiCad E.D.A. eeschema (2013-05-31 BZR 4019)-stable		Id: 5/7



Virginia Commonwealth University		
File: brainboard_wearable_v2_bt.sch		
Sheet: /Brainboard Bluetooth/		
Title: BrainBoard: Open-Source Hardware for Wearable BCI		
Size: A4	Date: 25 jul 2014	**Rev: 0**
KiCad E.D.A. eeschema (2013-05-31 BZR 4019)-stable		Id: 6/7



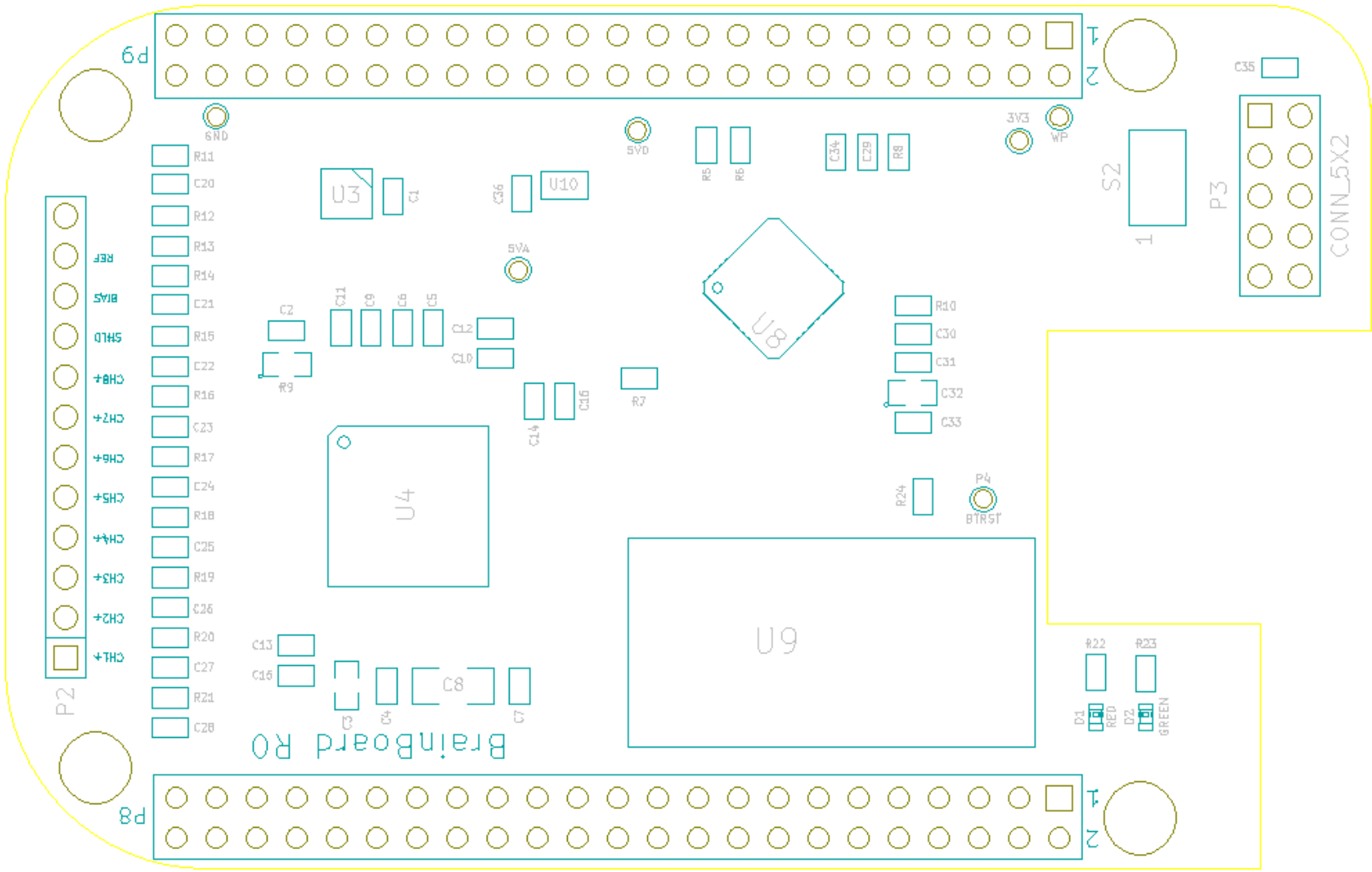


Virginia Commonwealth University		
File: brainboard_wearable_v2_ain.sch		
Sheet: /Brainboard Analog In/		
Title: BrainBoard: Open-Source Hardware for Wearable BCI		
Size: A4	Date: 25 jul 2014	**Rev: 0**
KiCad E.D.A. eeschema (2013-05-31 BZR 4019)-stable		Id: 7/7

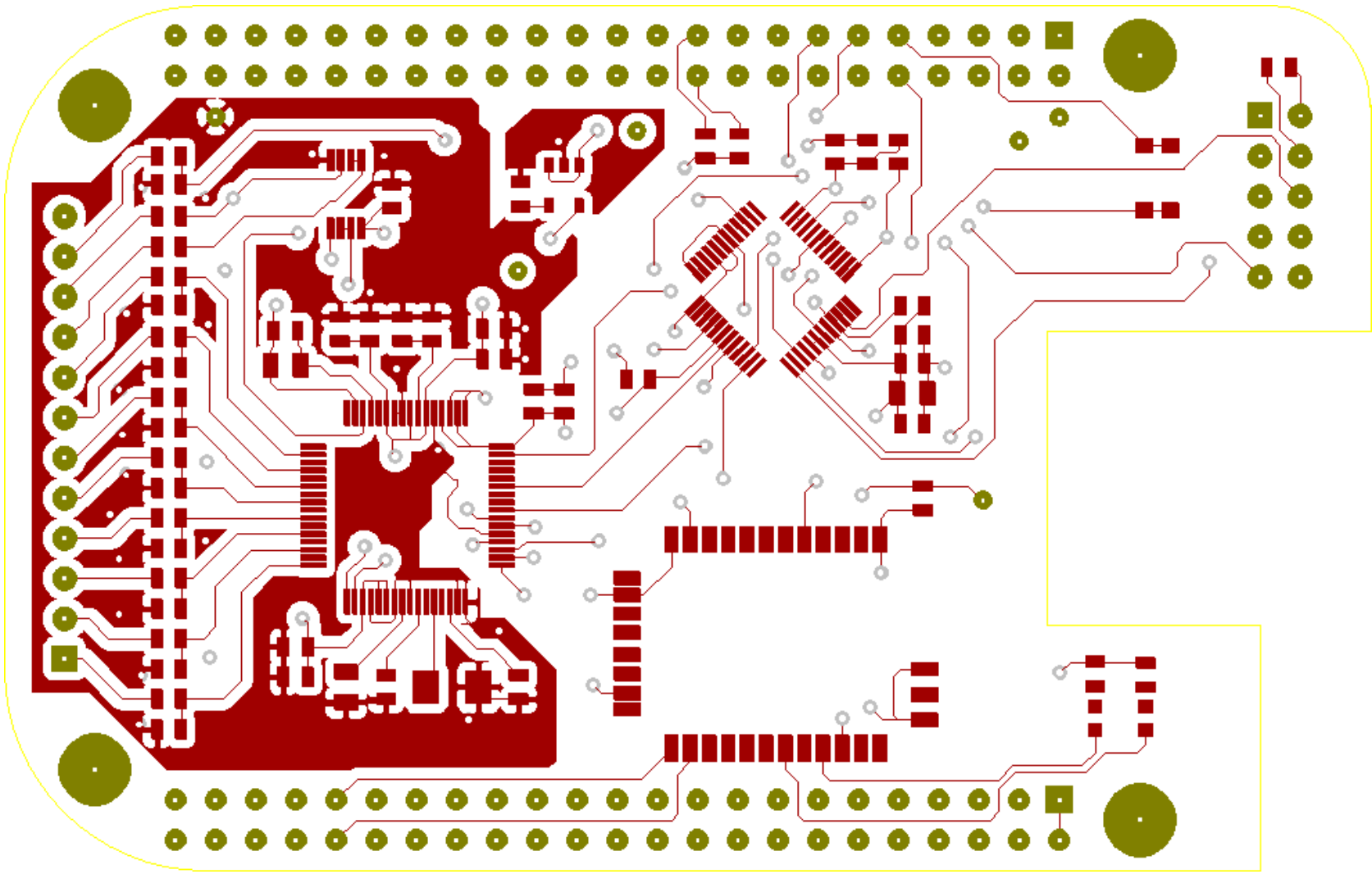
Appendix B

Board Layouts

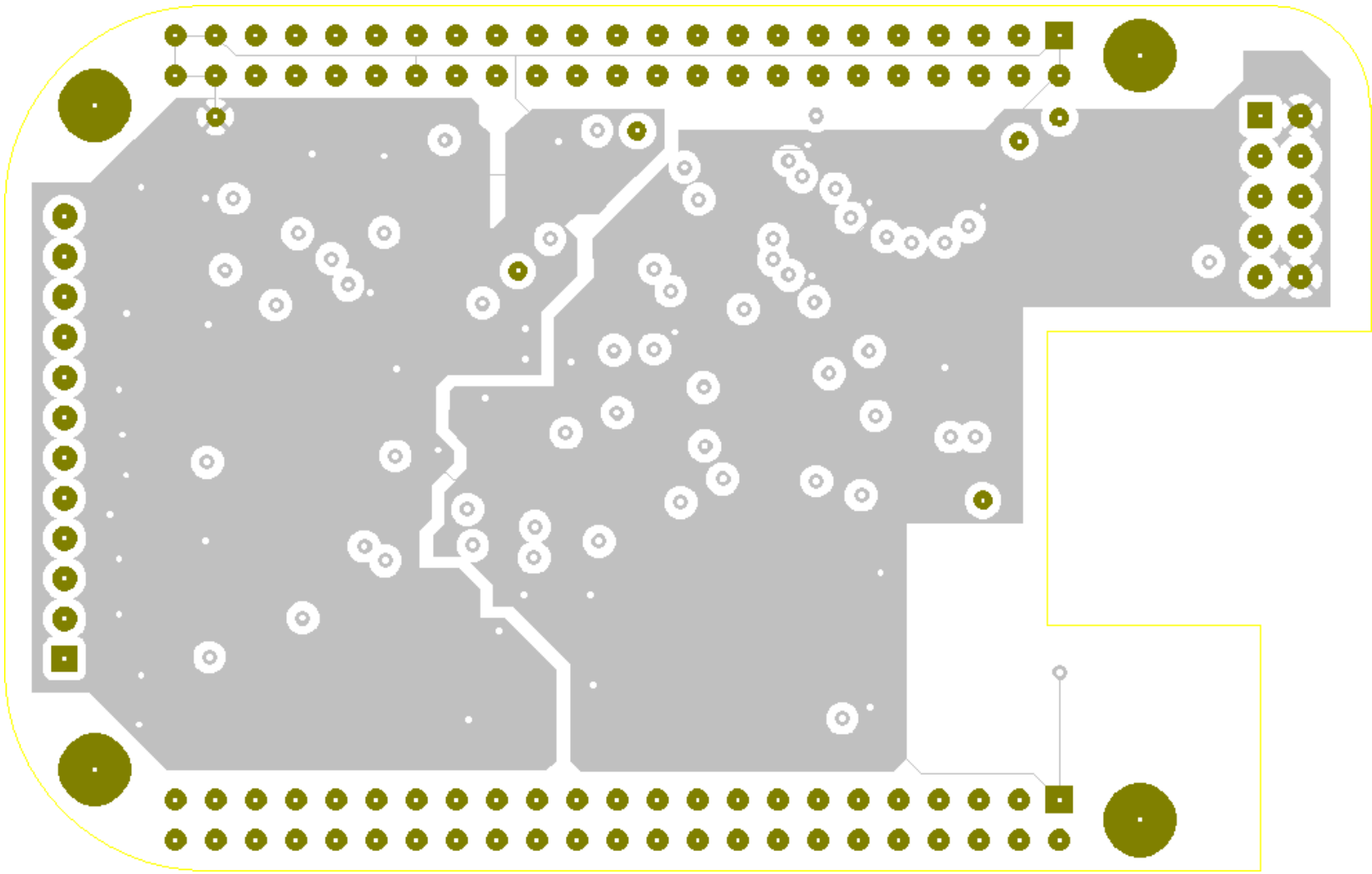
Brainboard R0



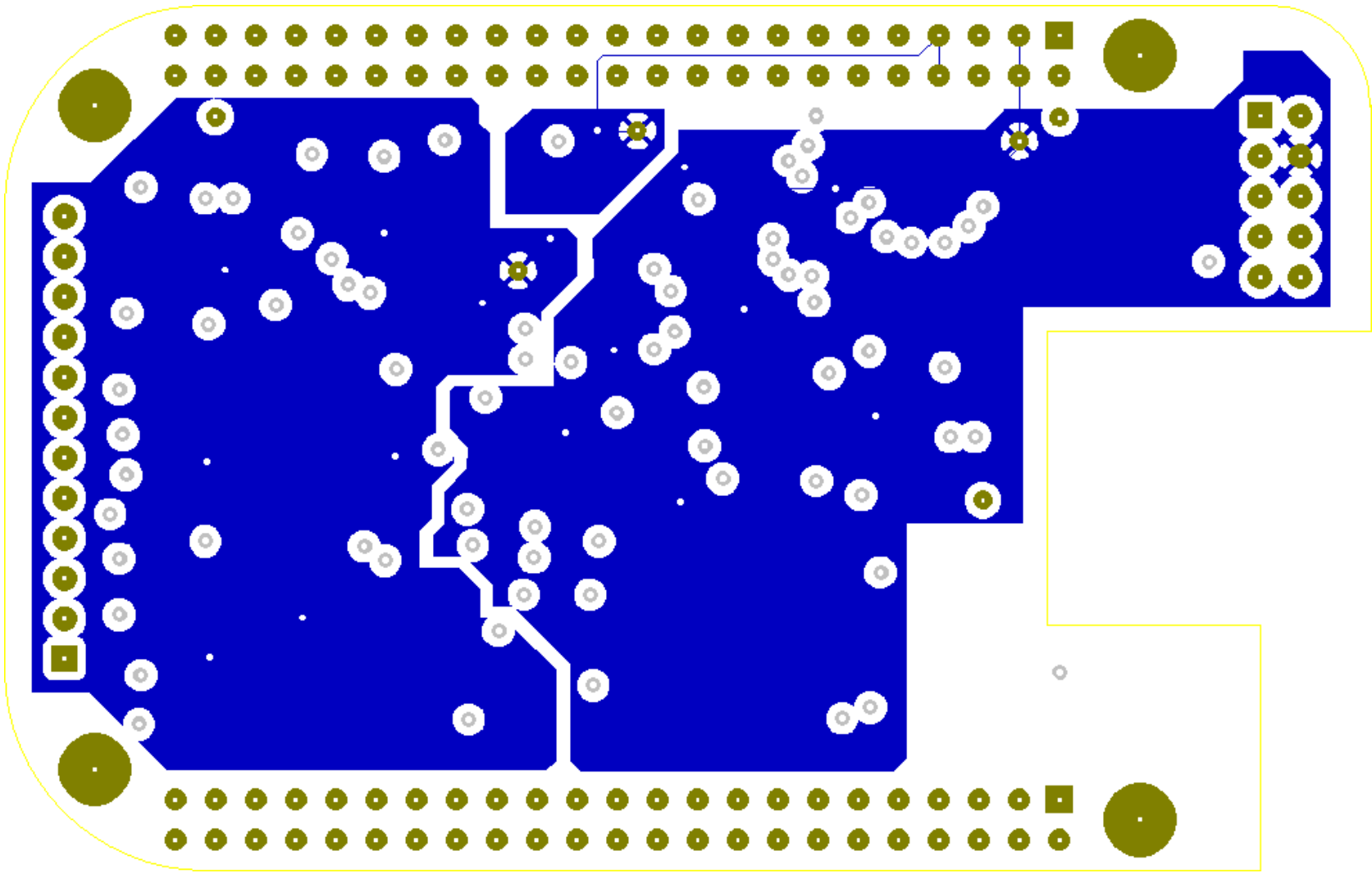
Top Silkscreen



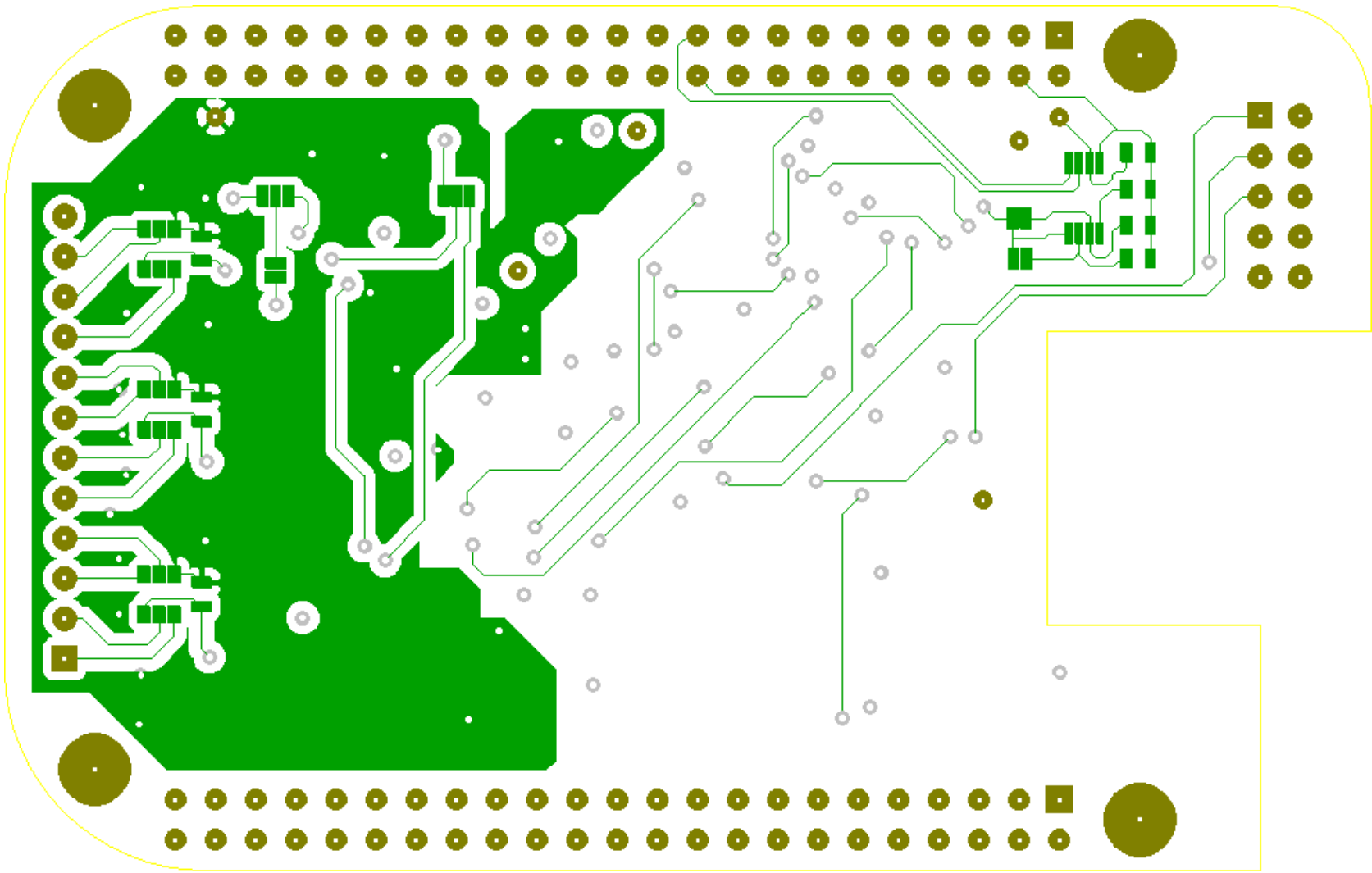
Top Copper



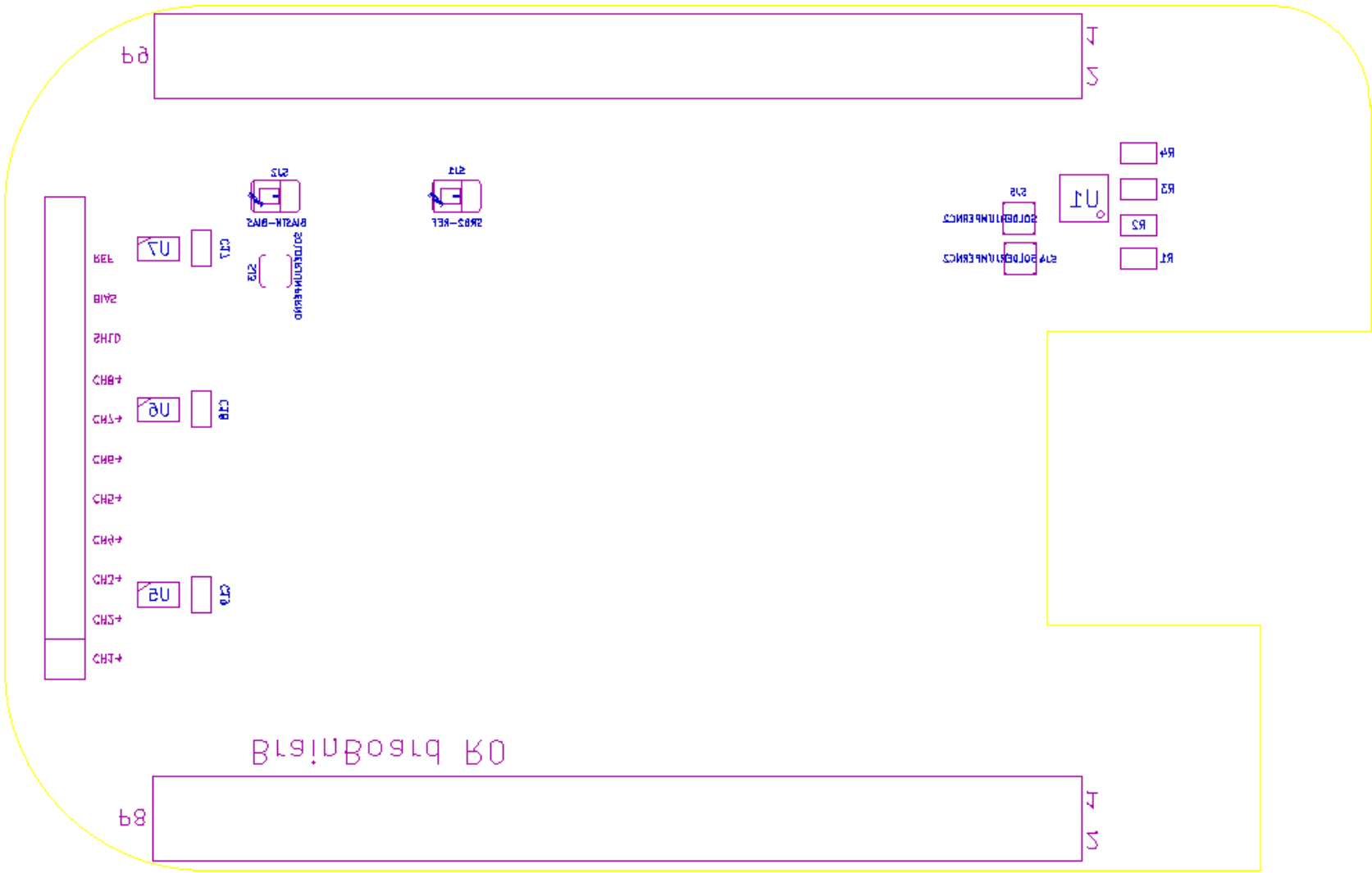
Copper 2 (Ground)



Copper 3 (Power)

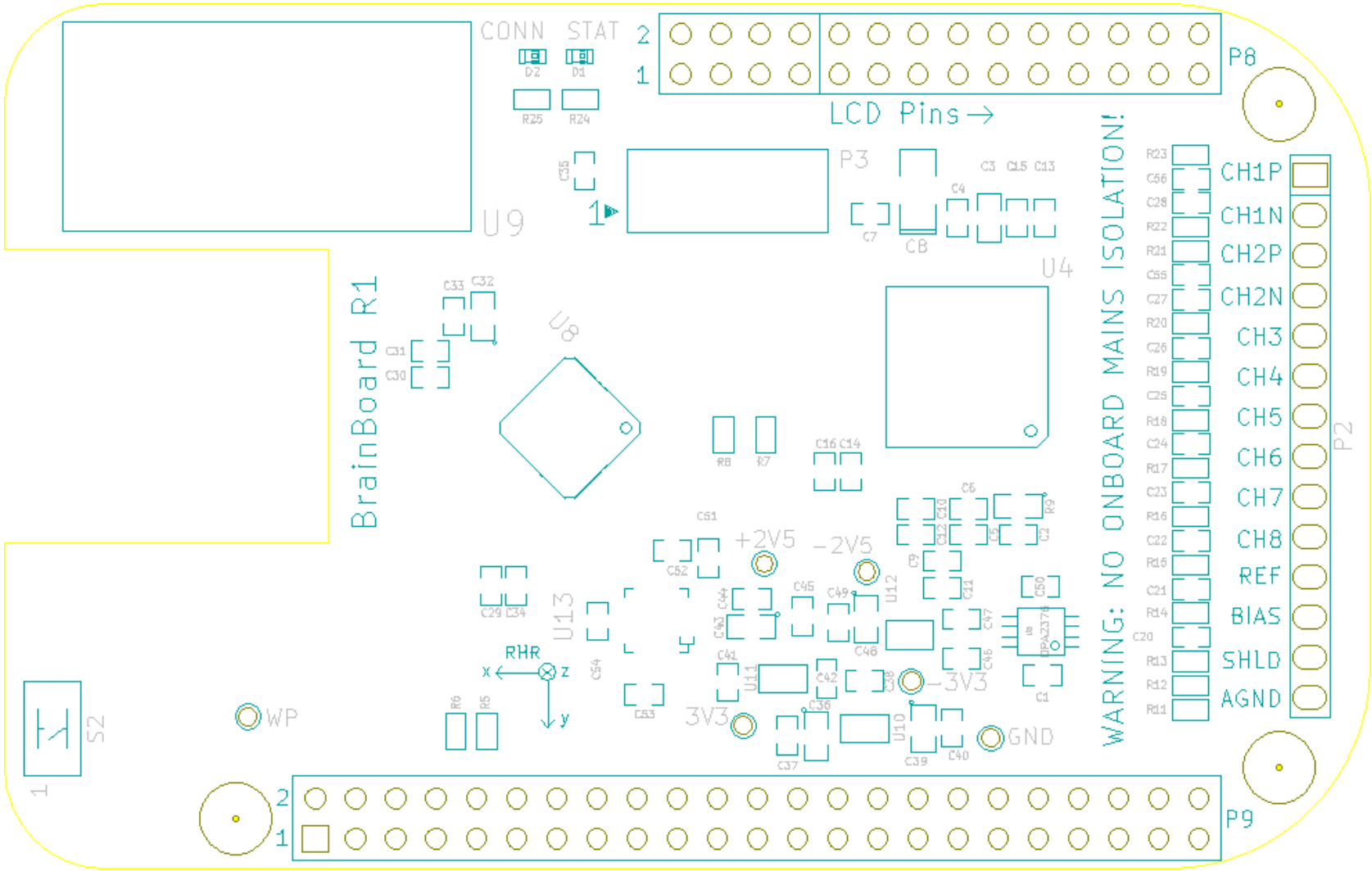


Bottom Copper

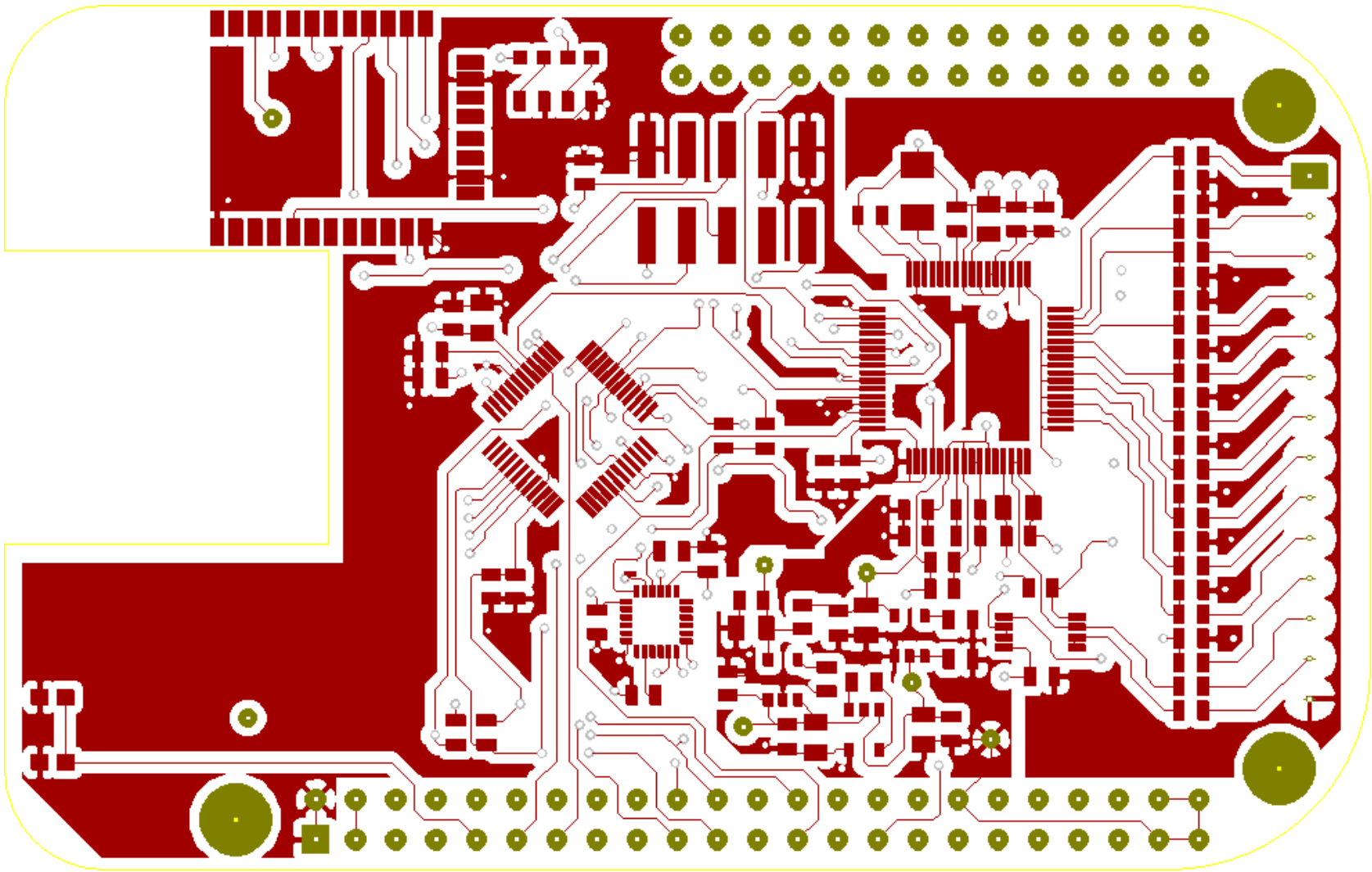


Bottom Silkscreen

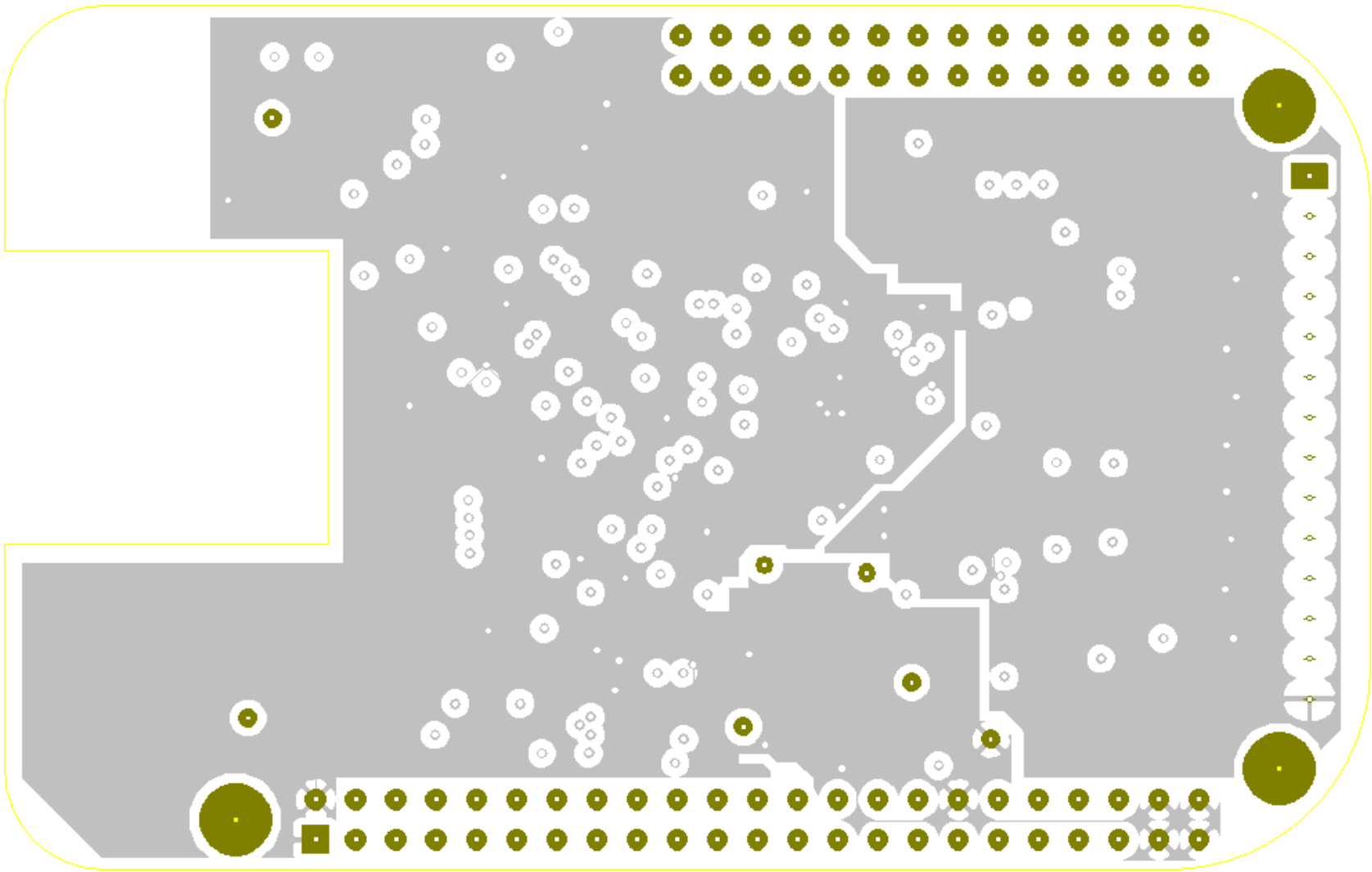
Brainboard R1



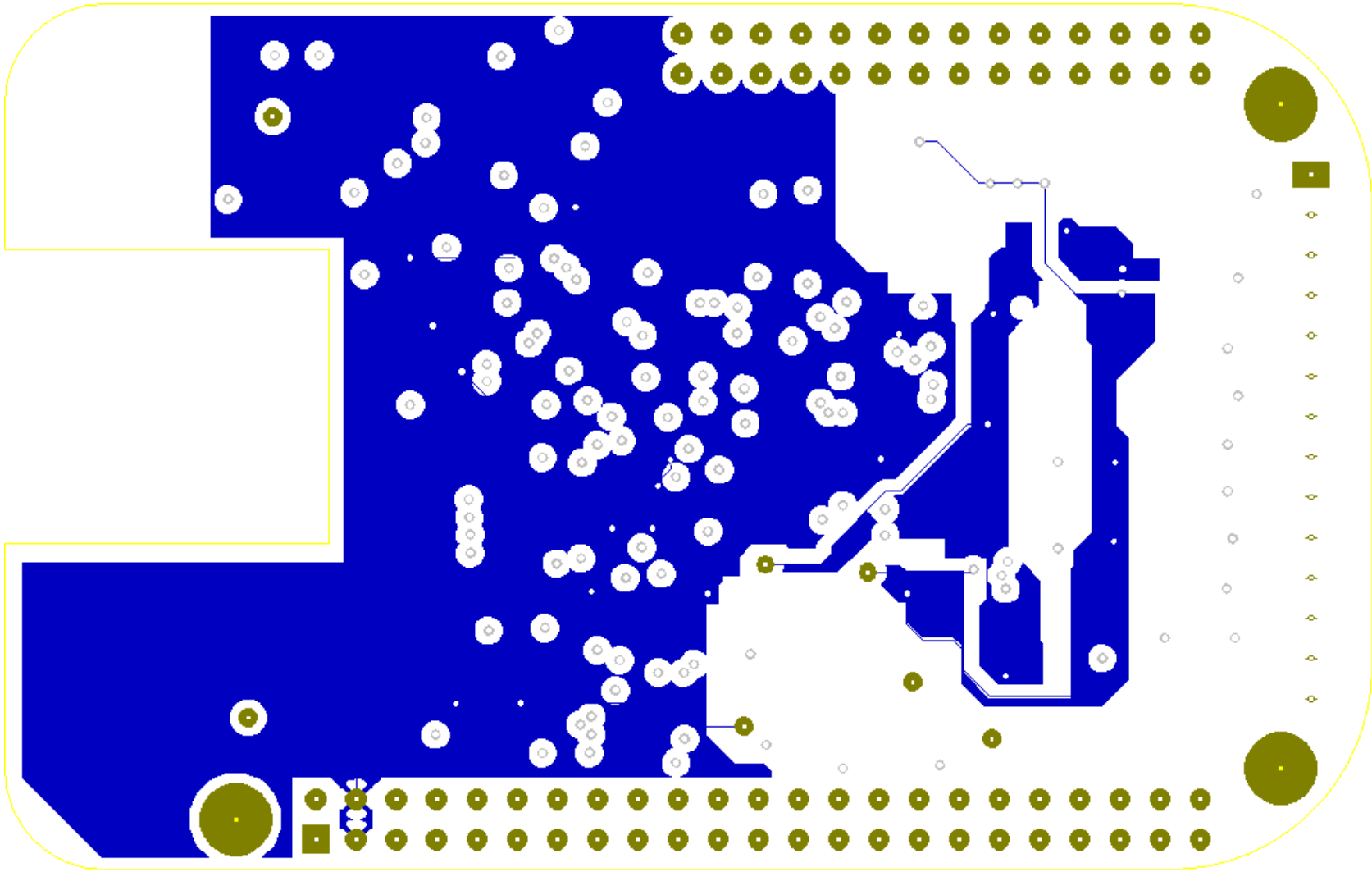
Top Silkscreen



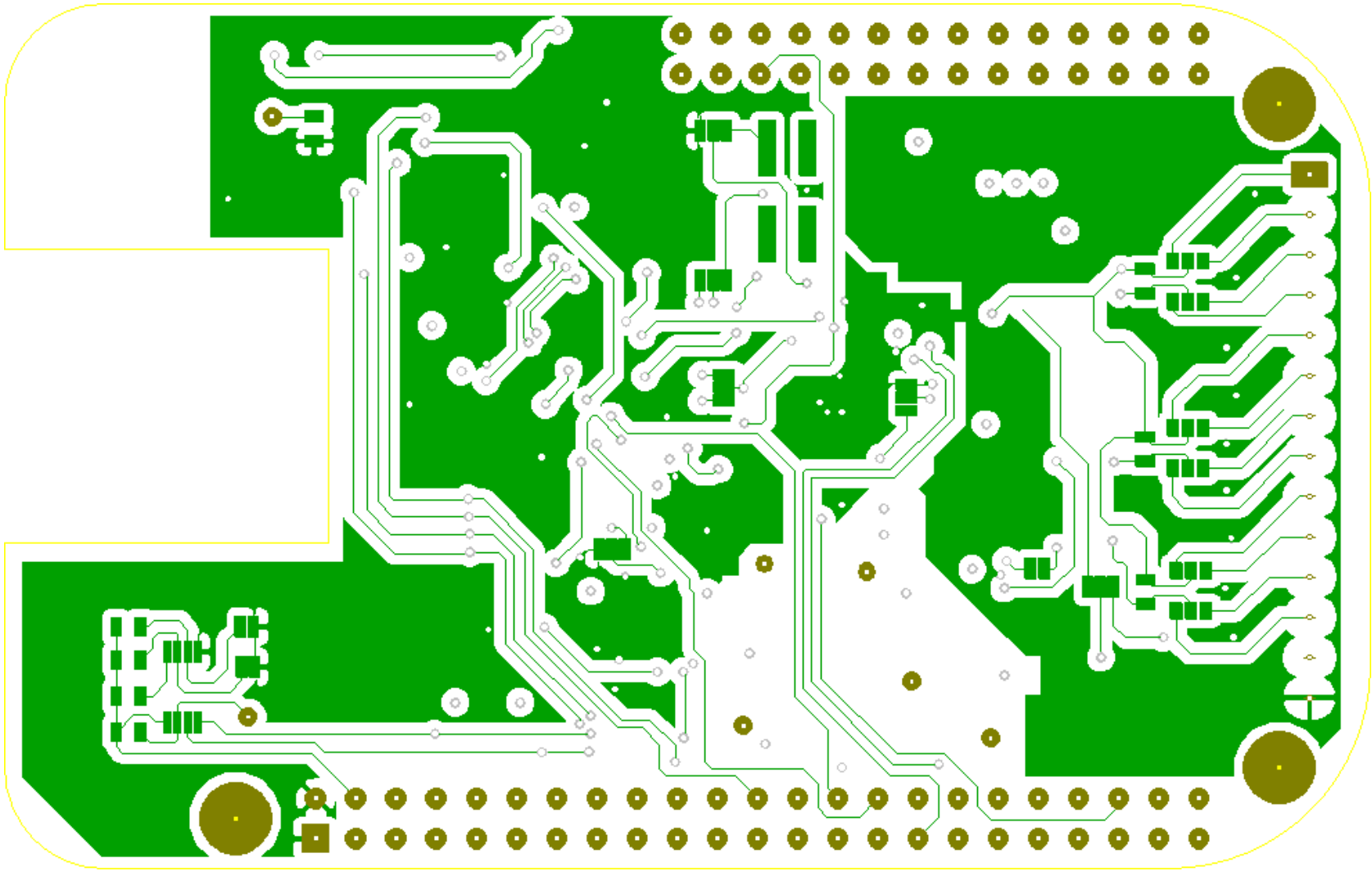
Top Copper



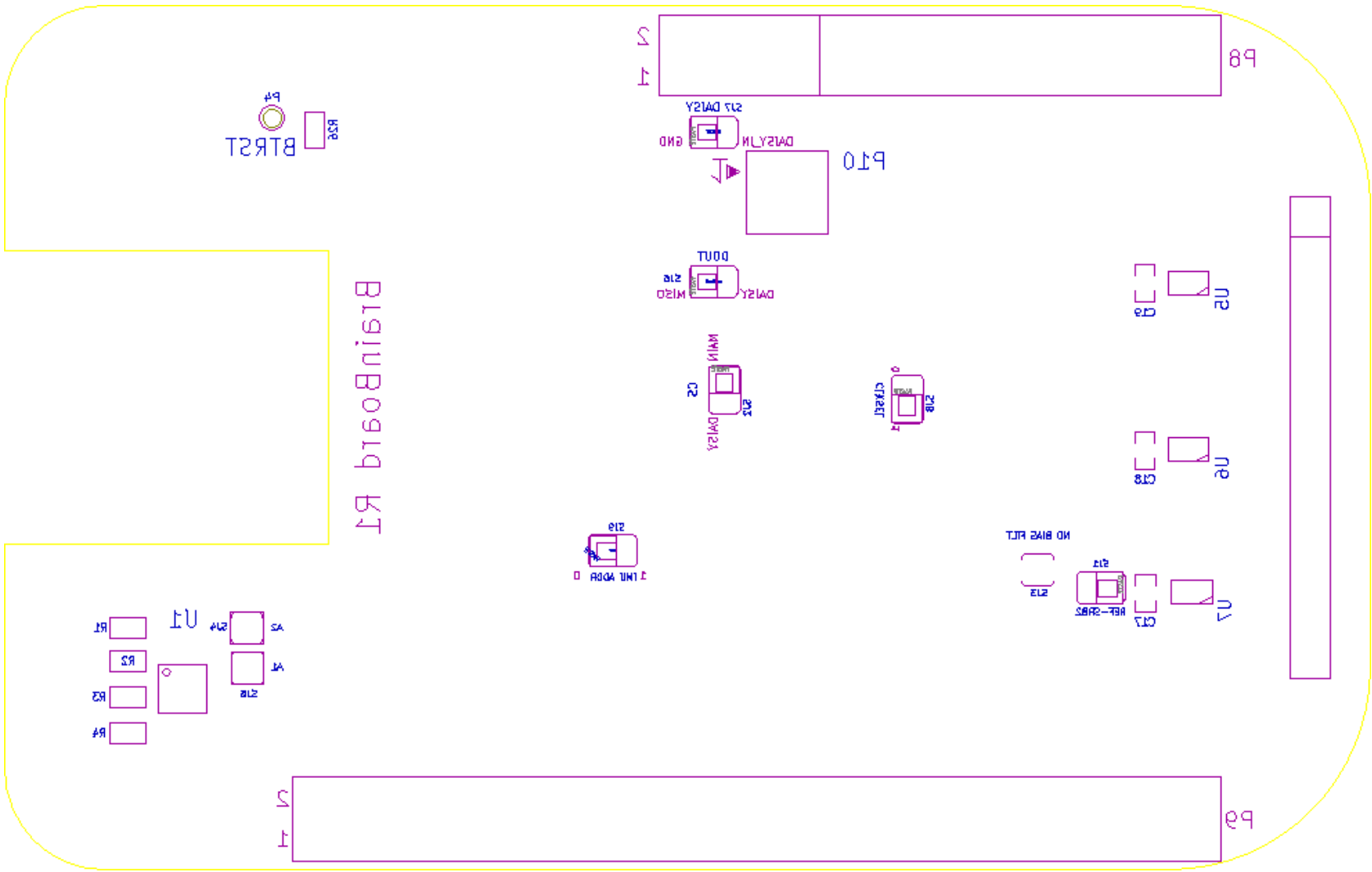
Copper 2 (Ground)



Copper 3 (Power)

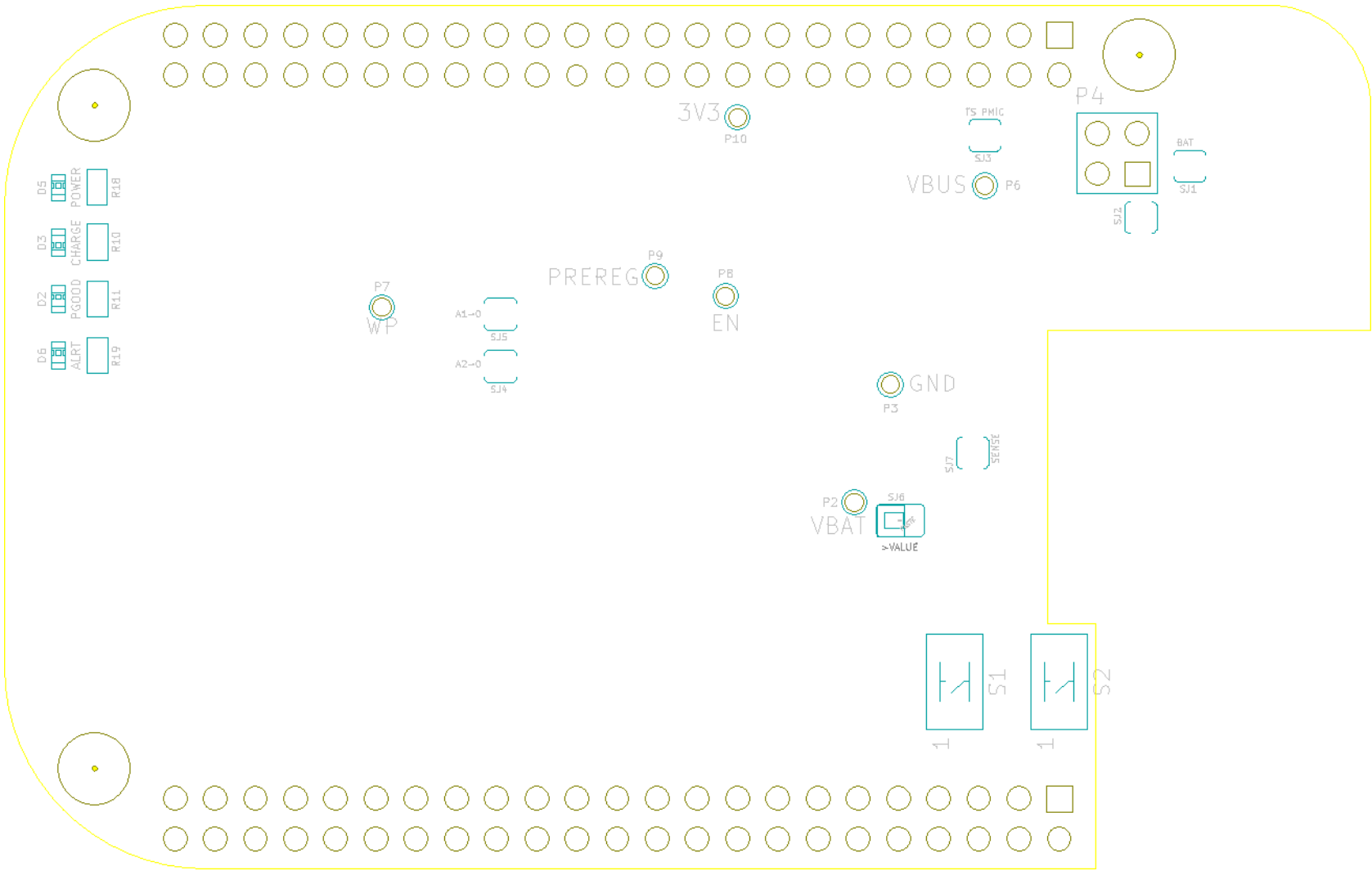


Bottom Copper

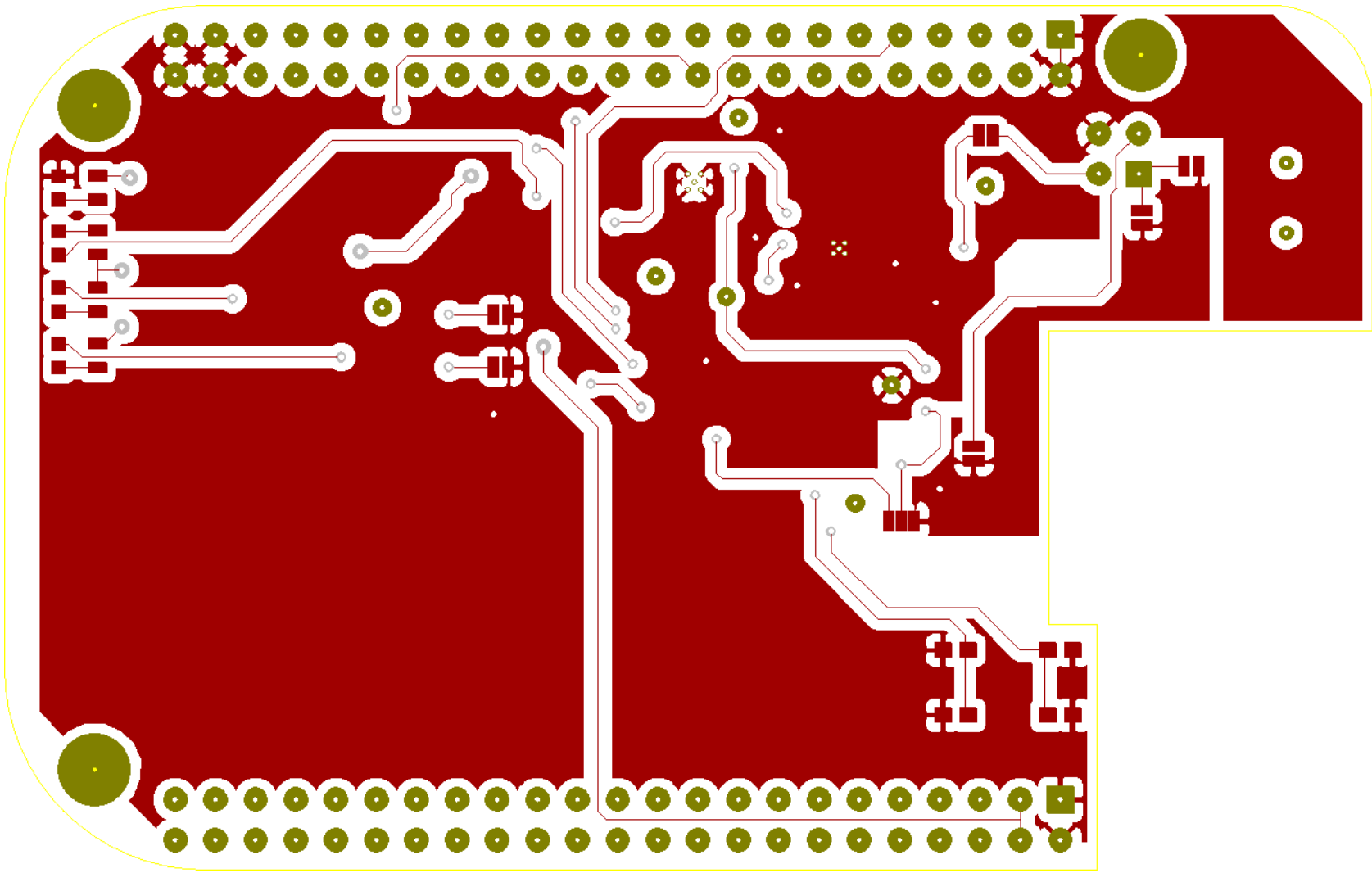


Bottom Silkscreen

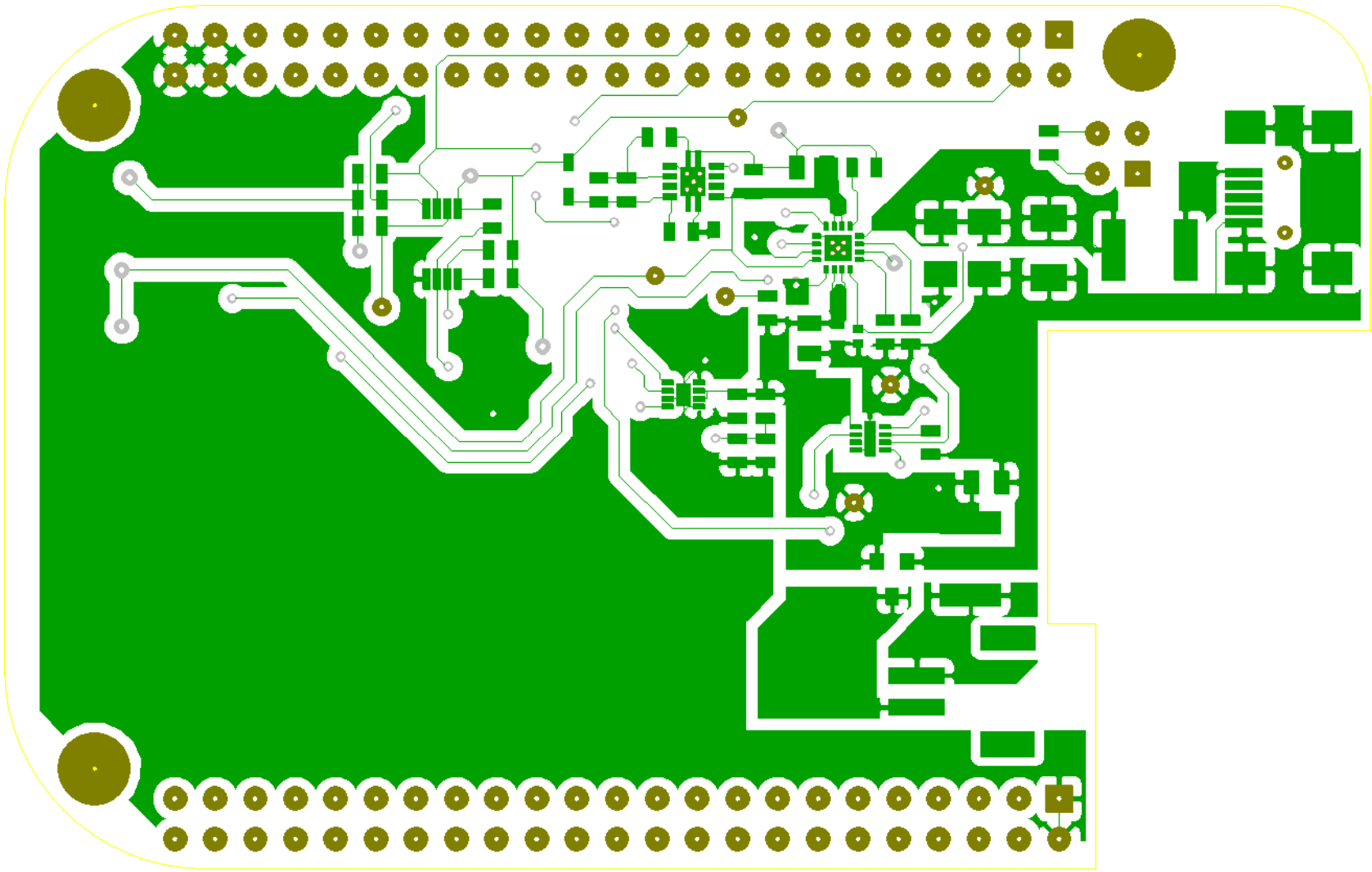
Battery Board



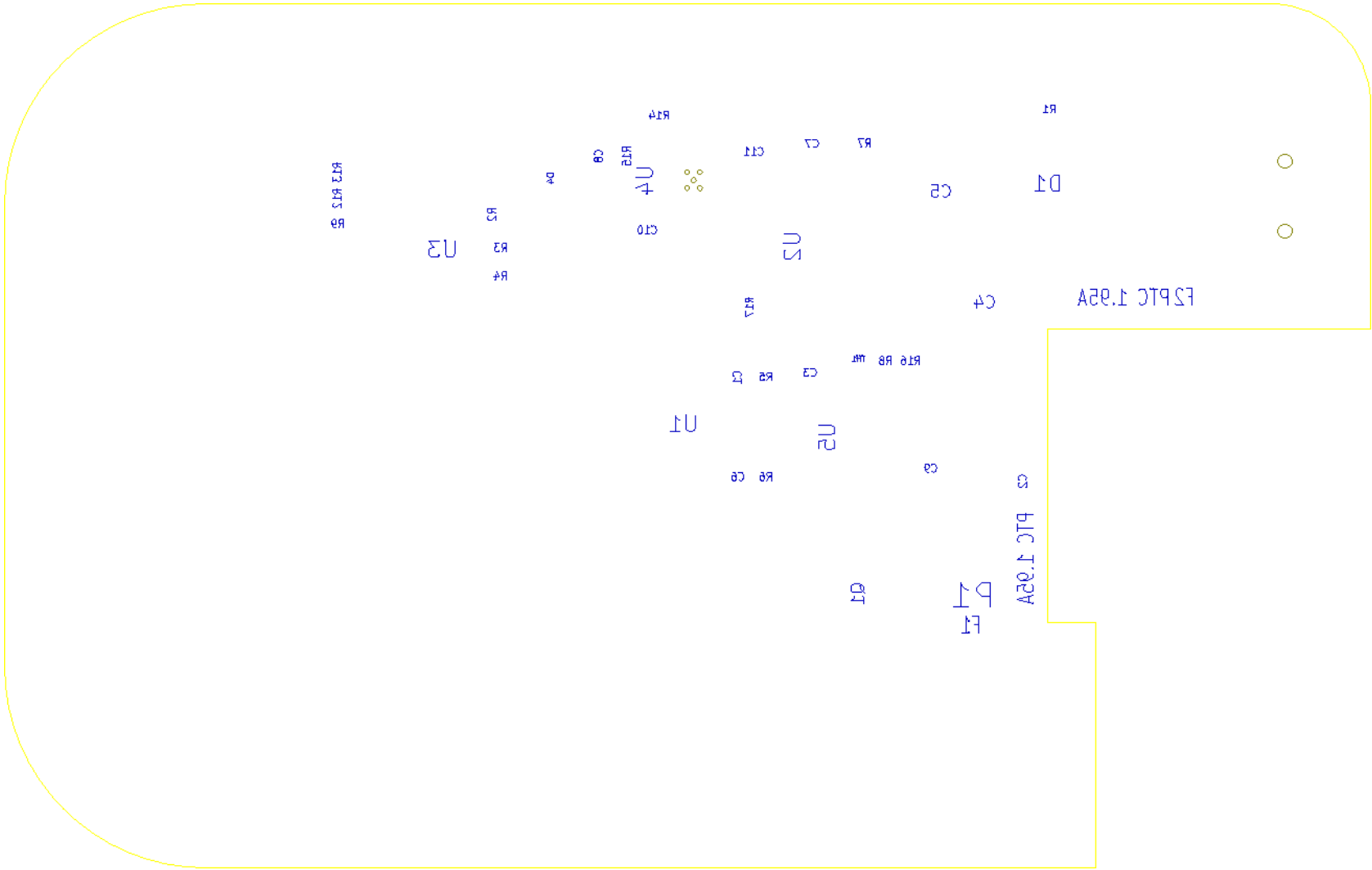
Top Silkscreen



Top Copper

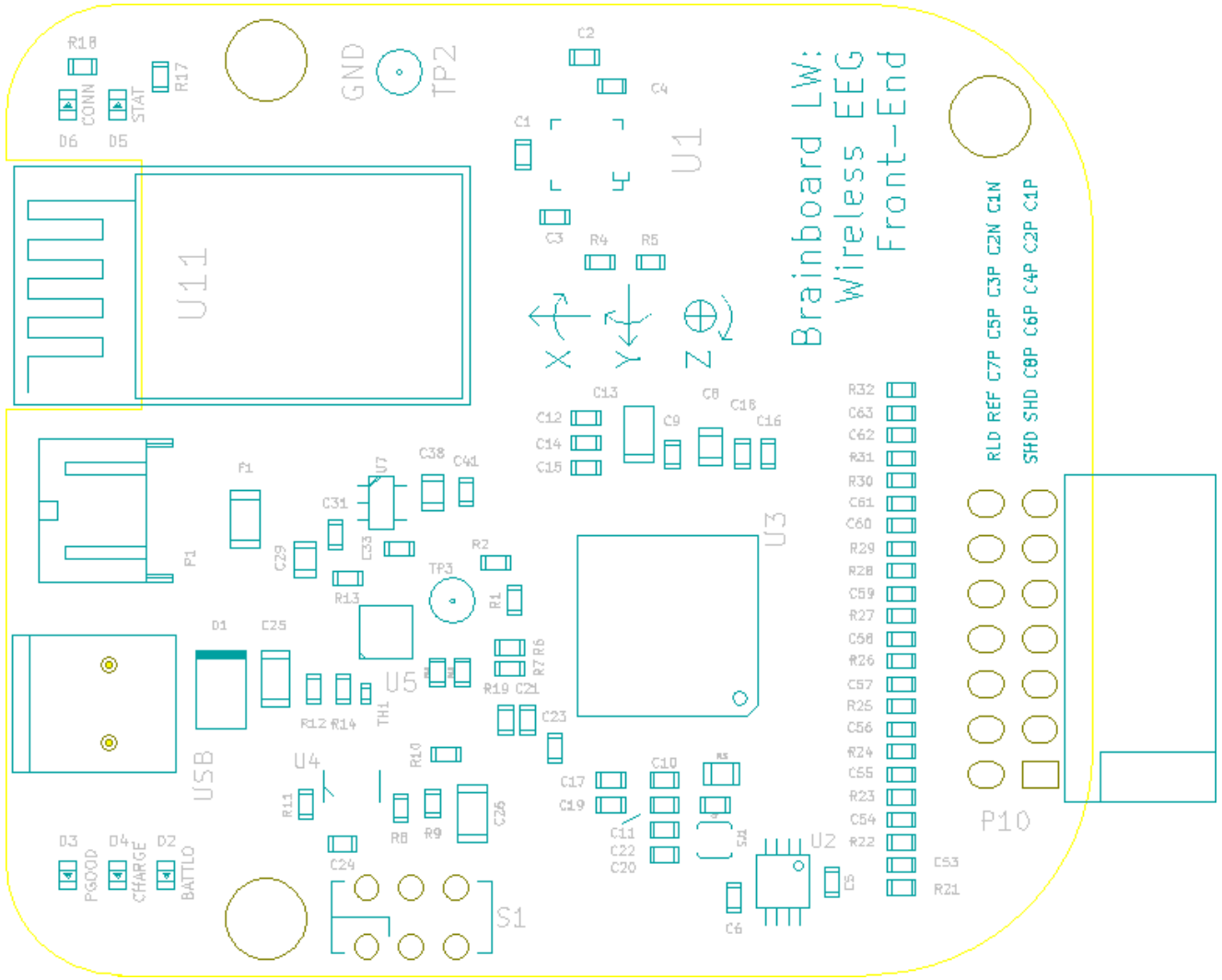


Bottom Copper

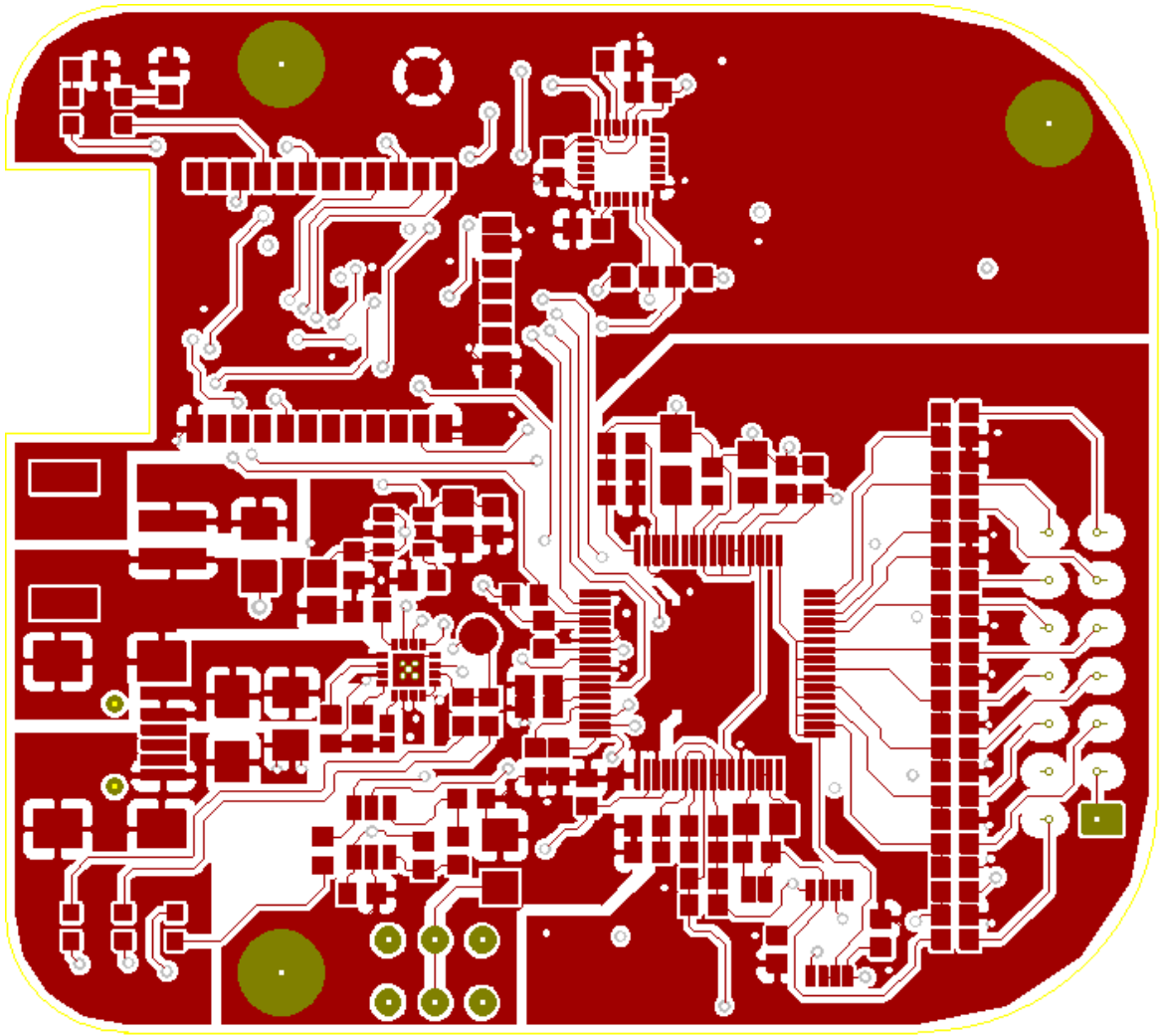


Bottom Silkscreen

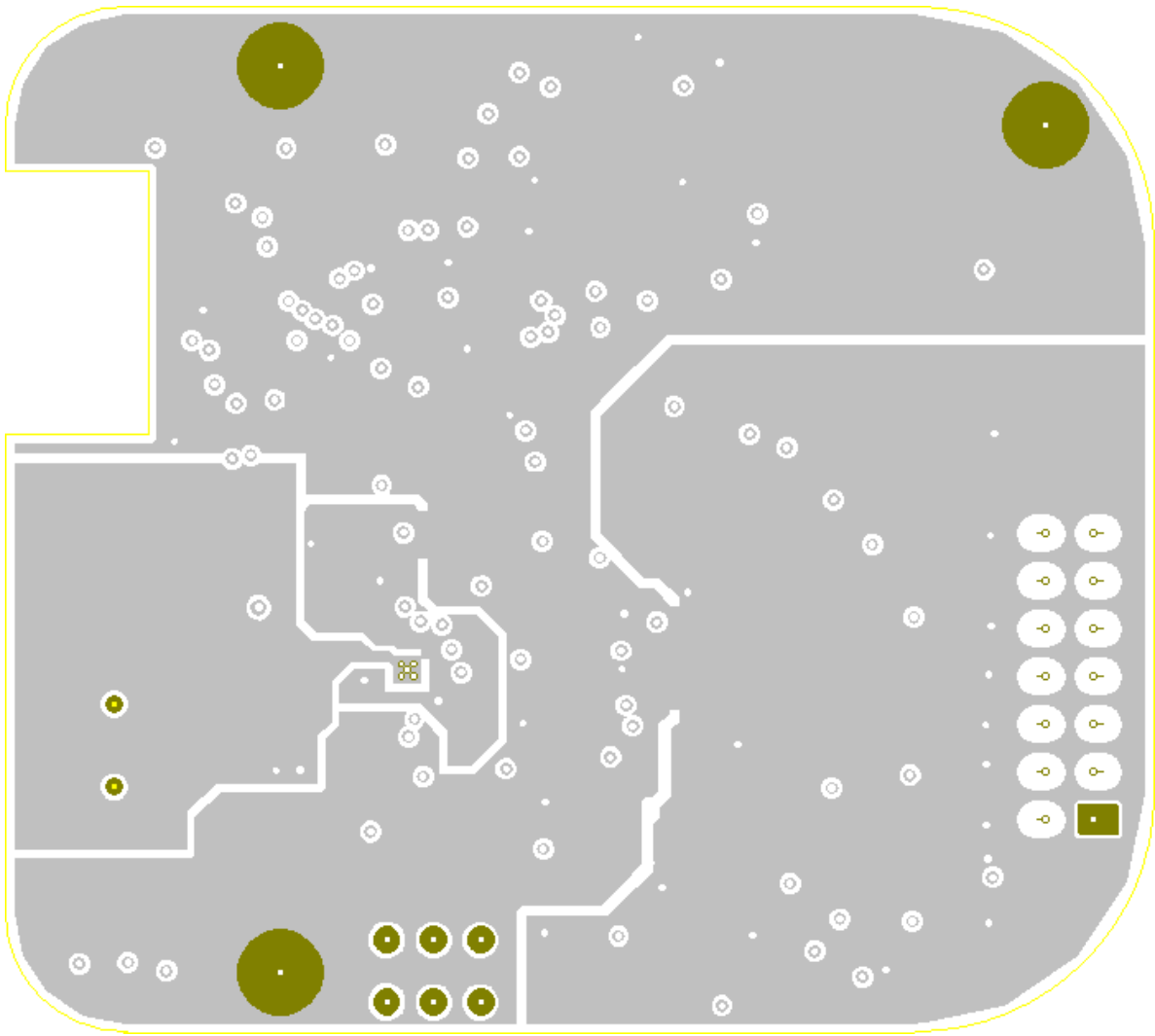
Brainboard LW



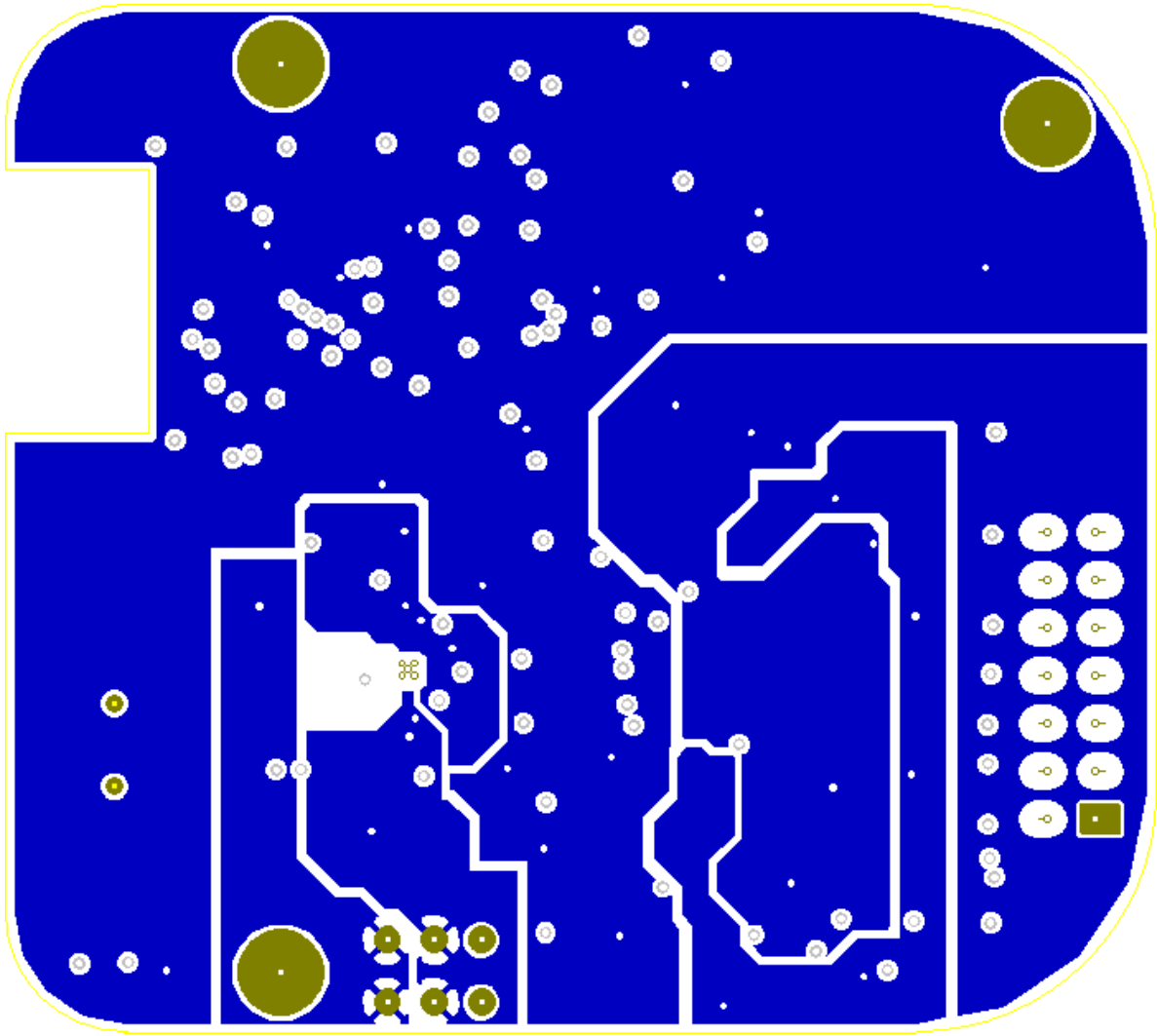
Top Silkscreen



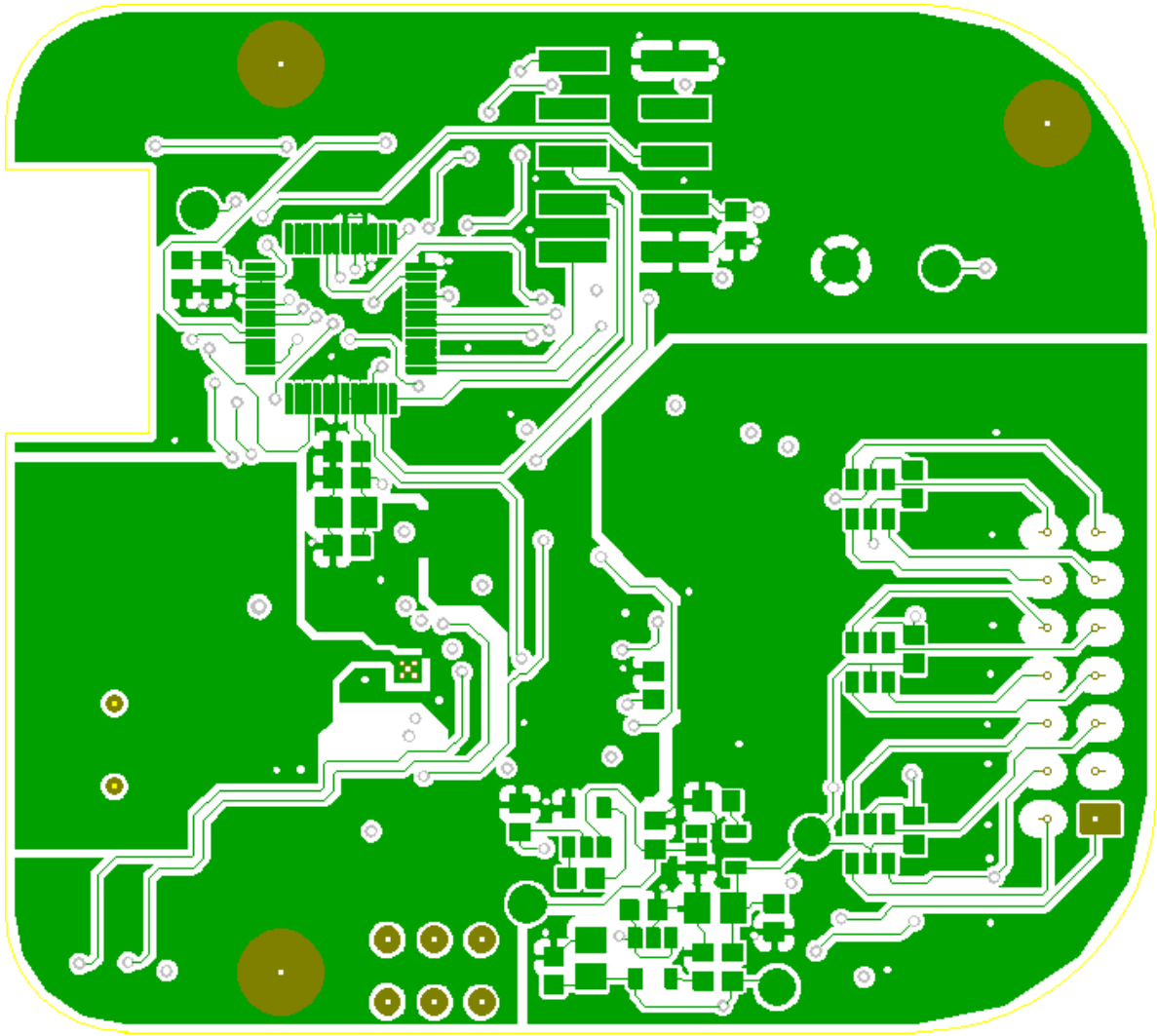
Top Copper



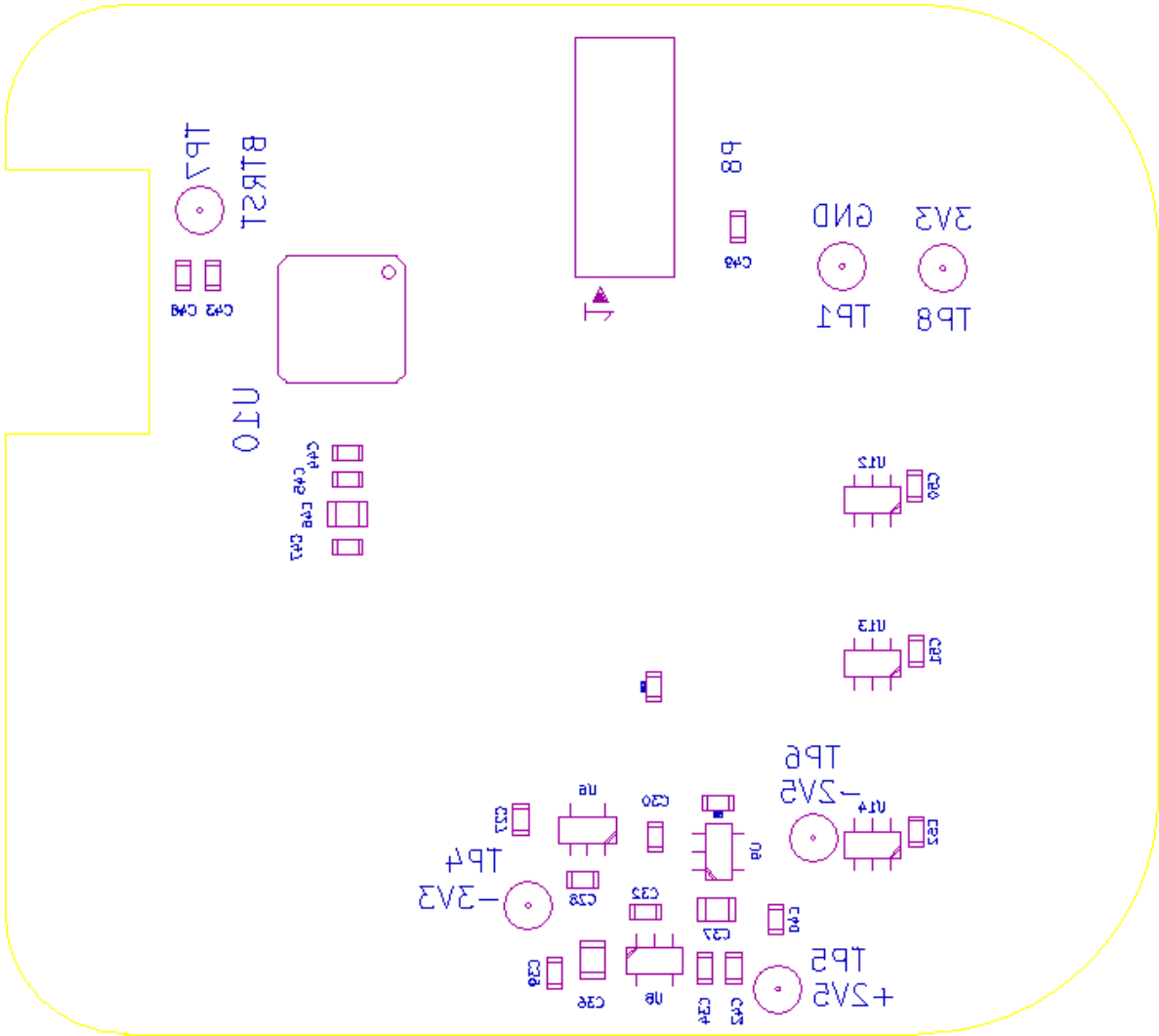
Copper 2 (Ground)



Copper 3 (Power)



Bottom Copper



Bottom Silkscreen

Appendix C

Firmware Code

C:\Documents and Settings\kellygs\My Documents\Atmel Studio\6.1\brainboard_fw\brainboard_fw\src\main.c1

```
/**
 *  \file main.c
 *
 *  \brief Brainboard setup and main loop.
 *  \author Graham Kelly
 *  \version 1.0
 *  \date August 2014
 *
 *  All Atmel Software Framework libraries used herein are copyright Atmel and
 *  subject to their appropriate licenses, which allow free redistribution with
 *  some restrictions. These restrictions are listed in their appropriate files.
 *
 *  \page License
 *
 *  Brainboard firmware code is placed under the MIT license
 *  Copyright (c) 2014 Graham Kelly
 *
 *  Permission is hereby granted, free of charge, to any person obtaining a copy
 *  of this software and associated documentation files (the "Software"), to deal
 *  in the Software without restriction, including without limitation the rights
 *  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 *  copies of the Software, and to permit persons to whom the Software is
 *  furnished to do so, subject to the following conditions:
 *
 *  The above copyright notice and this permission notice shall be included in
 *  all copies or substantial portions of the Software.
 *
 *  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 *  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 *  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 *  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 *  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 *  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 *  THE SOFTWARE.
 *
 */

#include <asf.h>
#include "brainboard.h"
#include "comms.h"
#include "ads1299.h"
#include "mpu60x0.h"

#define FRAME_SIZE                          128                                 // Size of each filtered segment (frame)
#define DSP_PREBUFFER_NB_SAMPLES            256                                 // # samples buffered between ADC and DSP engine
#ifndef FILT_COEFF_LENGTH
#define FILT_COEFF_LENGTH                   65
#endif
#define FILT_OUTPUT_SIZE                    (FRAME_SIZE-FILT_COEFF_LENGTH+1)    // Size of filter output vector
#if ((FILT_OUTPUT_SIZE % 4) != 0)
#error Improper filter output buffer size                                       /* Must ensure FILT_OUTPUT_SIZE is divisible by 4
 * to meet Atmel's optimized DSP library specs
 */
#endif

/* BCI state machine */
/* Ultimately this should have more states for different paradigms */
typedef enum {
    DATA_STREAM = 0,
    DSP_PROCESSING = 1
} processing_t;
processing_t processing_type;
```

```
 63
 64 /* Communication flags */
 65 volatile uint32_t frame_complete_flag;
 66
 67 /* Communication counters and buffer management */
 68 volatile uint32_t usart_tx_buffer_idx;
 69 uint32_t spi_buf_write_idx, spi_buf_read_idx;
 70
 71 /* Buffers for data shuttling */
 72 volatile uint8_t data_usart_rx_dma_buffer[3];
 73 volatile uint32_t spi_rx_ads1299_status;
 74 volatile int32_t spi_rx_data_buffer[DSP_PREBUFFER_NB_SAMPLES][MAX_EEG_CHANNELS];
 75
 76 /* Buffers for digital filtering: not currently implemented */
 77 __attribute__((aligned(4))) dsp16_t dsp_filter_input_buffer[FRAME_SIZE];
 78 __attribute__((aligned(4))) dsp16_t dsp_filter_output_buffer[FRAME_SIZE-FILT_COEFF_LENGTH+1];
 79 /* Average value for mu frequency band: not currently implemented */
 80 dsp32_t mu_band_avg;
 81
 82 /* Data packet struct */
 83 bboard_data24bit_packet_t data_usart_tx_packet =
 84 {
 85     .start      = 0xA55A,
 86     .packetnum  = 0,
 87     .eegstatus  = {0, 0, 0},
 88     // .eegdata will be init with loop in main()
 89     // .imudata will be init with loop in main()
 90 };
 91
 92 /* Function prototypes */
 93 void config_gpios(void);
 94 void config_eic(void);
 95 void config_dma(void);
 96 void config_spi(void);
 97 void config_usart(void);
 98 status_code_t config_twim(void);
 99 status_code_t config_rn42(volatile avr32_usart_t*, usart_options_t*);
100 status_code_t ads1299_check_device_id(uint8_t);
101
102 __attribute__((__interrupt__)) static void eic_handler(void)
103 {
104     Disable_global_interrupt();
105     eic_clear_interrupt_line(&AVR32_EIC, ADS1299_DRDY_INT);
106
107     if (processing_type == DATA_STREAM)
108     /* If we're in data stream mode, there is no intermediate buffering, so write directly to USART TX ↙
        packet */
109     {
110         /* Read back 24-bit status word and 8 channels of 24-bit data */
111         ads1299_rdata24_packet(SPI_ADS1299_MAIN_CHIPNUM, usart_tx_buffer_idx, &data_usart_tx_packet);
112         if (gpio_get_pin_value(MPU6050_INT_PIN))
113         {
114             /* Update inertial values Ax, Ay, Az, Gx, Gy, Gz */
115             mpu6050_rdata16_packet(MPU6050_DEFAULT_ADDRESS, usart_tx_buffer_idx, &data_usart_tx_packet) ↙
    ;
116         }
117         if (++usart_tx_buffer_idx == DATA_USART_TX_BUFFER_SIZE)
118         {
119             usart_tx_buffer_idx = 0;
120             pdca_enable(DATA_USART_TX_PDCA_CHANNEL);
121         }
122     }
123     else
124     /* We have to pre-buffer the data for processing */
125     {
126         /* Read back 24-bit status word and 8 channels of 24-bit data; store in 32-bit ints */
```

```
127         ads1299_rdata32_generic(SPI_ADS1299_MAIN_CHIPNUM, spi_buf_write_idx, spi_rx_ads1299_status,
    spi_rx_data_buffer);
128         /* We could add a step to read the inertial data here, but since we're not processing it for
    this proof of
129             * concept, it will be omitted
130             */
131         spi_buf_write_idx = (spi_buf_write_idx+1) % DSP_PREBUFFER_NB_SAMPLES;
132         if (spi_buf_write_idx % FRAME_SIZE == 0)
133         {
134             frame_complete_flag = 1;
135         }
136     }
137 }
138
139 __attribute__((__interrupt__)) static void pdca_data_usart_tx_handler(void)
140 {
141     pdca_disable(DATA_USART_TX_PDCA_CHANNEL);
142     if (processing_type == DATA_STREAM)
143     {
144         pdca_load_channel(DATA_USART_TX_PDCA_CHANNEL, (void *)&data_usart_tx_packet, sizeof
    (data_usart_tx_packet));
145         /* Circularly increment packet number */
146         data_usart_tx_packet.packetnum = (data_usart_tx_packet.packetnum+1) % 128;
147     }
148     else
149     {
150         mu_band_avg = 0;
151         pdca_load_channel(DATA_USART_TX_PDCA_CHANNEL, (void *)&mu_band_avg, sizeof(mu_band_avg));
152     }
153 }
154
155 __attribute__((__interrupt__)) static void pdca_data_usart_rx_handler(void)
156 {
157     uint8_t readback;
158
159     Disable_global_interrupt();
160
161     DATA_USART->cr = AVR32_USART_CR_RTSDIS_MASK;
162
163     /* To simplify DMA buffer implementation (i.e. to prevent me from having to code a timeout ISR
164      * when the user sends too few bytes to fill the buffer), multiple register reads/writes are not
165      * currently supported. For the sake of compatibility with the MATLAB command format of the
    previous
166      * implementation on Arduino, RREG and WREG commands are still 3 bytes long, but the second byte is
167      * ignored.
168      *
169      * !!!! For the following code to work, all commands MUST be 3 bytes long !!!!
170      */
171
172     /* Code would go here to parse command switching between data streaming and frequency band
    extraction
173      * modes. Make sure value of this command is not included in set of ADS1299 commands.
174      */
175
176     switch (data_usart_rx_dma_buffer[0] & 0xE0)
177     {
178         case 0x40:  // WREG
179             ads1299_wreg(SPI_ADS1299_MAIN_CHIPNUM, data_usart_rx_dma_buffer[0] & 0x1F,
    data_usart_rx_dma_buffer[2]);
180             break;
181         case 0x20:  // RREG
182             ads1299_rreg(SPI_ADS1299_MAIN_CHIPNUM, data_usart_rx_dma_buffer[0] & 0x1F, &readback);
183             usart_putchar(DATA_USART, (int)readback);
184             while(!usart_tx_empty(DATA_USART));
185             break;
186         default:
```

```
187             ads1299_send_byte(SPI_ADS1299_MAIN_CHIPNUM, data_usart_rx_dma_buffer[0]);
188             break;
189     }
190     if (data_usart_rx_dma_buffer[0] == ADS1299_OPC_STOP)
191     {
192         usart_tx_buffer_idx = 0;
193         pdca_disable(DATA_USART_TX_PDCA_CHANNEL);
194     }
195     pdca_load_channel(DATA_USART_RX_PDCA_CHANNEL, (void *)data_usart_rx_dma_buffer, 3);
196     DATA_USART->cr = AVR32_USART_CR_RTSEN_MASK;
197
198     Enable_global_interrupt();
199 }
200
201 void config_dma(void)
202 {
203     /* DMA configuration options for USART3 (RN-42 Bluetooth) data transmission.
204      * RN-42 is connected to USART3 on the UC3. The USART operates in byte mode, so transfer size must ↙
        be BYTE.
205      */
206     const pdca_channel_options_t pdca_data_usart_tx_opt =
207     {
208         .pid            = DATA_USART_TX_PDCA_PID,
209         .transfer_size  = PDCA_TRANSFER_SIZE_BYTE,
210         .r_addr         = NULL,
211         .r_size         = 0
212     };
213
214     /* DMA configuration options for USART3 (RN-42 Bluetooth) data receipt.
215      * RN-42 is connected to USART3 on the UC3. The USART operates in byte mode, so transfer size must ↙
        be BYTE.
216      */
217     const pdca_channel_options_t pdca_data_usart_rx_opt =
218     {
219         .pid            = DATA_USART_RX_PDCA_PID,
220         .transfer_size  = PDCA_TRANSFER_SIZE_BYTE,
221         .addr           = (void *)data_usart_rx_dma_buffer,
222         .size           = 3,
223         .r_addr         = NULL,
224         .r_size         = 0
225     };
226
227     bool global_irq_enabled = cpu_irq_is_enabled();
228     if (global_irq_enabled) {
229         /* Turn off interrupts while we set stuff up */
230         Disable_global_interrupt();
231     }
232
233     /* Init PDCA and register corresponding interrupts */
234     /* USART3 receipt */
235     pdca_init_channel(DATA_USART_RX_PDCA_CHANNEL, &pdca_data_usart_rx_opt);
236     INTC_register_interrupt((__int_handler)&pdca_data_usart_rx_handler, AVR32_PDCA_IRQ_0 +              ↙
        DATA_USART_RX_PDCA_CHANNEL, AVR32_INTC_INT3);
237     pdca_enable_interrupt_transfer_complete(DATA_USART_RX_PDCA_CHANNEL);
238
239     /* USART3 transmission */
240     pdca_init_channel(DATA_USART_TX_PDCA_CHANNEL, &pdca_data_usart_tx_opt);
241     INTC_register_interrupt((__int_handler)&pdca_data_usart_tx_handler, AVR32_PDCA_IRQ_0 +              ↙
        DATA_USART_TX_PDCA_CHANNEL, AVR32_INTC_INT2);
242     pdca_load_channel(DATA_USART_TX_PDCA_CHANNEL, (void *)&data_usart_tx_packet, sizeof                 ↙
        (data_usart_tx_packet));
243     pdca_enable_interrupt_transfer_complete(DATA_USART_TX_PDCA_CHANNEL);
244
245     if (global_irq_enabled) {
246         /* Enable all interrupts */
247         Enable_global_interrupt();
```

```
248     }
249 }
250
251 void config_eic(void)
252 {
253
254     const eic_options_t eic_drdy_opt =
255     {
256         .eic_mode    = EIC_MODE_EDGE_TRIGGERED,
257         .eic_level   = EIC_EDGE_FALLING_EDGE,
258         .eic_filter  = EIC_FILTER_DISABLED,
259         .eic_async   = EIC_ASYNCH_MODE,
260         .eic_line    = ADS1299_DRDY_INT
261     };
262
263     bool global_irq_enabled = cpu_irq_is_enabled();
264     if (global_irq_enabled) {
265         Disable_global_interrupt();
266     }
267
268     /* Register external DRDY interrupt */
269     INTC_register_interrupt((__int_handler)&eic_handler, ADS1299_DRDY_IRQ, AVR32_INTC_INT0);
270     eic_init(&AVR32_EIC, &eic_drdy_opt, 1);
271     eic_enable_line(&AVR32_EIC, eic_drdy_opt.eic_line);
272     eic_enable_interrupt_line(&AVR32_EIC, eic_drdy_opt.eic_line);
273
274     if (global_irq_enabled) {
275         Enable_global_interrupt();
276     }
277 }
278
279 void config_spi(void)
280 {
281     const spi_options_t spi_opt =
282     {
283         .reg            = SPI_ADS1299_MAIN_CHIPNUM,     // "Master" ADS1299 when daisy-chaining:       ↙
    connected to CS0
284         .baudrate       = 1000000,                      // 1 MHz preferred baudrate
285         .bits           = 8,                            // 8 bits per character (range is 8 to 16)
286         .spck_delay     = 0,                            // Delay before first clock pulse after        ↙
    selecting slave (in PBA clock periods)
287         .trans_delay    = 0,                            // Delay between each transfer/character (in   ↙
    PBA clock periods)
288         .stay_act       = true,                         // Sets this chip to stay active after last    ↙
    transfer to it
289         .spi_mode       = SPI_MODE_1,                   // Which SPI mode to use (ADS1299 is mode 1)
290         .modfdis        = true                          // Disable the mode fault detection
291     };
292
293     /* Init SPI module as master */
294     spi_initMaster(SPI_ADDRESS, &spi_opt);
295
296     /* Setup configuration for chip connected to CS0 */
297     spi_setupChipReg(SPI_ADDRESS, &spi_opt, sysclk_get_pba_hz());
298
299     /* Allow the module to transfer data */
300     spi_enable(SPI_ADDRESS);
301 }
302
303 void config_gpios(void)
304 {
305     /* Debug/timing pin for oscilloscope */
306     //gpio_enable_gpio_pin(AVR32_PIN_PA11);
307
308     #if (BRAINBOARD_REV > 0)
309     /* Config GPIOs for I2C/TWI comms with inertial measurement unit */
```

```
310     /* First we apparently have to assign the default TWIM pins as something so that we can
311      * use different ones. Stupid. Tore my hair out before I got this one...
312      */
313     gpio_enable_gpio_pin(AVR32_PIN_PB05);                                          // Default
        TWIMS0 TWCK
314     gpio_enable_gpio_pin(AVR32_PIN_PA21);                                          // Default
        TWIMS0 TWD
315     gpio_enable_module_pin(AVR32_TWIMS0_TWCK_0_0_PIN, AVR32_TWIMS0_TWCK_0_0_FUNCTION);  // PA04
316     gpio_enable_module_pin(AVR32_TWIMS0_TWD_0_2_PIN, AVR32_TWIMS0_TWD_0_2_FUNCTION);    // PA05
317
318     /* MPU6050 interrupt pin: polled at each ADS1299 DRDY */
319     gpio_enable_gpio_pin(MPU6050_INT_PIN);                                         // PA10
320     gpio_configure_pin(MPU6050_INT_PIN, GPIO_DIR_INPUT);
321     #endif
322
323     /* Config GPIOs for USART2 (wired data) */
324     gpio_enable_module_pin(AVR32_USART2_TXD_0_1_PIN, AVR32_USART2_TXD_0_1_FUNCTION);    // PA06
325     gpio_enable_pin_pull_up(AVR32_USART2_TXD_0_1_PIN);
326     gpio_enable_module_pin(AVR32_USART2_RXD_0_1_PIN, AVR32_USART2_RXD_0_1_FUNCTION);    // PA07
327
328     /* Config GPIOs for USART3 (RN-42 Bluetooth data) */
329     gpio_enable_module_pin(AVR32_USART3_TXD_0_1_PIN, AVR32_USART3_TXD_0_1_FUNCTION);    // PB06
330     gpio_enable_pin_pull_up(AVR32_USART3_TXD_0_1_PIN);
331     gpio_enable_module_pin(AVR32_USART3_RXD_0_1_PIN, AVR32_USART3_RXD_0_1_FUNCTION);    // PB07
332     gpio_enable_module_pin(AVR32_USART3_RTS_0_1_PIN, AVR32_USART3_RTS_0_1_FUNCTION);    // PB08
333     gpio_enable_module_pin(AVR32_USART3_CTS_0_1_PIN, AVR32_USART3_CTS_0_1_FUNCTION);    // PB09
334
335     /* Config GPIOs for SPI */
336     gpio_enable_module_pin(AVR32_SPI_MISO_0_1_PIN, AVR32_SPI_MISO_0_1_FUNCTION);        // PB02
337     gpio_enable_module_pin(AVR32_SPI_MOSI_0_1_PIN, AVR32_SPI_MOSI_0_1_FUNCTION);        // PB03
338     gpio_enable_module_pin(AVR32_SPI_SCK_0_1_PIN, AVR32_SPI_SCK_0_1_FUNCTION);          // PB01
339     gpio_enable_module_pin(AVR32_SPI_NPCS_0_1_PIN, AVR32_SPI_NPCS_0_1_FUNCTION);        // PB00
340
341     /* RN-42 hardware reset pin */
342     gpio_enable_gpio_pin(BT_RESET_PIN);                                            // PB11
343     gpio_configure_pin(BT_RESET_PIN, GPIO_DIR_OUTPUT);
344
345     /* ADS1299 control pins */
346     #if (BRAINBOARD_REV == 0)
347     /* BrainBoard R1 doesn't connect START and RESET pins, uses software commands instead */
348     gpio_enable_gpio_pin(ADS1299_PIN_START);                                       // PA17
349     gpio_configure_pin(ADS1299_PIN_START, GPIO_DIR_OUTPUT);
350     gpio_enable_gpio_pin(ADS1299_PIN_RESET);                                       // PB04
351     gpio_configure_pin(ADS1299_PIN_RESET, GPIO_DIR_OUTPUT);
352     #endif
353     gpio_enable_gpio_pin(ADS1299_PIN_PWDN);                                        // PA09
354     gpio_configure_pin(ADS1299_PIN_PWDN, GPIO_DIR_OUTPUT);
355     /* ADS1299 DRDY interrupt pin */
356     gpio_enable_module_pin(ADS1299_PIN_DRDY, ADS1299_DRDY_INT_FN);                 // PA18
357 }
358
359 status_code_t config_twim(void)
360 {
361     twim_options_t twim_opt =
362     {
363         .speed  = I2C_MASTER_SPEED_HZ,
364         .chip   = MPU6050_DEFAULT_ADDRESS,
365         .pba_hz = sysclk_get_pba_hz(),
366         .smbus  = false
367     };
368
369     sysclk_enable_peripheral_clock(I2C_MODULE);
370     return twim_master_init(I2C_MODULE, (const twim_options_t *)&twim_opt);
371 }
372
373 void config_usart(void)
```

```
374 {
375
376 //  const usart_options_t usartuc_opt =
377 //  {
378 //      .baudrate    = USARTUC_BAUDRATE,
379 //      .channelmode = USART_NORMAL_CHMODE,
380 //      .charlength  = 8,
381 //      .paritytype  = USART_NO_PARITY,
382 //      .stopbits    = USART_1_STOPBIT
383 //  };
384
385     usart_options_t usartbt_opt =
386     {
387         .baudrate    = USARTBT_DEFAULT_BAUDRATE,
388         .channelmode = USART_NORMAL_CHMODE,
389         .charlength  = 8,
390         .paritytype  = USART_NO_PARITY,
391         .stopbits    = USART_1_STOPBIT
392
393     };
394
395     /* Configure and enable USART module for inter-processor comms */
396 //  sysclk_enable_peripheral_clock(USARTUC_MODULE);
397 //  usart_init_rs232(USARTUC_MODULE, &usartuc_opt, sysclk_get_pba_hz());
398
399     /* Enable USART for Bluetooth module comms */
400     sysclk_enable_peripheral_clock(USARTBT_MODULE);
401     usart_init_hw_handshaking(USARTBT_MODULE, &usartbt_opt, sysclk_get_pba_hz());
402
403     /* Configure RN-42 Bluetooth module */
404     config_rn42(USARTBT_MODULE, &usartbt_opt);
405 }
406
407 status_code_t config_rn42(volatile avr32_usart_t* usart, usart_options_t* opt)
408 {
409
410     /* Reset-cycle RN-42 Bluetooth module */
411     gpio_clr_gpio_pin(BT_RESET_PIN);
412     delay_us(100);
413     gpio_set_gpio_pin(BT_RESET_PIN);
414     delay_ms(100);
415
416     /* Configure RN-42 */
417     /* Enter command mode */
418     usart_write_line(usart, "$$$");
419     while(!usart_tx_ready(usart));
420     delay_ms(15);
421
422     /* Reduce window for accepting device discovery requests (inquiry) to 80 ms. Default
423      * window is 160 ms (0x0100).
424      * The 16-bit hex value is multiplied by 0.625 ms to determine the interval. The minimum is
425      * 0x0012 (11.25 ms).
426      */
427     usart_write_line(usart, "SI,0080\r\n");
428     while(!usart_tx_ready(usart));
429     delay_ms(50);
430
431     /* Enable SNIFF mode with interval of 25 ms.
432      * In this mode, the RN-42 wakes up every [interval] to send/receive data. Data to transmit
433      * is buffered by the RN-42, so it shouldn't be lost (depending on buffer size). This saves
434      * a good amount of power.
435      * The 16-bit hex value is multiplied by 0.625 ms to determine the interval. Maximum is
436      * 0x7FFF, or about 20 s.
437      */
438     usart_write_line(usart, "SW,0028\r\n");
439     while(!usart_tx_ready(usart));
```

```
440     delay_ms(50);
441
442     /* Set all GPIO pins on RN-42 as inputs, reducing power.
443      *
444      * 16-bit hex value; first byte is mask to specify the GPIO number, second is command
445      * value, where 0 = input, 1 = output:
446      *
447      * PARAMETER[15:0] = MASK[7..0]<<8 | VALUE[7..0]
448      * e.g. to set GPIO5 (bit 5 of MASK) as an output, write 0b0010000000100000 = 0x2020
449      *      to set it as an input,                     write 0b0010000000000000 = 0x2000
450      *
451      * Sending 0x1000 sets ALL GPIOs to inputs (since GPIO4 is used for factory reset and
452      * isn't user-controllable).
453      *
454      * Here the STATUS and CONN LEDs are connected to GPIO5 and GPIO2, respectively.
455      * We may want to disable them to conserve power.
456      *
457      * Quick reference: 0x2424 sets GPIO5 and GPIO2 to outputs. 0x2400 sets them to inputs.
458      *
459      * NOTE: the S% command is stored on the flash, and GPIOs retain these settings on
460      * power cycle or reset. To make the change temporary, use S@ with the same argument.
461      * This change will be lost on a reboot, so it will need to be moved to after the next
462      * next command.
463      */
464     usart_write_line(usart, "S%,2424\r\n");
465     while(!usart_tx_ready(usart));
466     delay_ms(50);
467
468     /* Reboot RN-42 */
469     usart_write_line(usart, "R,1\r\n");
470     while(!usart_tx_ready(usart));
471     delay_ms(15);
472
473     return STATUS_OK;
474 }
475
476 status_code_t ads1299_check_device_id(uint8_t chipselect)
477 {
478     uint8_t id;
479
480     ads1299_rreg(chipselect, ADS1299_REGADDR_ID, &id);
481     /* ADS1299 should return ID of 0bXXXX1110 */
482     if ( (id & 0xF) != ADS1299_DEVICE_ID ) {
483         return ERR_IO_ERROR;
484     }
485     else {
486         return STATUS_OK;
487     }
488 }
489
490 int main(void)
491 {
492     volatile uint32_t k, n, m;
493
494     usart_tx_buffer_idx = 0;
495     spi_buf_write_idx = 0;
496     spi_buf_read_idx = 0;
497     mu_band_avg = 0;
498
499     frame_complete_flag = 0;
500
501     /* Initialize in data streaming mode; host computer may change later */
502     processing_type = DATA_STREAM;
503
504     /* Set up basic functions: clocks, GPIOs, sleep settings */
505     sysclk_init();
```

```
506     sleepmgr_init();
507     config_gpios();
508
509     /* Initialize packet data arrays */
510     for (m = 0; m < DATA_USART_TX_BUFFER_SIZE; m++) {
511         for (n = 0; n < MAX_EEG_CHANNELS; n++) {
512             data_usart_tx_packet.eegdata[n][m][0] = 0;
513             data_usart_tx_packet.eegdata[n][m][1] = 0;
514             data_usart_tx_packet.eegdata[n][m][2] = 0;
515         }
516         for (n = 0; n < MAX_IMU_DOF; n++) {
517             data_usart_tx_packet.imudata[n][m] = 0;
518         }
519     }
520
521     /* Initialize SPI input buffer */
522     for (m = 0; m < DSP_PREBUFFER_NB_SAMPLES; m++) {
523         for (n = 0; n < MAX_EEG_CHANNELS; n++) {
524             spi_rx_data_buffer[m][n] = 0;
525         }
526     }
527
528     /* Set up peripheral modules other than TWI, which needs INTC first */
529     config_usart();
530     config_spi();
531
532     /* Verify ADS1299 device ID */
533     usart_write_line(AUX_USART, "Checking ADS1299 device ID...\r\n");
534     if (ads1299_check_device_id(SPI_ADS1299_MAIN_CHIPNUM) == STATUS_OK) {
535         usart_write_line(AUX_USART, "Device ID verified.\r\n");
536     }
537     else {
538         usart_write_line(AUX_USART, "Invalid ID. Possible SPI error.\r\n");
539         while(1) { cpu_relax(); }
540     }
541
542     /* Configure global interrupt controller */
543     Disable_global_interrupt();
544     INTC_init_interrupts();
545
546     #if (BRAINBOARD_REV > 0)
547     /* Init TWI */
548     config_twim();
549
550     /* Initialize the MPU6050 IMU */
551     usart_write_line(AUX_USART, "Initializing IMU...\r\n");
552     if (mpu6050_initialize_normal(MPU6050_DEFAULT_ADDRESS) == STATUS_OK) {
553         usart_write_line(AUX_USART, "IMU successfully initialized.\r\n");
554     }
555     else {
556         usart_write_line(AUX_USART, "IMU failed to initialize.\r\n");
557         while (1) { cpu_relax(); }
558     }
559     #endif
560
561     /* Configure DRDY interrupt */
562     config_eic();
563
564     /* Set up DMA for SPI/USART communications */
565     config_dma();
566
567     Enable_global_interrupt();
568
569     /* Enable receipt of USART commands */
570     pdca_enable(DATA_USART_RX_PDCA_CHANNEL);
571
```

```
572     /* Begin A/D conversion on the ADS1299 */
573     /* Comment out to allow host computer to initiate data collection */
574 //  ads1299_device_init(SPI_ADS1299_MAIN_CHIPNUM);
575 //  ads1299_send_byte(SPI_ADS1299_MAIN_CHIPNUM, ADS1299_OPC_RDATAC);
576 //  ads1299_soft_start_conversion(SPI_ADS1299_MAIN_CHIPNUM);
577
578     while(1)
579     {
580         if (frame_complete_flag)
581         {
582             Disable_global_interrupt();
583             /* DSP stuff on current frame */
584             for (n = 0; n < MAX_EEG_CHANNELS; n++)
585             {
586                 k = spi_buf_read_idx;
587                 for (m = 0; m < FRAME_SIZE; m++)
588                 {
589                     /* Casting to 16-bit signed data type (dsp16_t) will narrow the dynamic
590                      * range to +/- (2^15-1)*LSB, or +/- 17.58 mV.
591                      * It's possible we could increase the dynamic range by eliminating
592                      * some of the lower bits through right-shifting before the cast, but
593                      * this should be sufficient for well-conditioned EEG electrodes.
594                      */
595
596                     dsp_filter_input_buffer[m] = (dsp16_t) (spi_rx_data_buffer[k][n]);
597                     k = (k+1) % DSP_PREBUFFER_NB_SAMPLES;
598                 }
599
600                 /* Execute filter and/or other operations */
601
602             }
603             spi_buf_read_idx = k;
604
605         }
606         else
607         {
608             /* Go to IDLE and wait for DRDY interrupt */
609             AVR32_INTC.ipr[0];  // Dummy read in case any PB write operations are incomplete
610             SLEEP(AVR32_PM_SMODE_IDLE);
611         }
612     }
613 }
614
```

```
/**
 *  \file mpu60x0.c
 *  \brief Configuration settings, register definitions, and function prototypes for using the MPU-60X0 ↙
     inertial sensors from InvenSense.
 *  \author Graham Kelly
 *  \version 1.0
 *  \date August 2014
 *
 *  Header file mpu60x0.h borrows heavily from the MPU-6050 I2Cdev library by
 *  Jeff Rowberg, released under the MIT license.
 *
 *  All Atmel Software Framework libraries used herein are copyright Atmel and
 *  subject to their appropriate licenses, which allow free redistribution with
 *  some restrictions. These restrictions are listed in their appropriate files.
 *
 *  \page License
 *
 *  Brainboard firmware code is placed under the MIT license
 *  Copyright (c) 2014 Graham Kelly
 *
 *  Permission is hereby granted, free of charge, to any person obtaining a copy
 *  of this software and associated documentation files (the "Software"), to deal
 *  in the Software without restriction, including without limitation the rights
 *  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 *  copies of the Software, and to permit persons to whom the Software is
 *  furnished to do so, subject to the following conditions:
 *
 *  The above copyright notice and this permission notice shall be included in
 *  all copies or substantial portions of the Software.
 *
 *  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 *  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 *  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 *  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 *  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 *  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 *  THE SOFTWARE.
 *
 */

#include "mpu60x0.h"

uint8_t mpu6050_tx_buffer[1];
uint8_t mpu6050_rx_buffer[14];

status_code_t mpu6050_initialize_normal(uint32_t saddr)
{
    status_code_t status;
    twim_package_t init_package =
    {
        .chip           = MPU6050_DEFAULT_ADDRESS,
        .addr_length    = 1,
        .buffer         = (void *)mpu6050_tx_buffer,
        .length         = 1
    };

    status = STATUS_OK;

    /* Disable temperature sensor, source the MPU6050's clock from the X gyro oscillator, leave SLEEP * ↙
    /
    init_package.addr[0] = MPU6050_RA_PWR_MGMT_1;
    mpu6050_tx_buffer[0] = (    (1 << MPU6050_PWR1_TEMP_DIS_BIT)
                            |   (0 << MPU6050_PWR1_SLEEP_BIT)
                            |         MPU6050_CLOCK_PLL_XGYRO    );
    status = twim_write_packet(I2C_MODULE, (const twim_package_t *)&init_package);

```

```
65      /* Sanity check: read back that register */
66  //  twim_read_packet(I2C_MODULE, (const twim_package_t*)&verify_package);
67
68      /* Set desired sample rate */
69      init_package.addr[0] = MPU6050_RA_SMPLRT_DIV;
70      mpu6050_tx_buffer[0] = (uint8_t)((MPU6050_GYRO_OUTPUT_RATE_HZ/MPU6050_SAMPLING_RATE_HZ)-1);
71      status = twim_write_packet(I2C_MODULE, (const twim_package_t*)&init_package);
72
73      /* Configure LPF for 256 Hz bandwidth */
74      init_package.addr[0] = MPU6050_RA_CONFIG;
75      mpu6050_tx_buffer[0] = MPU6050_DLPF_BW;
76      status = twim_write_packet(I2C_MODULE, (const twim_package_t*)&init_package);
77
78      /* Gyro full-scale at +/-250 deg/s */
79      init_package.addr[0] = MPU6050_RA_GYRO_CONFIG;
80      mpu6050_tx_buffer[0] = (MPU6050_GYRO_FS_250 << MPU6050_GCONFIG_FS_SEL_BIT);
81      status = twim_write_packet(I2C_MODULE, (const twim_package_t *)&init_package);
82
83      /* Accelerometer full-scale at +/-2 g */
84      init_package.addr[0] = MPU6050_RA_ACCEL_CONFIG;
85      mpu6050_tx_buffer[0] = (MPU6050_ACCEL_FS_2 << MPU6050_ACONFIG_AFS_SEL_BIT);
86      status = twim_write_packet(I2C_MODULE, (const twim_package_t *)&init_package);
87
88      /* INT pin active high, push-pull, latched until any register read; FSYNC disabled */
89      init_package.addr[0] = MPU6050_RA_INT_PIN_CFG;
90      mpu6050_tx_buffer[0] = (  (MPU6050_INTMODE_ACTIVEHIGH << MPU6050_INTCFG_INT_LEVEL_BIT   )
91                              | (MPU6050_INTDRV_PUSHPULL    << MPU6050_INTCFG_INT_OPEN_BIT    )
92                              | (MPU6050_INTLATCH_WAITCLEAR << MPU6050_INTCFG_LATCH_INT_EN_BIT)
93                              | (MPU6050_INTCLEAR_ANYREAD   << MPU6050_INTCFG_INT_RD_CLEAR_BIT)  );
94      status = twim_write_packet(I2C_MODULE, (const twim_package_t *)&init_package);
95
96      /* Enable FIFO for X,Y,Z gyro and accelerometers, disable for temperature */
97      /* Data registers are arranged as ACCEL[X,Y,Z], TEMP, GYRO[X,Y,Z] with 2 bytes each (14 total).
98       * They are loaded into the FIFO in that order, from ACCEL_XOUT to GYRO_ZOUT.
99       */
100 //  init_package.addr[0] = MPU6050_RA_FIFO_EN;
101 //  mpu6050_tx_buffer[0] = (   (1 << MPU6050_XG_FIFO_EN_BIT)
102 //                           |  (1 << MPU6050_YG_FIFO_EN_BIT)
103 //                           |  (1 << MPU6050_ZG_FIFO_EN_BIT)
104 //                           |  (1 << MPU6050_ACCEL_FIFO_EN_BIT)
105 //                           |  (0 << MPU6050_TEMP_FIFO_EN_BIT)  );
106 //  status = twim_write_packet(I2C_MODULE, (const twim_package_t *)&init_package);
107
108     /* Enable FIFO operations */
109 //  init_package.addr[0] = MPU6050_RA_USER_CTRL;
110 //  mpu6050_tx_buffer[0] = 0x02;
111 //  status = twim_write_packet(I2C_MODULE, (const twim_package_t *)&init_package);
112 //  mpu6050_tx_buffer[0] = (1 << MPU6050_USERCTRL_FIFO_EN_BIT);
113 //  status = twim_write_packet(I2C_MODULE, (const twim_package_t *)&init_package);
114
115     /* Enable INT pin assertion upon entering DATA_RDY state */
116     init_package.addr[0] = MPU6050_RA_INT_ENABLE;
117     mpu6050_tx_buffer[0] = (1 << MPU6050_INTERRUPT_DATA_RDY_BIT);
118     status = twim_write_packet(I2C_MODULE, (const twim_package_t *)&init_package);
119
120     if (status != STATUS_OK) {
121         return ERR_IO_ERROR;
122     }
123     return STATUS_OK;
124 }
125
126
127
128 status_code_t mpu6050_initialize_lowpower(uint32_t saddr)
129 {
130     status_code_t status;
```

```
131     twim_package_t init_package =
132     {
133         .chip           = MPU6050_DEFAULT_ADDRESS,
134         .addr_length    = 1,
135         .buffer         = (void *)mpu6050_tx_buffer,
136         .length         = 1
137     };
138
139     status = STATUS_OK;
140
141     /* Disable temperature sensor, enter cycle mode:
142      * wakeup at fixed interval to read accelerometer only
143      */
144     /* CORRECTION: you may enable cycle mode, but it would be good to code a
145      * separate interrupt for the MPU-6050 in this case: tend to lose samples
146      * otherwise. No cycle mode for now.
147      */
148     init_package.addr[0] = MPU6050_RA_PWR_MGMT_1;
149     mpu6050_tx_buffer[0] = (    (1 << MPU6050_PWR1_TEMP_DIS_BIT)
150                             //| (1 << MPU6050_PWR1_CYCLE_BIT)
151                             |   (0 << MPU6050_PWR1_SLEEP_BIT));
152     status = twim_write_packet(I2C_MODULE, (const twim_package_t *)&init_package);
153
154     /* Put gyros in standby, set wakeup rate to 10 Hz */
155     init_package.addr[0] = MPU6050_RA_PWR_MGMT_2;
156     mpu6050_tx_buffer[0] = (    (MPU6050_WAKE_FREQ_10 <<  6)
157                             |   (1 << MPU6050_PWR2_STBY_XG_BIT)
158                             |   (1 << MPU6050_PWR2_STBY_YG_BIT)
159                             |   (1 << MPU6050_PWR2_STBY_ZG_BIT));
160     status = twim_write_packet(I2C_MODULE, (const twim_package_t *)&init_package);
161
162     /* Sanity check: read back that register */
163 //  twim_read_packet(I2C_MODULE, (const twim_package_t*)&verify_package);
164
165     /* Set desired sample rate */
166     init_package.addr[0] = MPU6050_RA_SMPLRT_DIV;
167     mpu6050_tx_buffer[0] = (uint8_t)((MPU6050_GYRO_OUTPUT_RATE_HZ/MPU6050_SAMPLING_RATE_HZ)-1);
168     status = twim_write_packet(I2C_MODULE, (const twim_package_t*)&init_package);
169
170     /* Configure LPF for 256 Hz bandwidth */
171     init_package.addr[0] = MPU6050_RA_CONFIG;
172     mpu6050_tx_buffer[0] = MPU6050_DLPF_BW;
173     status = twim_write_packet(I2C_MODULE, (const twim_package_t*)&init_package);
174
175     /* Accelerometer full-scale at +/-2 g */
176     init_package.addr[0] = MPU6050_RA_ACCEL_CONFIG;
177     mpu6050_tx_buffer[0] = (MPU6050_ACCEL_FS_2 << MPU6050_ACONFIG_AFS_SEL_BIT);
178     status = twim_write_packet(I2C_MODULE, (const twim_package_t *)&init_package);
179
180     /* INT pin active high, push-pull, latched until any register read; FSYNC disabled */
181     init_package.addr[0] = MPU6050_RA_INT_PIN_CFG;
182     mpu6050_tx_buffer[0] = (  (MPU6050_INTMODE_ACTIVEHIGH << MPU6050_INTCFG_INT_LEVEL_BIT   )
183                             | (MPU6050_INTDRV_PUSHPULL    << MPU6050_INTCFG_INT_OPEN_BIT    )
184                             | (MPU6050_INTLATCH_WAITCLEAR << MPU6050_INTCFG_LATCH_INT_EN_BIT)
185                             | (MPU6050_INTCLEAR_ANYREAD   << MPU6050_INTCFG_INT_RD_CLEAR_BIT)  );
186     status = twim_write_packet(I2C_MODULE, (const twim_package_t *)&init_package);
187
188     /* Enable FIFO for X,Y,Z gyro and accelerometers, disable for temperature */
189     /* Data registers are arranged as ACCEL[X,Y,Z], TEMP, GYRO[X,Y,Z] with 2 bytes each (14 total).
190      * They are loaded into the FIFO in that order, from ACCEL_XOUT to GYRO_ZOUT.
191      */
192 //  init_package.addr[0] = MPU6050_RA_FIFO_EN;
193 //  mpu6050_tx_buffer[0] = (   (1 << MPU6050_XG_FIFO_EN_BIT)
194 //                          |  (1 << MPU6050_YG_FIFO_EN_BIT)
195 //                          |  (1 << MPU6050_ZG_FIFO_EN_BIT)
```

```
196 //                          |  (1 << MPU6050_ACCEL_FIFO_EN_BIT)
197 //                          |  (0 << MPU6050_TEMP_FIFO_EN_BIT)  );
198 //  status = twim_write_packet(I2C_MODULE, (const twim_package_t *)&init_package);
199
200     /* Enable FIFO operations */
201 //  init_package.addr[0] = MPU6050_RA_USER_CTRL;
202 //  mpu6050_tx_buffer[0] = 0x02;
203 //  status = twim_write_packet(I2C_MODULE, (const twim_package_t *)&init_package);
204 //  mpu6050_tx_buffer[0] = (1 << MPU6050_USERCTRL_FIFO_EN_BIT);
205 //  status = twim_write_packet(I2C_MODULE, (const twim_package_t *)&init_package);
206
207     /* Enable INT pin assertion upon entering DATA_RDY state */
208     init_package.addr[0] = MPU6050_RA_INT_ENABLE;
209     mpu6050_tx_buffer[0] = (1 << MPU6050_INTERRUPT_DATA_RDY_BIT);
210     status = twim_write_packet(I2C_MODULE, (const twim_package_t *)&init_package);
211
212     if (status != STATUS_OK) {
213         return ERR_IO_ERROR;
214     }
215     return STATUS_OK;
216 }
217
218 status_code_t mpu6050_rdata16_packet(uint32_t saddr, volatile uint32_t sample_idx, data_packet_t*
    packet)
219 {
220     status_code_t status;
221     int16_t data[7];
222     const twim_package_t twim_package =
223     {
224         .chip           = saddr,
225         .addr[0]        = MPU6050_RA_ACCEL_XOUT_H,
226         .addr_length    = 1,
227         .buffer         = (void *)data,
228         .length         = 14
229     };
230
231     status = twim_read_packet(I2C_MODULE, &twim_package);
232     /* Accelerometer data */
233     packet->imudata[sample_idx][0] = data[0];
234     packet->imudata[sample_idx][1] = data[1];
235     packet->imudata[sample_idx][2] = data[2];
236     /* Skip temperature data */
237     /* Gyro data */
238     packet->imudata[sample_idx][3] = data[4];
239     packet->imudata[sample_idx][4] = data[5];
240     packet->imudata[sample_idx][5] = data[6];
241
242     return status;
243 }
244
245 status_code_t mpu6050_rdata16_fifo(uint32_t saddr, volatile uint32_t sample_idx, data_packet_t* packet)
246 {
247     status_code_t status;
248     uint8_t fifo_count_lo, fifo_count_hi;
249     uint16_t fifo_count;
250     twim_package_t twim_package =
251     {
252         .chip           = saddr,
253         .addr[0]        = MPU6050_RA_FIFO_COUNTH,
254         .addr_length    = 1,
255         .buffer         = (void *)&fifo_count_hi,
256         .length         = 1
257     };
258
259     fifo_count_hi = 0;
260     fifo_count_lo = 0;
```

```
261
262     /* First check how many samples are in the FIFO */
263     /* TWIM packet was initialized to read the FIFO_COUNT high byte first, so read now */
264     status = twim_read_packet(I2C_MODULE, (const twim_package_t *)&twim_package);
265     /* Now read the low byte */
266     twim_package.buffer = (void *)&fifo_count_lo;
267     status = twim_read_packet(I2C_MODULE, (const twim_package_t *)&twim_package);
268     /* Concatenate */
269     fifo_count = fifo_count_lo | ((uint16_t)fifo_count_hi << 8);
270     /* Read the FIFO if there's enough data */
271     if (fifo_count >= 2*MAX_IMU_DOF) {
272         twim_package.addr[0] = MPU6050_RA_FIFO_R_W;
273         twim_package.buffer = (void *)packet->imudata[sample_idx];
274         twim_package.length = 2*MAX_IMU_DOF;
275         status = twim_read_packet(I2C_MODULE, (const twim_package_t *)&twim_package);
276     }
277     else return ERR_BUSY;
278     return status;
279 }
280
```

```
/**
 *  \file mpu60x0.h
 *
 *  \brief Useful #defines from MPU6050 inertial measurement I2C device class from I2Cdevlib
 *
 *  All Atmel Software Framework libraries used herein are copyright Atmel and
 *  subject to their appropriate licenses, which allow free redistribution with
 *  some restrictions. These restrictions are listed in their appropriate files.
 *
 *  Original code 10/3/2011 by Jeff Rowberg <jeff@rowberg.net>,
 *  modified 3/4/2014 by Graham Kelly <kellygs@vcu.edu>
 *
 *  \page License
 *
 *  I2Cdev device library code is placed under the MIT license
 *  Copyright (c) 2012 Jeff Rowberg
 *
 *  Permission is hereby granted, free of charge, to any person obtaining a copy
 *  of this software and associated documentation files (the "Software"), to deal
 *  in the Software without restriction, including without limitation the rights
 *  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 *  copies of the Software, and to permit persons to whom the Software is
 *  furnished to do so, subject to the following conditions:
 *
 *  The above copyright notice and this permission notice shall be included in
 *  all copies or substantial portions of the Software.
 *
 *  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 *  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 *  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 *  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 *  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 *  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 *  THE SOFTWARE.
 *
 */

#ifndef _MPU6050_H_
#define _MPU6050_H_

//#include "I2Cdev.h"
//#include <avr/pgmspace.h>
#include "brainboard.h"
#include "comms.h"
#include <twim.h>

#ifdef __cplusplus
extern "C" {
#endif

/*****************************************************************************************************
 *              Data Packet Definitions                                                              *
 *****************************************************************************************************/

#define I2C_MASTER_SPEED_HZ         400000  // 400 kHz

#define MPU6050_ADDRESS_AD0_LOW     0x68 // address pin low (GND), default for InvenSense evaluation board
#define MPU6050_ADDRESS_AD0_HIGH    0x69 // address pin high (VCC)
#define MPU6050_DEFAULT_ADDRESS     MPU6050_ADDRESS_AD0_LOW

/**
 * \brief Desired sampling rate.
```

```
 *
 *  Determines SMPLRT_DIV register value:
 *  SMPLRT_DIV = (Gyroscope output rate)/(Desired sampling rate) - 1
 */
#define MPU6050_SAMPLING_RATE_HZ    10

/**
 * \brief Digital low-pass filter setting for MPU-6050.
 *
 */
#define MPU6050_DLPF_BW     MPU6050_DLPF_BW_256

/* Gyro updates at full speed only if DLPF is off (i.e. BW = 256 Hz). */
#if (MPU6050_DLPF_BW == MPU6050_DLPF_BW_256)
#define MPU6050_GYRO_OUTPUT_RATE_HZ 8000
#else
#define MPU6050_GYRO_OUTPUT_RATE_HZ 1000
#endif

/**
 *  \brief MPU6050 register addresses.
 *
 * Consult the MPU6050 register map document for more information.
 */
#define MPU6050_RA_XG_OFFS_TC       0x00 //[7] PWR_MODE, [6:1] XG_OFFS_TC, [0] OTP_BNK_VLD
#define MPU6050_RA_YG_OFFS_TC       0x01 //[7] PWR_MODE, [6:1] YG_OFFS_TC, [0] OTP_BNK_VLD
#define MPU6050_RA_ZG_OFFS_TC       0x02 //[7] PWR_MODE, [6:1] ZG_OFFS_TC, [0] OTP_BNK_VLD
#define MPU6050_RA_X_FINE_GAIN      0x03 //[7:0] X_FINE_GAIN
#define MPU6050_RA_Y_FINE_GAIN      0x04 //[7:0] Y_FINE_GAIN
#define MPU6050_RA_Z_FINE_GAIN      0x05 //[7:0] Z_FINE_GAIN
#define MPU6050_RA_XA_OFFS_H        0x06 //[15:0] XA_OFFS
#define MPU6050_RA_XA_OFFS_L_TC     0x07
#define MPU6050_RA_YA_OFFS_H        0x08 //[15:0] YA_OFFS
#define MPU6050_RA_YA_OFFS_L_TC     0x09
#define MPU6050_RA_ZA_OFFS_H        0x0A //[15:0] ZA_OFFS
#define MPU6050_RA_ZA_OFFS_L_TC     0x0B
#define MPU6050_RA_XG_OFFS_USRH     0x13 //[15:0] XG_OFFS_USR
#define MPU6050_RA_XG_OFFS_USRL     0x14
#define MPU6050_RA_YG_OFFS_USRH     0x15 //[15:0] YG_OFFS_USR
#define MPU6050_RA_YG_OFFS_USRL     0x16
#define MPU6050_RA_ZG_OFFS_USRH     0x17 //[15:0] ZG_OFFS_USR
#define MPU6050_RA_ZG_OFFS_USRL     0x18
#define MPU6050_RA_SMPLRT_DIV       0x19
#define MPU6050_RA_CONFIG           0x1A
#define MPU6050_RA_GYRO_CONFIG      0x1B
#define MPU6050_RA_ACCEL_CONFIG     0x1C
#define MPU6050_RA_FF_THR           0x1D
#define MPU6050_RA_FF_DUR           0x1E
#define MPU6050_RA_MOT_THR          0x1F
#define MPU6050_RA_MOT_DUR          0x20
#define MPU6050_RA_ZRMOT_THR        0x21
#define MPU6050_RA_ZRMOT_DUR        0x22
#define MPU6050_RA_FIFO_EN          0x23
#define MPU6050_RA_I2C_MST_CTRL     0x24
#define MPU6050_RA_I2C_SLV0_ADDR    0x25
#define MPU6050_RA_I2C_SLV0_REG     0x26
#define MPU6050_RA_I2C_SLV0_CTRL    0x27
#define MPU6050_RA_I2C_SLV1_ADDR    0x28
#define MPU6050_RA_I2C_SLV1_REG     0x29
#define MPU6050_RA_I2C_SLV1_CTRL    0x2A
#define MPU6050_RA_I2C_SLV2_ADDR    0x2B
#define MPU6050_RA_I2C_SLV2_REG     0x2C
#define MPU6050_RA_I2C_SLV2_CTRL    0x2D
#define MPU6050_RA_I2C_SLV3_ADDR    0x2E
#define MPU6050_RA_I2C_SLV3_REG     0x2F
#define MPU6050_RA_I2C_SLV3_CTRL    0x30
```

```
129 #define MPU6050_RA_I2C_SLV4_ADDR    0x31
130 #define MPU6050_RA_I2C_SLV4_REG     0x32
131 #define MPU6050_RA_I2C_SLV4_DO      0x33
132 #define MPU6050_RA_I2C_SLV4_CTRL    0x34
133 #define MPU6050_RA_I2C_SLV4_DI      0x35
134 #define MPU6050_RA_I2C_MST_STATUS   0x36
135 #define MPU6050_RA_INT_PIN_CFG      0x37
136 #define MPU6050_RA_INT_ENABLE       0x38
137 #define MPU6050_RA_DMP_INT_STATUS   0x39
138 #define MPU6050_RA_INT_STATUS       0x3A
139 #define MPU6050_RA_ACCEL_XOUT_H     0x3B
140 #define MPU6050_RA_ACCEL_XOUT_L     0x3C
141 #define MPU6050_RA_ACCEL_YOUT_H     0x3D
142 #define MPU6050_RA_ACCEL_YOUT_L     0x3E
143 #define MPU6050_RA_ACCEL_ZOUT_H     0x3F
144 #define MPU6050_RA_ACCEL_ZOUT_L     0x40
145 #define MPU6050_RA_TEMP_OUT_H       0x41
146 #define MPU6050_RA_TEMP_OUT_L       0x42
147 #define MPU6050_RA_GYRO_XOUT_H      0x43
148 #define MPU6050_RA_GYRO_XOUT_L      0x44
149 #define MPU6050_RA_GYRO_YOUT_H      0x45
150 #define MPU6050_RA_GYRO_YOUT_L      0x46
151 #define MPU6050_RA_GYRO_ZOUT_H      0x47
152 #define MPU6050_RA_GYRO_ZOUT_L      0x48
153 #define MPU6050_RA_EXT_SENS_DATA_00 0x49
154 #define MPU6050_RA_EXT_SENS_DATA_01 0x4A
155 #define MPU6050_RA_EXT_SENS_DATA_02 0x4B
156 #define MPU6050_RA_EXT_SENS_DATA_03 0x4C
157 #define MPU6050_RA_EXT_SENS_DATA_04 0x4D
158 #define MPU6050_RA_EXT_SENS_DATA_05 0x4E
159 #define MPU6050_RA_EXT_SENS_DATA_06 0x4F
160 #define MPU6050_RA_EXT_SENS_DATA_07 0x50
161 #define MPU6050_RA_EXT_SENS_DATA_08 0x51
162 #define MPU6050_RA_EXT_SENS_DATA_09 0x52
163 #define MPU6050_RA_EXT_SENS_DATA_10 0x53
164 #define MPU6050_RA_EXT_SENS_DATA_11 0x54
165 #define MPU6050_RA_EXT_SENS_DATA_12 0x55
166 #define MPU6050_RA_EXT_SENS_DATA_13 0x56
167 #define MPU6050_RA_EXT_SENS_DATA_14 0x57
168 #define MPU6050_RA_EXT_SENS_DATA_15 0x58
169 #define MPU6050_RA_EXT_SENS_DATA_16 0x59
170 #define MPU6050_RA_EXT_SENS_DATA_17 0x5A
171 #define MPU6050_RA_EXT_SENS_DATA_18 0x5B
172 #define MPU6050_RA_EXT_SENS_DATA_19 0x5C
173 #define MPU6050_RA_EXT_SENS_DATA_20 0x5D
174 #define MPU6050_RA_EXT_SENS_DATA_21 0x5E
175 #define MPU6050_RA_EXT_SENS_DATA_22 0x5F
176 #define MPU6050_RA_EXT_SENS_DATA_23 0x60
177 #define MPU6050_RA_MOT_DETECT_STATUS    0x61
178 #define MPU6050_RA_I2C_SLV0_DO      0x63
179 #define MPU6050_RA_I2C_SLV1_DO      0x64
180 #define MPU6050_RA_I2C_SLV2_DO      0x65
181 #define MPU6050_RA_I2C_SLV3_DO      0x66
182 #define MPU6050_RA_I2C_MST_DELAY_CTRL   0x67
183 #define MPU6050_RA_SIGNAL_PATH_RESET    0x68
184 #define MPU6050_RA_MOT_DETECT_CTRL      0x69
185 #define MPU6050_RA_USER_CTRL        0x6A
186 #define MPU6050_RA_PWR_MGMT_1       0x6B
187 #define MPU6050_RA_PWR_MGMT_2       0x6C
188 #define MPU6050_RA_BANK_SEL         0x6D
189 #define MPU6050_RA_MEM_START_ADDR   0x6E
190 #define MPU6050_RA_MEM_R_W          0x6F
191 #define MPU6050_RA_DMP_CFG_1        0x70
192 #define MPU6050_RA_DMP_CFG_2        0x71
193 #define MPU6050_RA_FIFO_COUNTH      0x72
194 #define MPU6050_RA_FIFO_COUNTL      0x73
```

```
195 #define MPU6050_RA_FIFO_R_W         0x74
196 #define MPU6050_RA_WHO_AM_I         0x75
197
198 #define MPU6050_TC_PWR_MODE_BIT     7
199 #define MPU6050_TC_OFFSET_BIT       6
200 #define MPU6050_TC_OFFSET_LENGTH    6
201 #define MPU6050_TC_OTP_BNK_VLD_BIT  0
202
203 #define MPU6050_VDDIO_LEVEL_VLOGIC  0
204 #define MPU6050_VDDIO_LEVEL_VDD     1
205
206 #define MPU6050_CFG_EXT_SYNC_SET_BIT    5
207 #define MPU6050_CFG_EXT_SYNC_SET_LENGTH 3
208 #define MPU6050_CFG_DLPF_CFG_BIT    2
209 #define MPU6050_CFG_DLPF_CFG_LENGTH 3
210
211 #define MPU6050_EXT_SYNC_DISABLED       0x0
212 #define MPU6050_EXT_SYNC_TEMP_OUT_L     0x1
213 #define MPU6050_EXT_SYNC_GYRO_XOUT_L    0x2
214 #define MPU6050_EXT_SYNC_GYRO_YOUT_L    0x3
215 #define MPU6050_EXT_SYNC_GYRO_ZOUT_L    0x4
216 #define MPU6050_EXT_SYNC_ACCEL_XOUT_L   0x5
217 #define MPU6050_EXT_SYNC_ACCEL_YOUT_L   0x6
218 #define MPU6050_EXT_SYNC_ACCEL_ZOUT_L   0x7
219
220 #define MPU6050_DLPF_BW_256         0x00
221 #define MPU6050_DLPF_BW_188         0x01
222 #define MPU6050_DLPF_BW_98          0x02
223 #define MPU6050_DLPF_BW_42          0x03
224 #define MPU6050_DLPF_BW_20          0x04
225 #define MPU6050_DLPF_BW_10          0x05
226 #define MPU6050_DLPF_BW_5           0x06
227
228 #define MPU6050_GCONFIG_FS_SEL_BIT      4
229 #define MPU6050_GCONFIG_FS_SEL_LENGTH   2
230
231 #define MPU6050_GYRO_FS_250         0x00
232 #define MPU6050_GYRO_FS_500         0x01
233 #define MPU6050_GYRO_FS_1000        0x02
234 #define MPU6050_GYRO_FS_2000        0x03
235
236 #define MPU6050_ACONFIG_XA_ST_BIT           7
237 #define MPU6050_ACONFIG_YA_ST_BIT           6
238 #define MPU6050_ACONFIG_ZA_ST_BIT           5
239 #define MPU6050_ACONFIG_AFS_SEL_BIT         4
240 #define MPU6050_ACONFIG_AFS_SEL_LENGTH      2
241 #define MPU6050_ACONFIG_ACCEL_HPF_BIT       2
242 #define MPU6050_ACONFIG_ACCEL_HPF_LENGTH    3
243
244 #define MPU6050_ACCEL_FS_2          0x00
245 #define MPU6050_ACCEL_FS_4          0x01
246 #define MPU6050_ACCEL_FS_8          0x02
247 #define MPU6050_ACCEL_FS_16         0x03
248
249 #define MPU6050_DHPF_RESET          0x00
250 #define MPU6050_DHPF_5              0x01
251 #define MPU6050_DHPF_2P5            0x02
252 #define MPU6050_DHPF_1P25           0x03
253 #define MPU6050_DHPF_0P63           0x04
254 #define MPU6050_DHPF_HOLD           0x07
255
256 #define MPU6050_TEMP_FIFO_EN_BIT    7
257 #define MPU6050_XG_FIFO_EN_BIT      6
258 #define MPU6050_YG_FIFO_EN_BIT      5
259 #define MPU6050_ZG_FIFO_EN_BIT      4
260 #define MPU6050_ACCEL_FIFO_EN_BIT   3
```

```
261 #define MPU6050_SLV2_FIFO_EN_BIT    2
262 #define MPU6050_SLV1_FIFO_EN_BIT    1
263 #define MPU6050_SLV0_FIFO_EN_BIT    0
264
265 #define MPU6050_MULT_MST_EN_BIT     7
266 #define MPU6050_WAIT_FOR_ES_BIT     6
267 #define MPU6050_SLV_3_FIFO_EN_BIT   5
268 #define MPU6050_I2C_MST_P_NSR_BIT   4
269 #define MPU6050_I2C_MST_CLK_BIT     3
270 #define MPU6050_I2C_MST_CLK_LENGTH  4
271
272 #define MPU6050_CLOCK_DIV_348       0x0
273 #define MPU6050_CLOCK_DIV_333       0x1
274 #define MPU6050_CLOCK_DIV_320       0x2
275 #define MPU6050_CLOCK_DIV_308       0x3
276 #define MPU6050_CLOCK_DIV_296       0x4
277 #define MPU6050_CLOCK_DIV_286       0x5
278 #define MPU6050_CLOCK_DIV_276       0x6
279 #define MPU6050_CLOCK_DIV_267       0x7
280 #define MPU6050_CLOCK_DIV_258       0x8
281 #define MPU6050_CLOCK_DIV_500       0x9
282 #define MPU6050_CLOCK_DIV_471       0xA
283 #define MPU6050_CLOCK_DIV_444       0xB
284 #define MPU6050_CLOCK_DIV_421       0xC
285 #define MPU6050_CLOCK_DIV_400       0xD
286 #define MPU6050_CLOCK_DIV_381       0xE
287 #define MPU6050_CLOCK_DIV_364       0xF
288
289 #define MPU6050_I2C_SLV_RW_BIT      7
290 #define MPU6050_I2C_SLV_ADDR_BIT    6
291 #define MPU6050_I2C_SLV_ADDR_LENGTH 7
292 #define MPU6050_I2C_SLV_EN_BIT      7
293 #define MPU6050_I2C_SLV_BYTE_SW_BIT 6
294 #define MPU6050_I2C_SLV_REG_DIS_BIT 5
295 #define MPU6050_I2C_SLV_GRP_BIT     4
296 #define MPU6050_I2C_SLV_LEN_BIT     3
297 #define MPU6050_I2C_SLV_LEN_LENGTH  4
298
299 #define MPU6050_I2C_SLV4_RW_BIT         7
300 #define MPU6050_I2C_SLV4_ADDR_BIT       6
301 #define MPU6050_I2C_SLV4_ADDR_LENGTH    7
302 #define MPU6050_I2C_SLV4_EN_BIT         7
303 #define MPU6050_I2C_SLV4_INT_EN_BIT     6
304 #define MPU6050_I2C_SLV4_REG_DIS_BIT    5
305 #define MPU6050_I2C_SLV4_MST_DLY_BIT    4
306 #define MPU6050_I2C_SLV4_MST_DLY_LENGTH 5
307
308 #define MPU6050_MST_PASS_THROUGH_BIT    7
309 #define MPU6050_MST_I2C_SLV4_DONE_BIT   6
310 #define MPU6050_MST_I2C_LOST_ARB_BIT    5
311 #define MPU6050_MST_I2C_SLV4_NACK_BIT   4
312 #define MPU6050_MST_I2C_SLV3_NACK_BIT   3
313 #define MPU6050_MST_I2C_SLV2_NACK_BIT   2
314 #define MPU6050_MST_I2C_SLV1_NACK_BIT   1
315 #define MPU6050_MST_I2C_SLV0_NACK_BIT   0
316
317 #define MPU6050_INTCFG_INT_LEVEL_BIT        7
318 #define MPU6050_INTCFG_INT_OPEN_BIT         6
319 #define MPU6050_INTCFG_LATCH_INT_EN_BIT     5
320 #define MPU6050_INTCFG_INT_RD_CLEAR_BIT     4
321 #define MPU6050_INTCFG_FSYNC_INT_LEVEL_BIT  3
322 #define MPU6050_INTCFG_FSYNC_INT_EN_BIT     2
323 #define MPU6050_INTCFG_I2C_BYPASS_EN_BIT    1
324 #define MPU6050_INTCFG_CLKOUT_EN_BIT        0
325
326 #define MPU6050_INTMODE_ACTIVEHIGH  0x00
```

```
327 #define MPU6050_INTMODE_ACTIVELOW   0x01
328
329 #define MPU6050_INTDRV_PUSHPULL     0x00
330 #define MPU6050_INTDRV_OPENDRAIN    0x01
331
332 #define MPU6050_INTLATCH_50USPULSE  0x00
333 #define MPU6050_INTLATCH_WAITCLEAR  0x01
334
335 #define MPU6050_INTCLEAR_STATUSREAD 0x00
336 #define MPU6050_INTCLEAR_ANYREAD    0x01
337
338 #define MPU6050_INTERRUPT_FF_BIT            7
339 #define MPU6050_INTERRUPT_MOT_BIT           6
340 #define MPU6050_INTERRUPT_ZMOT_BIT          5
341 #define MPU6050_INTERRUPT_FIFO_OFLOW_BIT    4
342 #define MPU6050_INTERRUPT_I2C_MST_INT_BIT   3
343 #define MPU6050_INTERRUPT_PLL_RDY_INT_BIT   2
344 #define MPU6050_INTERRUPT_DMP_INT_BIT       1
345 #define MPU6050_INTERRUPT_DATA_RDY_BIT      0
346
347 // TODO: figure out what these actually do
348 // UMPL source code is not very obivous
349 #define MPU6050_DMPINT_5_BIT            5
350 #define MPU6050_DMPINT_4_BIT            4
351 #define MPU6050_DMPINT_3_BIT            3
352 #define MPU6050_DMPINT_2_BIT            2
353 #define MPU6050_DMPINT_1_BIT            1
354 #define MPU6050_DMPINT_0_BIT            0
355
356 #define MPU6050_MOTION_MOT_XNEG_BIT     7
357 #define MPU6050_MOTION_MOT_XPOS_BIT     6
358 #define MPU6050_MOTION_MOT_YNEG_BIT     5
359 #define MPU6050_MOTION_MOT_YPOS_BIT     4
360 #define MPU6050_MOTION_MOT_ZNEG_BIT     3
361 #define MPU6050_MOTION_MOT_ZPOS_BIT     2
362 #define MPU6050_MOTION_MOT_ZRMOT_BIT    0
363
364 #define MPU6050_DELAYCTRL_DELAY_ES_SHADOW_BIT   7
365 #define MPU6050_DELAYCTRL_I2C_SLV4_DLY_EN_BIT   4
366 #define MPU6050_DELAYCTRL_I2C_SLV3_DLY_EN_BIT   3
367 #define MPU6050_DELAYCTRL_I2C_SLV2_DLY_EN_BIT   2
368 #define MPU6050_DELAYCTRL_I2C_SLV1_DLY_EN_BIT   1
369 #define MPU6050_DELAYCTRL_I2C_SLV0_DLY_EN_BIT   0
370
371 #define MPU6050_PATHRESET_GYRO_RESET_BIT    2
372 #define MPU6050_PATHRESET_ACCEL_RESET_BIT   1
373 #define MPU6050_PATHRESET_TEMP_RESET_BIT    0
374
375 #define MPU6050_DETECT_ACCEL_ON_DELAY_BIT       5
376 #define MPU6050_DETECT_ACCEL_ON_DELAY_LENGTH    2
377 #define MPU6050_DETECT_FF_COUNT_BIT             3
378 #define MPU6050_DETECT_FF_COUNT_LENGTH          2
379 #define MPU6050_DETECT_MOT_COUNT_BIT            1
380 #define MPU6050_DETECT_MOT_COUNT_LENGTH         2
381
382 #define MPU6050_DETECT_DECREMENT_RESET  0x0
383 #define MPU6050_DETECT_DECREMENT_1      0x1
384 #define MPU6050_DETECT_DECREMENT_2      0x2
385 #define MPU6050_DETECT_DECREMENT_4      0x3
386
387 #define MPU6050_USERCTRL_DMP_EN_BIT             7
388 #define MPU6050_USERCTRL_FIFO_EN_BIT            6
389 #define MPU6050_USERCTRL_I2C_MST_EN_BIT         5
390 #define MPU6050_USERCTRL_I2C_IF_DIS_BIT         4
391 #define MPU6050_USERCTRL_DMP_RESET_BIT          3
392 #define MPU6050_USERCTRL_FIFO_RESET_BIT         2
```

```
393 #define MPU6050_USERCTRL_I2C_MST_RESET_BIT      1
394 #define MPU6050_USERCTRL_SIG_COND_RESET_BIT     0
395
396 #define MPU6050_PWR1_DEVICE_RESET_BIT   7
397 #define MPU6050_PWR1_SLEEP_BIT          6
398 #define MPU6050_PWR1_CYCLE_BIT          5
399 #define MPU6050_PWR1_TEMP_DIS_BIT       3
400 #define MPU6050_PWR1_CLKSEL_BIT         2
401 #define MPU6050_PWR1_CLKSEL_LENGTH      3
402
403 #define MPU6050_CLOCK_INTERNAL          0x00
404 #define MPU6050_CLOCK_PLL_XGYRO         0x01
405 #define MPU6050_CLOCK_PLL_YGYRO         0x02
406 #define MPU6050_CLOCK_PLL_ZGYRO         0x03
407 #define MPU6050_CLOCK_PLL_EXT32K        0x04
408 #define MPU6050_CLOCK_PLL_EXT19M        0x05
409 #define MPU6050_CLOCK_KEEP_RESET        0x07
410
411 #define MPU6050_PWR2_LP_WAKE_CTRL_BIT       7
412 #define MPU6050_PWR2_LP_WAKE_CTRL_LENGTH    2
413 #define MPU6050_PWR2_STBY_XA_BIT            5
414 #define MPU6050_PWR2_STBY_YA_BIT            4
415 #define MPU6050_PWR2_STBY_ZA_BIT            3
416 #define MPU6050_PWR2_STBY_XG_BIT            2
417 #define MPU6050_PWR2_STBY_YG_BIT            1
418 #define MPU6050_PWR2_STBY_ZG_BIT            0
419
420 #define MPU6050_WAKE_FREQ_1P25      0x0
421 #define MPU6050_WAKE_FREQ_2P5       0x1
422 #define MPU6050_WAKE_FREQ_5         0x2
423 #define MPU6050_WAKE_FREQ_10        0x3
424
425 #define MPU6050_BANKSEL_PRFTCH_EN_BIT       6
426 #define MPU6050_BANKSEL_CFG_USER_BANK_BIT   5
427 #define MPU6050_BANKSEL_MEM_SEL_BIT         4
428 #define MPU6050_BANKSEL_MEM_SEL_LENGTH      5
429
430 #define MPU6050_WHO_AM_I_BIT        6
431 #define MPU6050_WHO_AM_I_LENGTH     6
432
433 #define MPU6050_DMP_MEMORY_BANKS        8
434 #define MPU6050_DMP_MEMORY_BANK_SIZE    256
435 #define MPU6050_DMP_MEMORY_CHUNK_SIZE   16
436
437 status_code_t mpu6050_initialize_normal(uint32_t);
438 status_code_t mpu6050_initialize_lowpower(uint32_t);
439 status_code_t mpu6050_rdata16_packet(uint32_t, volatile uint32_t, data_packet_t*);
440 status_code_t mpu6050_rdata16_fifo(uint32_t, volatile uint32_t, data_packet_t*);
441
442 #ifdef __cplusplus
443 }
444 #endif
445
446 #endif /* #ifndef _MPU6050_H_ */
```

```
/**
 *  \file ads1299.c
 *  \brief Some useful functions for using the ADS1299 EEG analog front-end from Texas Instruments.
 *  \details The ADS1299 is an SPI slave device. Communication with it requires at least one SPI master module on the host.
 *  \author Graham Kelly
 *  \version 1.0
 *  \date August 2014
 *
 *  All Atmel Software Framework libraries used herein are copyright Atmel and
 *  subject to their appropriate licenses, which allow free redistribution with
 *  some restrictions. These restrictions are listed in their appropriate files.
 *
 *  \page License
 *
 *  Brainboard firmware code is placed under the MIT license
 *  Copyright (c) 2014 Graham Kelly
 *
 *  Permission is hereby granted, free of charge, to any person obtaining a copy
 *  of this software and associated documentation files (the "Software"), to deal
 *  in the Software without restriction, including without limitation the rights
 *  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 *  copies of the Software, and to permit persons to whom the Software is
 *  furnished to do so, subject to the following conditions:
 *
 *  The above copyright notice and this permission notice shall be included in
 *  all copies or substantial portions of the Software.
 *
 *  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 *  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 *  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 *  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 *  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 *  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 *  THE SOFTWARE.
 *
 */

#ifdef __cplusplus
extern "C" {
#endif

#include "ads1299.h"

/****************************************************************************************************
 *              Function Definitions                                                                *
 ****************************************************************************************************/

/* SYSTEM CONTROL FUNCTIONS ************************************************************************/

ads1299_error_t ads1299_device_init(uint8_t chip_select)
{
    #if UC3
    /* Power cycle ADS1299 */
    gpio_clr_gpio_pin(ADS1299_PIN_PWDN);
    delay_us(20);
    gpio_set_gpio_pin(ADS1299_PIN_PWDN);

    /* Allow oscillator warm-up */
    delay_ms(1000);

    /* Tell chip to exit continuous data mode */
```

```
62      ads1299_send_byte(chip_select, ADS1299_OPC_SDATAC);
63      /* Stop taking conversions; apparently not done automatically */
64      ads1299_send_byte(chip_select, ADS1299_OPC_STOP);
65
66      /* Write to GPIO register, set all pins to driven-low output */
67      ads1299_wreg(chip_select, ADS1299_REGADDR_GPIO, ADS1299_REG_GPIO_GPIOC4_OUTPUT |
68                                                      ADS1299_REG_GPIO_GPIOD4_LOW    |
69                                                      ADS1299_REG_GPIO_GPIOC3_OUTPUT |
70                                                      ADS1299_REG_GPIO_GPIOD3_LOW    |
71                                                      ADS1299_REG_GPIO_GPIOC2_OUTPUT |
72                                                      ADS1299_REG_GPIO_GPIOD2_LOW    |
73                                                      ADS1299_REG_GPIO_GPIOC1_OUTPUT |
74                                                      ADS1299_REG_GPIO_GPIOD1_LOW    );
75
76      /* Write to CONFIG1, set data rate to 250 Hz */
77      ads1299_wreg(chip_select, ADS1299_REGADDR_CONFIG1, ADS1299_REG_CONFIG1_RESERVED_VALUE |
78                                                         ADS1299_REG_CONFIG1_FMOD_DIV_BY_4096);
79      /* Write to CONFIG2 register, generate test signal internally */
80      ads1299_wreg(chip_select, ADS1299_REGADDR_CONFIG2, ADS1299_REG_CONFIG2_RESERVED_VALUE |
81                                                         ADS1299_REG_CONFIG2_CAL_INT        |
82                                                         ADS1299_REG_CONFIG2_CAL_PULSE_FCLK_DIV_2_21);
83
84      /* Write to CONFIG3, enable internal reference buffer, bias internally generated, bias buffer ↙
        enabled */
85      ads1299_wreg(chip_select, ADS1299_REGADDR_CONFIG3, ADS1299_REG_CONFIG3_REFBUF_ENABLED |
86                                                         ADS1299_REG_CONFIG3_BIASREF_INT    |
87                                                         ADS1299_REG_CONFIG3_BIASBUF_ENABLED);
88      /* Reference settling time */
89      delay_ms(150);
90
91      /* Write to CH1 settings register, set as normal input, gain 24 */
92      ads1299_wreg(chip_select, ADS1299_REGADDR_CH1SET, ADS1299_REG_CHNSET_CHANNEL_ON         |
93                                                        ADS1299_REG_CHNSET_GAIN_24            |
94                                                        ADS1299_REG_CHNSET_SRB2_DISCONNECTED  |
95                                                        ADS1299_REG_CHNSET_NORMAL_ELECTRODE);
96      /* Write to CH2 settings register, set as normal input, gain 24 */
97      ads1299_wreg(chip_select, ADS1299_REGADDR_CH2SET, ADS1299_REG_CHNSET_CHANNEL_ON         |
98                                                        ADS1299_REG_CHNSET_GAIN_24            |
99                                                        ADS1299_REG_CHNSET_SRB2_DISCONNECTED  |
100                                                       ADS1299_REG_CHNSET_NORMAL_ELECTRODE);
101     /* Write to CH3 settings register, set as normal input, gain 24 */
102     ads1299_wreg(chip_select, ADS1299_REGADDR_CH3SET, ADS1299_REG_CHNSET_CHANNEL_ON         |
103                                                       ADS1299_REG_CHNSET_GAIN_24            |
104                                                       ADS1299_REG_CHNSET_SRB2_DISCONNECTED  |
105                                                       ADS1299_REG_CHNSET_NORMAL_ELECTRODE);
106     /* Write to CH4 settings register, set as normal input, gain 24 */
107     ads1299_wreg(chip_select, ADS1299_REGADDR_CH4SET, ADS1299_REG_CHNSET_CHANNEL_ON         |
108                                                       ADS1299_REG_CHNSET_GAIN_24            |
109                                                       ADS1299_REG_CHNSET_SRB2_DISCONNECTED  |
110                                                       ADS1299_REG_CHNSET_NORMAL_ELECTRODE);
111     /* Write to CH5 settings register, set as normal input, gain 24 */
112     ads1299_wreg(chip_select, ADS1299_REGADDR_CH5SET, ADS1299_REG_CHNSET_CHANNEL_ON         |
113                                                       ADS1299_REG_CHNSET_GAIN_24            |
114                                                       ADS1299_REG_CHNSET_SRB2_DISCONNECTED  |
115                                                       ADS1299_REG_CHNSET_NORMAL_ELECTRODE);
116     /* Write to CH6 settings register, set as normal input, gain 24 */
117     ads1299_wreg(chip_select, ADS1299_REGADDR_CH6SET, ADS1299_REG_CHNSET_CHANNEL_ON         |
118                                                       ADS1299_REG_CHNSET_GAIN_24            |
119                                                       ADS1299_REG_CHNSET_SRB2_DISCONNECTED  |
120                                                       ADS1299_REG_CHNSET_NORMAL_ELECTRODE);
121     /* Write to CH5 settings register, set as normal input, gain 24 */
122     ads1299_wreg(chip_select, ADS1299_REGADDR_CH7SET, ADS1299_REG_CHNSET_CHANNEL_ON         |
123                                                       ADS1299_REG_CHNSET_GAIN_24            |
124                                                       ADS1299_REG_CHNSET_SRB2_DISCONNECTED  |
125                                                       ADS1299_REG_CHNSET_NORMAL_ELECTRODE);
126     /* Write to CH6 settings register, set as normal input, gain 24 */
```

```
127     ads1299_wreg(chip_select, ADS1299_REGADDR_CH8SET, ADS1299_REG_CHNSET_CHANNEL_ON         |
128                                                        ADS1299_REG_CHNSET_GAIN_24            |
129                                                        ADS1299_REG_CHNSET_SRB2_DISCONNECTED  |
130                                                        ADS1299_REG_CHNSET_NORMAL_ELECTRODE);
131
132     /* Write to MISC1 register, SRB1 on (ref electrode) */
133     ads1299_wreg(chip_select, ADS1299_REGADDR_MISC1, ADS1299_REG_MISC1_SRB1_ON);
134
135     return ADS1299_STATUS_OK;
136     #else
137     #endif /* #if UC3 */
138 }
139
140
141
142 /* REGISTER READ/WRITE FUNCTIONS ************************************************************** ↙
        ****************************************/
143
144 ads1299_error_t ads1299_rreg(uint8_t chip_select, uint8_t reg_addr, uint8_t* read_reg_val_ptr)
145 {
146     #if UC3
147     uint16_t read_data;
148
149     spi_selectChip(SPI_ADDRESS, chip_select);
150
151     /* First byte: read command for specified register */
152     ads1299_send_byte_no_cs(ADS1299_OPC_RREG | reg_addr);
153
154     /* Second byte: Read only 1 register (send n-1, where n is number of registers to read) */
155     ads1299_send_byte_no_cs(0x00);
156
157     /* Dummy byte to clock in data */
158     ads1299_send_byte_no_cs(DUMMY_BYTE);
159
160     delay_us(10);
161     spi_unselectChip(SPI_ADDRESS, chip_select);
162
163     /* Read SPI RX register */
164     read_data = spi_get(SPI_ADDRESS);
165     *read_reg_val_ptr = (uint8_t) read_data;
166
167     return ADS1299_STATUS_OK;
168     #else
169     #endif /* #if UC3 */
170 }
171
172 ads1299_error_t ads1299_wreg(uint8_t chip_select, uint8_t reg_addr, uint8_t reg_val_to_write)
173 {
174     #if UC3
175     spi_selectChip(SPI_ADDRESS, chip_select);
176
177     /* First byte: write command for specified register */
178     ads1299_send_byte_no_cs(ADS1299_OPC_WREG | reg_addr);
179
180     /* Second byte: number of registers to write (1) */
181     ads1299_send_byte_no_cs(0x00);
182
183     /* Third byte: write register value */
184     ads1299_send_byte_no_cs(reg_val_to_write);
185
186     spi_unselectChip(SPI_ADDRESS, chip_select);
187
188     return ADS1299_STATUS_OK;
189     #else
190     #endif /* #if UC3 */
191 }
```

```
192
193 /* DATA RETRIEVAL FUNCTIONS ************************************************************************
    ****************************************/
194
195 ads1299_error_t ads1299_rdata32_packet(uint8_t chip_select, volatile uint32_t sample_idx,
    bboard_data32bit_packet_t* packet_ptr)
196 {
197     #if UC3
198     volatile uint8_t channel_idx;
199     union {
200         uint32_t raw;
201         uint8_t status[4];
202     } __attribute__((packed)) statustemp;
203     union {
204         int32_t raw;
205         uint8_t data[4];
206     } __attribute__((packed)) sigtemp;
207     statustemp.raw = 0;
208     sigtemp.raw = 0;
209
210     /* Begin SPI comms */
211     spi_selectChip(SPI_ADDRESS, chip_select);
212
213     /* Function assumes we've already sent RDATA command or are in RDATAC mode */
214
215     /* Read in status word first (24 bits) */
216     spi_read_packet(SPI_ADDRESS, &statustemp.status[1], 3);
217     packet_ptr->eegstatus = statustemp.raw;
218
219     /* Begin reading in data */
220     for (channel_idx = 0; channel_idx < MAX_EEG_CHANNELS; channel_idx++)
221     {
222         spi_read_packet(SPI_ADDRESS, &sigtemp.data[1], 3);
223         packet_ptr->eegdata[sample_idx][channel_idx] = SIGN_EXT_24(sigtemp.raw);
224     }
225
226     spi_unselectChip(SPI_ADDRESS, chip_select);
227
228     #else
229     #endif  /* #if UC3 */
230     return ADS1299_STATUS_OK;
231 }
232
233 ads1299_error_t ads1299_rdata24_packet(uint8_t chip_select, volatile uint32_t sample_idx,
    bboard_data24bit_packet_t* packet_ptr)
234 {
235     #if UC3
236     volatile uint8_t channel_idx;
237
238     uint8_t temp[3];
239
240     /* Begin SPI comms */
241     spi_selectChip(SPI_ADDRESS, chip_select);
242
243     /* Function assumes we've already sent RDATA command or are in RDATAC mode */
244
245     /* Read in status word first (24 bits) */
246     spi_read_packet(SPI_ADDRESS, temp, 3);
247     packet_ptr->eegstatus[0] = temp[0];
248     packet_ptr->eegstatus[1] = temp[1];
249     packet_ptr->eegstatus[2] = temp[2];
250
251     /* Begin reading in data */
252     for (channel_idx = 0; channel_idx < MAX_EEG_CHANNELS; channel_idx++)
253     {
254         spi_read_packet(SPI_ADDRESS, temp, 3);
```

```
255         packet_ptr->eegdata[sample_idx][channel_idx][0] = temp[0];
256         packet_ptr->eegdata[sample_idx][channel_idx][1] = temp[1];
257         packet_ptr->eegdata[sample_idx][channel_idx][2] = temp[2];
258     }
259
260     spi_unselectChip(SPI_ADDRESS, chip_select);
261
262     #else
263     #endif  /* #if UC3 */
264     return ADS1299_STATUS_OK;
265 }
266
267 ads1299_error_t ads1299_rdata24_generic(uint8_t chip_select, volatile uint32_t sample_idx, volatile
        uint8_t status_array[], volatile uint8_t data_array[][MAX_EEG_CHANNELS][3])
268 {
269     #if UC3
270     volatile uint8_t channel_idx;
271     uint8_t sigtemp[3];
272
273     /* Begin SPI comms */
274     spi_selectChip(SPI_ADDRESS, chip_select);
275
276     /* Function assumes we've already sent RDATA command or are in RDATAC mode */
277
278     /* Read in status word first (24 bits) */
279     spi_read_packet(SPI_ADDRESS, (uint8_t*) status_array, 3);
280
281     /* Begin reading in data */
282     for (channel_idx = 0; channel_idx < MAX_EEG_CHANNELS; channel_idx++)
283     {
284         spi_read_packet(SPI_ADDRESS, sigtemp, 3);
285         data_array[sample_idx][channel_idx][0] = sigtemp[0];
286         data_array[sample_idx][channel_idx][1] = sigtemp[1];
287         data_array[sample_idx][channel_idx][2] = sigtemp[2];
288     }
289
290     spi_unselectChip(SPI_ADDRESS, chip_select);
291
292     #else
293     #endif  /* #if UC3 */
294     return ADS1299_STATUS_OK;
295 }
296
297 ads1299_error_t ads1299_rdata32_generic(uint8_t chip_select, volatile uint32_t sample_idx, volatile
        uint32_t status, volatile int32_t data_array[][MAX_EEG_CHANNELS])
298 {
299     #if UC3
300     volatile uint8_t channel_idx;
301     union {
302         uint32_t raw;
303         uint8_t status[4];
304     } __attribute__((packed)) statustemp;
305     union {
306         int32_t raw;
307         uint8_t data[4];
308     } __attribute__((packed)) sigtemp;
309     statustemp.raw = 0;
310     sigtemp.raw = 0;
311
312     /* Begin SPI comms */
313     spi_selectChip(SPI_ADDRESS, chip_select);
314
315     /* Function assumes we've already sent RDATA command or are in RDATAC mode */
316
317     /* Read in status word first (24 bits) */
318     spi_read_packet(SPI_ADDRESS, &statustemp.status[1], 3);
```

```
319     status = statustemp.raw;
320
321     /* Begin reading in data */
322     /* Begin reading in data */
323     for (channel_idx = 0; channel_idx < MAX_EEG_CHANNELS; channel_idx++)
324     {
325         spi_read_packet(SPI_ADDRESS, &sigtemp.data[1], 3);
326         data_array[sample_idx][channel_idx] = SIGN_EXT_24(sigtemp.raw);
327     }
328
329     spi_unselectChip(SPI_ADDRESS, chip_select);
330
331     #else
332     #endif  /* #if UC3 */
333     return ADS1299_STATUS_OK;
334 }
335
336 #ifdef __cplusplus
337 }
338 #endif
```

```
/**
 *  \file ads1299.h
 *  \brief Configuration settings, register definitions, and function prototypes for using the ADS1299 EEG analog front-end from Texas Instruments.
 *  \author Graham Kelly
 *  \version 1.0
 *  \date August 2014
 *
 *  All Atmel Software Framework libraries used herein are copyright Atmel and
 *  subject to their appropriate licenses, which allow free redistribution with
 *  some restrictions. These restrictions are listed in their appropriate files.
 *
 *  \page License
 *
 *  Brainboard firmware code is placed under the MIT license
 *  Copyright (c) 2014 Graham Kelly
 *
 *  Permission is hereby granted, free of charge, to any person obtaining a copy
 *  of this software and associated documentation files (the "Software"), to deal
 *  in the Software without restriction, including without limitation the rights
 *  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 *  copies of the Software, and to permit persons to whom the Software is
 *  furnished to do so, subject to the following conditions:
 *
 *  The above copyright notice and this permission notice shall be included in
 *  all copies or substantial portions of the Software.
 *
 *  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 *  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 *  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 *  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 *  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 *  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 *  THE SOFTWARE.
 *
 */

#ifndef _ADS1299_H_
#define _ADS1299_H_

#ifdef __cplusplus
extern "C" {
#endif

#include "brainboard.h"
#include "comms.h"
#include <spi_master.h>
#include <gpio.h>
#include <delay.h>

/******************************************************************************************************************************************
 *              Sampling Config                                                                                                           *
 ******************************************************************************************/

/**
 *  \brief Default data rate from the ADS1299.
 *
 *  To monitor electrode impedance continuously, an AC current is pulsed through each electrode and the corresponding
 *  voltage perturbation observed in the measured signal. This signal will not be easily separable from the EEG if it
 *  is within the typical 0-100 Hz EEG bandwidth; since the fastest possible AC excitation rate the ADS1299 can
```

```
 *  generate is (data rate)/4, the lowest recommended data rate that allows continuous impedance monitoring is 1000 Hz.
 *  Using a 500 Hz data rate will generate an AC excitation at 125 Hz, which is dangerously close to, if not in,
 *  the EEG band.
 */
#define DEFAULT_SAMPLE_RATE         250


/*****************************************************************************************************************
*               Other Useful Definitions                                                                         *
*****************************************************************************************************************/


#define SIGN_EXT_24(VAL)                        ((int32_t)((uint32_t)(VAL) ^ (1UL<<(23))) - (1L<<(23)))

/* Default register values */
#define ADS1299_REGDEFAULT_ID           ADS1299_DEVICE_ID
#define ADS1299_REGDEFAULT_CONFIG1      0x96            ///< Multiple readback mode, OSC output disabled, DR = FMOD/4096
#define ADS1299_REGDEFAULT_CONFIG2      0xD0            ///< Test signal sourced internally, low-amplitude test signal pulsed at FCLK/(2^21)
#define ADS1299_REGDEFAULT_CONFIG3      0x68            ///< Ref buffer off, bias measurement off, internal bias ref, bias buffer off, bias sense disabled
#define ADS1299_REGDEFAULT_LOFF         0x00            ///< 95%/5% LOFF comparator threshold, DC lead-off at 6 nA
#define ADS1299_REGDEFAULT_CHNSET       0xE0            ///< Channel off, gain 24, SRB2 disconnected, normal electrode input
#define ADS1299_REGDEFAULT_BIAS_SENSP   0x00            ///< All BIAS channels disconnected from positive leads
#define ADS1299_REGDEFAULT_BIAS_SENSN   0x00            ///< All BIAS channels disconnected from negative leads
#define ADS1299_REGDEFAULT_LOFF_SENSP   0x00            ///< All LOFF channels disconnected from positive leads
#define ADS1299_REGDEFAULT_LOFF_SENSN   0x00            ///< All LOFF channels disconnected from negative leads
#define ADS1299_REGDEFAULT_LOFF_FLIP    0x00            ///< No flipping in this house; source/pull-up at INP, sink/pull-down at INN
#define ADS1299_REGDEFAULT_LOFF_STATP   0x00            ///< This is a read-only register; reset value is 0x00
#define ADS1299_REGDEFAULT_LOFF_STATN   0x00            ///< This is a read-only register; reset value is 0x00
#define ADS1299_REGDEFAULT_GPIO         0x0F            ///< All GPIO set to inputs
#define ADS1299_REGDEFAULT_MISC1        0x00            ///< SRB1 disconnected
#define ADS1299_REGDEFAULT_MISC2        0x00            ///< Register not used in this silicon; should stay at 0x00
#define ADS1299_REGDEFAULT_CONFIG4      0x00            ///< Continuous conversion, LOFF comparator powered down


/*****************************************************************************************************************
*               Typedefs and Struct Declarations/Definitions                                                     *
*****************************************************************************************************************/

/**
 *  \brief Error codes for interacting with the ADS1299.
 *
 */
typedef enum
{
```

```
104     ADS1299_STATUS_OK               = 0,        ///< No error.
105     ADS1299_ERROR_SPI_TIMEOUT       = 1,        ///< SPI timed out. Check SPI configuration and
        hardware connections.
106     /* Expand with other codes if desired */
107 } ads1299_error_t;
108
109 /**
110  *  \brief ADS1299 register addresses.
111  *
112  * Consult the ADS1299 datasheet and user's guide for more information.
113  */
114 #define ADS1299_REGADDR_ID          0x00          ///< Chip ID register. Read-only. ID[4:0] should be
         11110.
115 #define ADS1299_REGADDR_CONFIG1     0x01          ///< Configuration register 1. Controls daisy-chain
         mode; clock output; and data rate.
116 #define ADS1299_REGADDR_CONFIG2     0x02          ///< Configuration register 2. Controls calibration
         signal source, amplitude, and frequency.
117 #define ADS1299_REGADDR_CONFIG3     0x03          ///< Configuration register 3. Controls reference
        buffer power and the bias functionality.
118 #define ADS1299_REGADDR_LOFF        0x04          ///< Lead-off control register. Controls lead-off
        frequency, magnitude, and threshold.
119 #define ADS1299_REGADDR_CH1SET      0x05          ///< Channel 1 settings register. Controls channel
        1 input mux, SRB2 switch, gain, and power-down.
120 #define ADS1299_REGADDR_CH2SET      0x06          ///< Channel 2 settings register. Controls channel
        2 input mux, SRB2 switch, gain, and power-down.
121 #define ADS1299_REGADDR_CH3SET      0x07          ///< Channel 3 settings register. Controls channel
        3 input mux, SRB2 switch, gain, and power-down.
122 #define ADS1299_REGADDR_CH4SET      0x08          ///< Channel 4 settings register. Controls channel
        4 input mux, SRB2 switch, gain, and power-down.
123 #define ADS1299_REGADDR_CH5SET      0x09          ///< Channel 5 settings register. Controls channel
        5 input mux, SRB2 switch, gain, and power-down.
124 #define ADS1299_REGADDR_CH6SET      0x0A          ///< Channel 6 settings register. Controls channel
        6 input mux, SRB2 switch, gain, and power-down.
125 #define ADS1299_REGADDR_CH7SET      0x0B          ///< Channel 7 settings register. Controls channel
        7 input mux, SRB2 switch, gain, and power-down.
126 #define ADS1299_REGADDR_CH8SET      0x0C          ///< Channel 8 settings register. Controls channel
        8 input mux, SRB2 switch, gain, and power-down.
127 #define ADS1299_REGADDR_BIAS_SENSP  0x0D          ///< Bias drive positive sense selection. Selects
        channels for bias drive derivation (positive side).
128 #define ADS1299_REGADDR_BIAS_SENSN  0x0E          ///< Bias drive negative sense selection. Selects
        channels for bias drive derivation (negative side).
129 #define ADS1299_REGADDR_LOFF_SENSP  0x0F          ///< Lead-off positive sense selection. Selects
        channels that will use lead-off detection (positive side).
130 #define ADS1299_REGADDR_LOFF_SENSN  0x10          ///< Lead-off negative sense selection. Selects
        channels that will use lead-off detection (negative side).
131 #define ADS1299_REGADDR_LOFF_FLIP   0x11          ///< 1: Swaps lead-off current source and sink on
        the corresponding channel. See datasheet.
132 #define ADS1299_REGADDR_LOFF_STATP  0x12          ///< Lead-off positive side status register. Read-
        only. 0: lead on, 1: lead off.
133 #define ADS1299_REGADDR_LOFF_STATN  0x13          ///< Lead-off negative side status register. Read-
        only. 0: lead on, 1: lead off.
134 #define ADS1299_REGADDR_GPIO        0x14          ///< GPIO register. Controls state and direction of
         the ADS1299 GPIO pins.
135 #define ADS1299_REGADDR_MISC1       0x15          ///< Miscellaneous 1. Connects/disconnects SRB1 to
        all channels' inverting inputs.
136 #define ADS1299_REGADDR_MISC2       0x16          ///< Miscellaneous 2. No functionality in current
        revision of ADS1299.
137 #define ADS1299_REGADDR_CONFIG4     0x17          ///< Configuration register 4. Enables/disables
        single-shot mode and controls lead-off comparator power.
138
139 /**
140  *  \brief ADS1299 SPI communication opcodes.
141  *
142  * Consult the ADS1299 datasheet and user's guide for more information.
143  * For RREG and WREG opcodes, the first byte (opcode) must be ORed with the address of the register to
        be read/written.
```

```
144  * The command is completed with a second byte 000n nnnn, where n nnnn is (# registers to read) - 1.
145  */
146 #define ADS1299_OPC_WAKEUP       0x02          ///< Wake up from standby.
147 #define ADS1299_OPC_STANDBY      0x04          ///< Enter standby.
148 #define ADS1299_OPC_RESET        0x06          ///< Reset all registers.
149 #define ADS1299_OPC_START        0x08          ///< Start data conversions.
150 #define ADS1299_OPC_STOP         0x0A          ///< Stop data conversions.
151
152 #define ADS1299_OPC_RDATAC       0x10          ///< Read data continuously (registers cannot be read or written in this mode).
153 #define ADS1299_OPC_SDATAC       0x11          ///< Stop continuous data read.
154 #define ADS1299_OPC_RDATA        0x12          ///< Read single data value.
155
156 #define ADS1299_OPC_RREG         0x20          ///< Read register value.
157 #define ADS1299_OPC_WREG         0x40          ///< Write register value.
158
159 /* ID REGISTER ********************************************************************************************/
160
161 /**
162  *  \brief Factory-programmed device ID for ADS1299, stored in ID[3:0].
163  */
164 // Factory-programmed device ID for ADS1299. Stored in ID[3:0].
165 #define ADS1299_DEVICE_ID       0b1110
166
167 /* CONFIG1 REGISTER ***************************************************************************************/
168
169 /**
170  *  \brief Bit location and size definitions for CONFIG1.CLK_EN bit (oscillator output on CLK pin en/disabled).
171  *
172  * Consult the ADS1299 datasheet, page 40, for more information.
173  */
174 #define ADS1299_REG_CONFIG1_CLOCK_OUTPUT_DISABLED    (0<<5)
175 #define ADS1299_REG_CONFIG1_CLOCK_OUTPUT_ENABLED     (1<<5)
176
177 /**
178  *  \brief Bit location and size definitions for CONFIG1.DAISY_EN bit.
179  *
180  * Consult the ADS1299 datasheet, pp. 40 and 31-34, for more information.
181  */
182 #define ADS1299_REG_CONFIG1_DAISY_CHAIN_MODE         (0<<6)
183 #define ADS1299_REG_CONFIG1_MULTI_READBACK_MODE      (1<<6)
184
185 /**
186  *  \brief Bit mask definitions for CONFIG1.DR (data rate).
187  *
188  * FMOD = FCLK/2, where FCLK is the clock frequency of the ADS1299. This is normally 2.048 MHz.
189  */
190 #define ADS1299_REG_CONFIG1_FMOD_DIV_BY_64           0      ///< Data is output at FMOD/64, or 16 kHz at 2.048 MHz.
191 #define ADS1299_REG_CONFIG1_FMOD_DIV_BY_128          1      ///< Data is output at FMOD/128, or 8 kHz at 2.048 MHz.
192 #define ADS1299_REG_CONFIG1_FMOD_DIV_BY_256          2      ///< Data is output at FMOD/256, or 4 kHz at 2.048 MHz.
193 #define ADS1299_REG_CONFIG1_FMOD_DIV_BY_512          3      ///< Data is output at FMOD/512, or 2 kHz at 2.048 MHz.
194 #define ADS1299_REG_CONFIG1_FMOD_DIV_BY_1024         4      ///< Data is output at FMOD/1024, or 1 kHz at 2.048 MHz.
195 #define ADS1299_REG_CONFIG1_FMOD_DIV_BY_2048         5      ///< Data is output at FMOD/2048, or 500 Hz at 2.048 MHz.
196 #define ADS1299_REG_CONFIG1_FMOD_DIV_BY_4096         6      ///< Data is output at FMOD/4096, or 250 Hz at 2.048 MHz.
197
198 /**
```

```
 *  \brief Combined value of reserved bits in CONFIG1 register.
 *
 * Consult the ADS1299 datasheet, page 40, for more information.
 */
#define ADS1299_REG_CONFIG1_RESERVED_VALUE          (1<<7)|(1<<4)

/* CONFIG2 REGISTER ******************************************************************************************/

/**
 *  \brief Bit mask definitions for CONFIG2.CAL_FREQ (calibration signal frequency).
 *
 * FCLK is the clock frequency of the ADS1299. This is normally 2.048 MHz.
 */
#define ADS1299_REG_CONFIG2_CAL_PULSE_FCLK_DIV_2_21     0       ///< Calibration signal pulsed at FCLK/2^21, or approx. 1 Hz at 2.048 MHz.
#define ADS1299_REG_CONFIG2_CAL_PULSE_FCLK_DIV_2_20     1       ///< Calibration signal pulsed at FCLK/2^20, or approx. 2 Hz at 2.048 MHz.
#define ADS1299_REG_CONFIG2_CAL_DC                      3       ///< Calibration signal is not pulsed.


/**
 *  \brief Bit mask definitions for CONFIG2.CAL_AMP0 (calibration signal amplitude).
 */
#define ADS1299_REG_CONFIG2_CAL_AMP_VREF_DIV_2_4_MV     (0<<2)      ///< Calibration signal amplitude is 1 x (VREFP - VREFN)/(2.4 mV).
#define ADS1299_REG_CONFIG2_CAL_AMP_2VREF_DIV_2_4_MV    (1<<2)      ///< Calibration signal amplitude is 2 x (VREFP - VREFN)/(2.4 mV).

/**
 *  \brief Bit mask definitions for CONFIG2.INT_CAL (calibration signal source).
 */
#define ADS1299_REG_CONFIG2_CAL_EXT                     (0<<4)      ///< Calibration signal is driven externally.
#define ADS1299_REG_CONFIG2_CAL_INT                     (1<<4)      ///< Calibration signal is driven internally.

/**
 *  \brief Combined value of reserved bits in CONFIG2 register.
 *
 * Consult the ADS1299 datasheet, page 41, for more information.
 */
#define ADS1299_REG_CONFIG2_RESERVED_VALUE              (6<<5)


/* CONFIG3 REGISTER ******************************************************************************************/

/**
 *  \brief Bit mask definitions for CONFIG3.PD_REFBUF (internal voltage reference buffer enable/disable).
 *
 * Note that disabling the buffer for the internal voltage reference requires that a reference voltage
 * must be externally applied on VREFP for proper operation. This is not related to the reference ELECTRODE
 * buffer, which is an external op-amp on the PCB. Brainboard does not apply a voltage to VREFP, and thus
 * the buffer must be enabled.
 */
#define ADS1299_REG_CONFIG3_REFBUF_DISABLED         (0<<7)
#define ADS1299_REG_CONFIG3_REFBUF_ENABLED          (1<<7)

/**
 *  \brief Bit mask definitions for CONFIG3.BIAS_MEAS (enable or disable bias measurement through BIASIN pin).
 */
```

```
253 #define ADS1299_REG_CONFIG3_BIAS_MEAS_DISABLED          (0<<4)
254 #define ADS1299_REG_CONFIG3_BIAS_MEAS_ENABLED           (1<<4)
255
256 /**
257  *  \brief Bit mask definitions for CONFIG3.BIASREF_INT (bias reference internally or externally
        generated).
258  */
259 #define ADS1299_REG_CONFIG3_BIASREF_EXT                 (0<<3)
260 #define ADS1299_REG_CONFIG3_BIASREF_INT                 (1<<3)
261
262 /**
263  *  \brief Bit mask definitions for CONFIG3.PD_BIAS (power-down or enable bias buffer amplifier).
264  */
265 #define ADS1299_REG_CONFIG3_BIASBUF_DISABLED            (0<<2)
266 #define ADS1299_REG_CONFIG3_BIASBUF_ENABLED             (1<<2)
267
268 /**
269  *  \brief Bit mask definitions for CONFIG3.BIAS_LOFF_SENS (detection of bias lead-off en/disable).
270  */
271 #define ADS1299_REG_CONFIG3_BIAS_LOFF_SENSE_DISABLED    (0<<1)
272 #define ADS1299_REG_CONFIG3_BIAS_LOFF_SENSE_ENABLED     (1<<1)
273
274 /**
275  *  \brief Combined value of reserved bits in CONFIG3 register.
276  *
277  * Consult the ADS1299 datasheet, page 42, for more information.
278  */
279 #define ADS1299_REG_CONFIG3_RESERVED_VALUE              (3<<5)
280
281 /* CONFIG4 REGISTER ************************************************************************
        ****************************************/
282
283 /**
284  *  \brief Bit mask definitions for CONFIG4.SINGLE_SHOT (single-shot or continuous conversion setting).
285  *
286  * This can more easily be set with the RDATAC/SDATAC opcodes.
287  */
288 #define ADS1299_REG_CONFIG4_CONTINUOUS_CONVERSION_MODE      (0<<3)
289 #define ADS1299_REG_CONFIG4_SINGLE_SHOT_MODE                (1<<3)
290
291 /**
292  *  \brief Bit mask definitions for CONFIG4.PD_LOFF_COMP (power-down lead-off comparators).
293  *
294  */
295 #define ADS1299_REG_CONFIG4_LEAD_OFF_DISABLED               (0<<1)
296 #define ADS1299_REG_CONFIG4_LEAD_OFF_ENABLED                (1<<1)
297
298 /**
299  *  \brief Combined value of reserved bits in CONFIG4 register.
300  *
301  * Consult the ADS1299 datasheet, page 47, for more information.
302  */
303 #define ADS1299_REG_CONFIG4_RESERVED_VALUE      0
304
305 /* LOFF REGISTER ***************************************************************************
        ****************************************/
306
307 /**
308  *  \brief Bit mask definitions for LOFF.COMP_TH (lead-off comparator threshold).
309  *
310  * Definition names are for the positive side (LOFFP). The corresponding LOFFN thresholds
311  * are the difference between these thresholds and 100%. Default value is _95_PERCENT.
312  */
313 #define ADS1299_REG_LOFF_95_PERCENT                 (0<<5)
314 #define ADS1299_REG_LOFF_92_5_PERCENT               (1<<5)
315 #define ADS1299_REG_LOFF_90_PERCENT                 (2<<5)
```

```
316 #define ADS1299_REG_LOFF_87_5_PERCENT              (3<<5)
317 #define ADS1299_REG_LOFF_85_PERCENT                (4<<5)
318 #define ADS1299_REG_LOFF_80_PERCENT                (5<<5)
319 #define ADS1299_REG_LOFF_75_PERCENT                (6<<5)
320 #define ADS1299_REG_LOFF_70_PERCENT                (7<<5)
321
322 /**
323  *  \brief Bit mask definitions for LOFF.ILEAD_OFF (lead-off current magnitude).
324  *
325  * This should be as small as possible for continuous lead-off detection, so as not to noticeably alter
326  * the acquired signal. Default is _6_NA.
327  */
328 #define ADS1299_REG_LOFF_6_NA                      (0<<2)          ///< 6 nA lead-off current.
329 #define ADS1299_REG_LOFF_24_NA                     (1<<2)          ///< 24 nA lead-off current.
330 #define ADS1299_REG_LOFF_6_UA                      (2<<2)          ///< 6 uA lead-off current.
331 #define ADS1299_REG_LOFF_24_UA                     (3<<2)          ///< 24 uA lead-off current.

332
333 /**
334  *  \brief Bit mask definitions for LOFF.FLEAD_OFF (lead-off current frequency).
335  *
336  * This should be as large as possible for continuous AC lead-off detection to ensure that it is out
337  * of the EEG frequency band (approx. 0-100 Hz for most applications). The excitation signal can then
338  * be filtered out of the acquired overall signal, and its voltage amplitude measured in order to
     determine
339  * the electrode impedance.
340  * FCLK is the clock frequency of the ADS1299. This is normally 2.048 MHz.
341  * FDR is the output data rate. With the default clock, this must be at least 1 kHz in order to use
342  * continuous AC impedance monitoring, since the excitation frequency of FDR/4 = 250 Hz is the lowest
343  * possible frequency outside of the EEG band. If only a specific band is needed and it is lower than
344  * 62.5 Hz or 125 Hz, the 250/500 Hz settings may be used.
345  */
346 #define ADS1299_REG_LOFF_DC_LEAD_OFF               0       ///< Lead-off current is at DC.
347 #define ADS1299_REG_LOFF_AC_LEAD_OFF_FCLK_DIV_2_18 1       ///< Lead-off current is at FCLK/2^18, or 7
     .8125 Hz at 2.048 MHz.
348 #define ADS1299_REG_LOFF_AC_LEAD_OFF_FCLK_DIV_2_16 2       ///< Lead-off current is at FCLK/2^16, or
     31.25 Hz at 2.048 MHz.
349 #define ADS1299_REG_LOFF_AC_LEAD_OFF_FDR_DIV_4     3       ///< Lead-off current is at FDR/4.
350
351 /**
352  *  \brief Combined value of reserved bits in LOFF register.
353  *
354  */
355 #define ADS1299_REG_LOFF_RESERVED_VALUE            0
356
357 /* CHnSET REGISTERS ***************************************************************************
     ****************************************/
358
359 /**
360  *  \brief Bit mask definitions for CHnSET.PD (channel power-down).
361  */
362 #define ADS1299_REG_CHNSET_CHANNEL_ON          (0<<7)
363 #define ADS1299_REG_CHNSET_CHANNEL_OFF         (1<<7)
364
365 /**
366  *  \brief Bit mask definitions for CHnSET.GAIN (channel PGA gain).
367  *
368  * Take care to ensure that the gain is appropriate for the common-mode level of the device inputs.
369  * Higher gain settings have lower input-referred noise.
370  * Consult the ADS1299 datasheet, pages 6-7 and 19-20, for more information.
371  */
372 #define ADS1299_REG_CHNSET_GAIN_1              (0<<4)          ///< PGA gain = 1.
373 #define ADS1299_REG_CHNSET_GAIN_2              (1<<4)          ///< PGA gain = 2.
374 #define ADS1299_REG_CHNSET_GAIN_4              (2<<4)          ///< PGA gain = 4.
375 #define ADS1299_REG_CHNSET_GAIN_6              (3<<4)          ///< PGA gain = 6.
376 #define ADS1299_REG_CHNSET_GAIN_8              (4<<4)          ///< PGA gain = 8.
```

```
377 #define ADS1299_REG_CHNSET_GAIN_12              (5<<4)          ///< PGA gain = 12.
378 #define ADS1299_REG_CHNSET_GAIN_24              (6<<4)          ///< PGA gain = 24.
379
380 /**
381  *  \brief Bit mask definitions for CHnSET.SRB2 (channel internal connection to SRB2 pin).
382  */
383 #define ADS1299_REG_CHNSET_SRB2_DISCONNECTED    (0<<3)
384 #define ADS1299_REG_CHNSET_SRB2_CONNECTED       (1<<3)
385
386 /**
387  *  \brief Bit mask definitions for CHnSET.MUX (channel mux setting).
388  *
389  * Controls the channel multiplexing on the ADS1299.
390  * Consult the ADS1299 datasheet, pages 16-17, for more information.
391  */
392 #define ADS1299_REG_CHNSET_NORMAL_ELECTRODE     0           ///< Channel is connected to the
    corresponding positive and negative input pins.
393 #define ADS1299_REG_CHNSET_INPUT_SHORTED        1           ///< Channel inputs are shorted together.
    Used for offset and noise measurements.
394 #define ADS1299_REG_CHNSET_BIAS_MEASUREMENT     2           ///< Used with CONFIG3.BIAS_MEAS for bias
    measurement. See ADS1299 datasheet, pp. 53-54.
395 #define ADS1299_REG_CHNSET_MVDD_SUPPLY          3           ///< Used for measuring analog and digital
    supplies. See ADS1299 datasheet, p. 17.
396 #define ADS1299_REG_CHNSET_TEMPERATURE_SENSOR   4           ///< Measures device temperature. See
    ADS1299 datasheet, p. 17.
397 #define ADS1299_REG_CHNSET_TEST_SIGNAL          5           ///< Measures calibration signal. See
    ADS1299 datasheet, pp. 17 and 41.
398 #define ADS1299_REG_CHNSET_BIAS_DRIVE_P         6           ///< Connects positive side of channel to
    bias drive output.
399 #define ADS1299_REG_CHNSET_BIAS_DRIVE_N         7           ///< Connects negative side of channel to
    bias drive output.
400
401 /**
402  *  \brief Combined value of reserved bits in CHnSET registers.
403  *
404  */
405 #define ADS1299_REG_CHNSET_RESERVED_VALUE       0
406
407 /* BIAS_SENSP REGISTER ************************************************************************
    ****************************************/
408
409 /**
410  *  \brief Bit mask definitions for BIAS_SENSP register (read-only).
411  *
412  * Consult the ADS1299 datasheet, page 44, for more information.
413  */
414 #define ADS1299_REG_BIAS_SENSP_BIASP8   (1<<7)
415 #define ADS1299_REG_BIAS_SENSP_BIASP7   (1<<6)
416 #define ADS1299_REG_BIAS_SENSP_BIASP6   (1<<5)
417 #define ADS1299_REG_BIAS_SENSP_BIASP5   (1<<4)
418 #define ADS1299_REG_BIAS_SENSP_BIASP4   (1<<3)
419 #define ADS1299_REG_BIAS_SENSP_BIASP3   (1<<2)
420 #define ADS1299_REG_BIAS_SENSP_BIASP2   (1<<1)
421 #define ADS1299_REG_BIAS_SENSP_BIASP1   (1<<0)
422
423
424 /* BIAS_SENSN REGISTER ************************************************************************
    ****************************************/
425
426 /**
427  *  \brief Bit mask definitions for BIAS_SENSN register (read-only).
428  *
429  * Consult the ADS1299 datasheet, page 44, for more information.
430  */
431 #define ADS1299_REG_BIAS_SENSN_BIASN8   (1<<7)
432 #define ADS1299_REG_BIAS_SENSN_BIASN7   (1<<6)
```

```
433 #define ADS1299_REG_BIAS_SENSN_BIASN6   (1<<5)
434 #define ADS1299_REG_BIAS_SENSN_BIASN5   (1<<4)
435 #define ADS1299_REG_BIAS_SENSN_BIASN4   (1<<3)
436 #define ADS1299_REG_BIAS_SENSN_BIASN3   (1<<2)
437 #define ADS1299_REG_BIAS_SENSN_BIASN2   (1<<1)
438 #define ADS1299_REG_BIAS_SENSN_BIASN1   (1<<0)
439
440 /* LOFF_SENSP REGISTER ********************************************************************************
        ****************************************/
441
442 /**
443  *  \brief Bit mask definitions for LOFF_SENSP register (read-only).
444  *
445  * Consult the ADS1299 datasheet, page 45, for more information.
446  */
447 #define ADS1299_REG_LOFF_SENSP_LOFFP8   (1<<7)
448 #define ADS1299_REG_LOFF_SENSP_LOFFP7   (1<<6)
449 #define ADS1299_REG_LOFF_SENSP_LOFFP6   (1<<5)
450 #define ADS1299_REG_LOFF_SENSP_LOFFP5   (1<<4)
451 #define ADS1299_REG_LOFF_SENSP_LOFFP4   (1<<3)
452 #define ADS1299_REG_LOFF_SENSP_LOFFP3   (1<<2)
453 #define ADS1299_REG_LOFF_SENSP_LOFFP2   (1<<1)
454 #define ADS1299_REG_LOFF_SENSP_LOFFP1   (1<<0)
455
456
457 /* LOFF_SENSN REGISTER ********************************************************************************
        ****************************************/
458
459 /**
460  *  \brief Bit mask definitions for LOFF_SENSN register (read-only).
461  *
462  * Consult the ADS1299 datasheet, page 45, for more information.
463  */
464 #define ADS1299_REG_LOFF_SENSN_LOFFN8   (1<<7)
465 #define ADS1299_REG_LOFF_SENSN_LOFFN7   (1<<6)
466 #define ADS1299_REG_LOFF_SENSN_LOFFN6   (1<<5)
467 #define ADS1299_REG_LOFF_SENSN_LOFFN5   (1<<4)
468 #define ADS1299_REG_LOFF_SENSN_LOFFN4   (1<<3)
469 #define ADS1299_REG_LOFF_SENSN_LOFFN3   (1<<2)
470 #define ADS1299_REG_LOFF_SENSN_LOFFN2   (1<<1)
471 #define ADS1299_REG_LOFF_SENSN_LOFFN1   (1<<0)
472
473 /* LOFF_FLIP REGISTER ********************************************************************************
        ****************************************/
474
475 /**
476  *  \brief Bit mask definitions for LOFF_FLIP register (read-only).
477  *
478  * Consult the ADS1299 datasheet, page 45, for more information.
479  */
480 #define ADS1299_REG_LOFF_FLIP_LOFF_FLIP8    (1<<7)
481 #define ADS1299_REG_LOFF_FLIP_LOFF_FLIP7    (1<<6)
482 #define ADS1299_REG_LOFF_FLIP_LOFF_FLIP6    (1<<5)
483 #define ADS1299_REG_LOFF_FLIP_LOFF_FLIP5    (1<<4)
484 #define ADS1299_REG_LOFF_FLIP_LOFF_FLIP4    (1<<3)
485 #define ADS1299_REG_LOFF_FLIP_LOFF_FLIP3    (1<<2)
486 #define ADS1299_REG_LOFF_FLIP_LOFF_FLIP2    (1<<1)
487 #define ADS1299_REG_LOFF_FLIP_LOFF_FLIP1    (1<<0)
488
489
490 /* LOFF_STATP REGISTER ********************************************************************************
        ****************************************/
491
492 /**
493  *  \brief Bit mask definitions for LOFF_STATP register (read-only).
494  *
```

```
 * Consult the ADS1299 datasheet, page 45, for more information.
 */
#define ADS1299_REG_LOFF_STATP_IN8P_OFF (1<<7)
#define ADS1299_REG_LOFF_STATP_IN7P_OFF (1<<6)
#define ADS1299_REG_LOFF_STATP_IN6P_OFF (1<<5)
#define ADS1299_REG_LOFF_STATP_IN5P_OFF (1<<4)
#define ADS1299_REG_LOFF_STATP_IN4P_OFF (1<<3)
#define ADS1299_REG_LOFF_STATP_IN3P_OFF (1<<2)
#define ADS1299_REG_LOFF_STATP_IN2P_OFF (1<<1)
#define ADS1299_REG_LOFF_STATP_IN1P_OFF (1<<0)

/* LOFF_STATN REGISTER ********************************************************************************
    ****************************************/

/**
 *  \brief Bit mask definitions for LOFF_STATN register (read-only).
 *
 * Consult the ADS1299 datasheet, page 45, for more information.
 */
#define ADS1299_REG_LOFF_STATN_IN8N_OFF (1<<7)
#define ADS1299_REG_LOFF_STATN_IN7N_OFF (1<<6)
#define ADS1299_REG_LOFF_STATN_IN6N_OFF (1<<5)
#define ADS1299_REG_LOFF_STATN_IN5N_OFF (1<<4)
#define ADS1299_REG_LOFF_STATN_IN4N_OFF (1<<3)
#define ADS1299_REG_LOFF_STATN_IN3N_OFF (1<<2)
#define ADS1299_REG_LOFF_STATN_IN2N_OFF (1<<1)
#define ADS1299_REG_LOFF_STATN_IN1N_OFF (1<<0)

/* GPIO REGISTER **************************************************************************************
    ****************************************/

/**
 *  \brief Bit mask definitions for GPIO.GPIODn (GPIO direction bits).
 *
 * The ADS1299 has 4 GPIO pins that can be manipulated via the SPI bus if there are not enough
 * GPIO pins available on the host.
 * GPIOD[4:1] controls the logic levels on GPIO pins 4:1.
 *
 * Consult the ADS1299 datasheet, page 46, for more information.
 */
#define ADS1299_REG_GPIO_GPIOD4_LOW         (0<<7)
#define ADS1299_REG_GPIO_GPIOD4_HIGH        (1<<7)
#define ADS1299_REG_GPIO_GPIOD3_LOW         (0<<6)
#define ADS1299_REG_GPIO_GPIOD3_HIGH        (1<<6)
#define ADS1299_REG_GPIO_GPIOD2_LOW         (0<<5)
#define ADS1299_REG_GPIO_GPIOD2_HIGH        (1<<5)
#define ADS1299_REG_GPIO_GPIOD1_LOW         (0<<4)
#define ADS1299_REG_GPIO_GPIOD1_HIGH        (1<<4)

/**
 *  \brief Bit mask definitions for GPIO.GPIOCn (GPIO level).
 *
 * The ADS1299 has 4 GPIO pins that can be manipulated via the SPI bus if there are not enough
 * GPIO pins available on the host.
 * GPIOC[4:1] controls the pin direction on GPIO pins 4:1.
 *
 * Consult the ADS1299 datasheet, page 46, for more information.
 */
#define ADS1299_REG_GPIO_GPIOC4_OUTPUT      (0<<3)
#define ADS1299_REG_GPIO_GPIOC4_INPUT       (1<<3)
#define ADS1299_REG_GPIO_GPIOC3_OUTPUT      (0<<2)
#define ADS1299_REG_GPIO_GPIOC3_INPUT       (1<<2)
#define ADS1299_REG_GPIO_GPIOC2_OUTPUT      (0<<1)
#define ADS1299_REG_GPIO_GPIOC2_INPUT       (1<<1)
#define ADS1299_REG_GPIO_GPIOC1_OUTPUT      (0<<0)
#define ADS1299_REG_GPIO_GPIOC1_INPUT       (1<<0)
```

```
559
560 /**
561  *  \brief Combined value of reserved bits in GPIO register.
562  *
563  */
564 #define ADS1299_REG_GPIO_RESERVED_VALUE             0
565
566 /* MISC1 REGISTER ********************************************************************************
    ****************************************/
567
568 /**
569  *  \brief Bit mask definitions for MISC1.SRB1 (SRB1 internal connection).
570  */
571 #define ADS1299_REG_MISC1_SRB1_OFF      (0<<5)          ///< Stim/ref/bias 1 turned off.
572 #define ADS1299_REG_MISC1_SRB1_ON       (1<<5)          ///< Stim/ref/bias 1 connected to all channel
    inverting inputs.
573
574 /**
575  *  \brief Combined value of reserved bits in MISC1 register.
576  *
577  */
578 #define ADS1299_REG_MISC1_RESERVED_VALUE                0
579
580
581 /* MISC2 REGISTER ********************************************************************************
    ****************************************/
582
583 /**
584  *  \brief Combined value of reserved bits in MISC2 register.
585  *
586  * MISC2 don't do nothin' right now!
587  * Consult the ADS1299 user's guide, page 46, for more information.
588  */
589 #define ADS1299_REG_MISC2_RESERVED_VALUE                0
590
591
592 /************************************************************************************************
    ****************************************
593 *               Prototypes                                                                      *
                                            *
594 ************************************************************************************************
    ****************************************/
595
596 /**
597  *  \brief Initialize the ADS1299.
598  *
599  * This function performs the power-on reset and initialization procedure documented on page 58 of the
600  * ADS1299 datasheet, up to "Send SDATAC Command."
601  *
602  * \pre Requires spi.h from the Atmel Software Framework and ads1299_spi_adapt.h.
603  * \param chip_select The chip select number of the ADS1299 to be initialized.
604  * \return Zero if successful, or an error code if unsuccessful.
605  */
606 ads1299_error_t ads1299_device_init(uint8_t);
607
608 /**
609  *  \brief Read a single register from the ADS1299.
610  *
611  * This function sends the RREG opcode, logical OR'd with the specified register address, and
612  * writes the obtained register value to a variable. This command will have no effect if the
613  * device is in continuous read mode.
614  *
615  * \pre Requires spi.h from the Atmel Software Framework and ads1299_spi_adapt.h.
616  * \param chip_select The chip select number of the ADS1299 to be initialized.
617  * \param reg_addr The register address of the register to be read.
618  * \param read_reg_val_ptr Pointer to the variable to store the read register value.
```

```
619  * \return Zero if successful, or an error code if unsuccessful.
620  */
621 ads1299_error_t ads1299_rreg(uint8_t, uint8_t, uint8_t*);
622
623 /**
624  *  \brief Write a single register on the ADS1299.
625  *
626  * This function sends the WREG opcode, logical OR'd with the specified register address, and
627  * then writes the specified value to that register. This command will have no effect if the
628  * device is in continuous read mode.
629  *
630  * \pre Requires spi.h from the Atmel Software Framework and ads1299_spi_adapt.h.
631  * \param chip_select The chip select number of the ADS1299 to be initialized.
632  * \param reg_addr The register address of the register to be written.
633  * \param reg_val_to_write The value to be written to the specified register.
634  * \return Zero if successful, or an error code if unsuccessful.
635  */
636 ads1299_error_t ads1299_wreg(uint8_t, uint8_t, uint8_t);
637
638 /**
639  *  \brief Read a single 24-bit data sample for each enabled channel.
640  *
641  * This function reads back the 24-bit status word and 8 channels of 24-bit data.
642  * These 24-bit values are stored right-aligned in 32-bit integers.
643  *
644  * \param chip_select The chip select number of the ADS1299 to be initialized.
645  * \param sample_idx The index within each channel array of the sample to be acquired.
646  * \param packet_ptr Pointer to EEG data packet structure.
647  * \return Zero if successful, or an error code if unsuccessful.
648  */
649 ads1299_error_t ads1299_rdata32_packet(uint8_t, volatile uint32_t, bboard_data32bit_packet_t*);
650
651 /**
652  *  \brief Read a single 24-bit data sample for each enabled channel.
653  *
654  * This function reads back the 24-bit status word and 8 channels of 24-bit data.
655  * These 24-bit values are stored right-aligned in 32-bit integers.
656  *
657  * \param chip_select The chip select number of the ADS1299 to be initialized.
658  * \param sample_idx The index within each channel array of the sample to be acquired.
659  * \param status ADS1299 status word.
660  * \param data_array Array of 32-bit signed ints (dsp32_t from the ASF is equivalent) for storing EEG ↙
       channel data.
661  * \return Zero if successful, or an error code if unsuccessful.
662  */
663 ads1299_error_t ads1299_rdata32_generic(uint8_t, volatile uint32_t, volatile uint32_t, volatile int32_t ↙
       [][MAX_EEG_CHANNELS]);
664
665 /**
666  *  \brief Read a single 24-bit data sample for each enabled channel into a specified packet structure.
667  *
668  * This function reads back the 24-bit status word and 8 channels of 24-bit data.
669  * These 24-bit values are stored in two arrays: one 3-byte array for the status
670  * word, and a 24-byte array for the data.
671  *
672  * \param chip_select The chip select number of the ADS1299 to be initialized.
673  * \param sample_idx The index within each channel array of the sample to be acquired.
674  * \param packet_ptr Pointer to EEG data packet structure.
675  * \return Zero if successful, or an error code if unsuccessful.
676  */
677 ads1299_error_t ads1299_rdata24_packet(uint8_t, volatile uint32_t, bboard_data24bit_packet_t*);
678
679 /**
680  *  \brief Read a single 24-bit data sample for each enabled channel into an array of channels.
681  *
682  * This function reads back the 24-bit status word and 8 channels of 24-bit data.
```

```
683  * These 24-bit values are stored in two arrays: one 3-byte array for the status
684  * word, and a 24-byte array for the data.
685  *
686  * \param chip_select The chip select number of the ADS1299 to be initialized.
687  * \param sample_idx The index within each channel array of the sample to be acquired.
688  * \param status_array Array of 3 bytes representing the 24-bit status word.
689  * \param data_array Array of dimensions [BUFFERSIZE][EEG_CHANNELS][3] representing EEG data.
690  * \return Zero if successful, or an error code if unsuccessful.
691  */
692 ads1299_error_t ads1299_rdata24_generic(uint8_t, volatile uint32_t, volatile uint8_t[], volatile
        uint8_t[][MAX_EEG_CHANNELS][3]);
693
694 /* INLINE FUNCTIONS ************************************************************************
        ****************************************/
695
696 /**
697  *  \brief Send a single byte to the ADS1299 without manipulating chip select.
698  *
699  *  Use this function when multiple bytes need to be sent and you want the chip to remain selected
700  *  throughout the process.
701  *
702  * \pre Requires spi.h from the Atmel Software Framework and ads1299_spi_adapt.h.
703  * \param opcode The opcode to send.
704  * \return Zero if successful, or an error code if unsuccessful.
705  */
706 static inline ads1299_error_t ads1299_send_byte_no_cs(uint8_t opcode)
707 {
708     while(!spi_is_tx_ready(SPI_ADDRESS));
709     spi_put(SPI_ADDRESS, opcode);
710     while(!spi_is_tx_empty(SPI_ADDRESS));
711
712     return ADS1299_STATUS_OK;
713 }
714
715 /**
716  *  \brief Send a single opcode to the ADS1299.
717  *
718  * This function sends the specified byte to the ADS1299. Chip select is cleared (activated) and set
719  * (inactivated) within the function.
720  *
721  * \pre Requires spi.h from the Atmel Software Framework and ads1299_spi_adapt.h.
722  * \param chip_select The chip select number of the ADS1299 to be initialized.
723  * \param opcode The opcode to send.
724  * \return Zero if successful, or an error code if unsuccessful.
725  */
726 static inline ads1299_error_t ads1299_send_byte(uint8_t chip_select, uint8_t opcode)
727 {
728     //#if UC3
729     spi_selectChip(SPI_ADDRESS, chip_select);
730
731     ads1299_send_byte_no_cs(opcode);
732
733     spi_unselectChip(SPI_ADDRESS, chip_select);
734
735     return ADS1299_STATUS_OK;
736     //#else
737     //#endif
738 }
739
740 /**
741  *  \brief Put the ADS1299 in standby mode.
742  *
743  * This function sends the STANDBY opcode to the ADS1299. This places the device in a low-power mode by
744  * shutting down all parts of the circuit except for the reference section. Return from standby using
745  * ads1299_wake(). Do not send any other commands during standby mode.
746  *
```

```
747  * \pre Requires spi.h from the Atmel Software Framework and ads1299_spi_adapt.h.
748  * \param chip_select The chip select number of the ADS1299 to be initialized.
749  * \return Zero if successful, or an error code if unsuccessful.
750  */
751 static inline ads1299_error_t ads1299_standby(uint8_t chip_select)
752 {
753     //#if UC3
754     ads1299_send_byte(chip_select, ADS1299_OPC_STANDBY);
755 
756     return ADS1299_STATUS_OK;
757     //#else
758     //#endif
759 }
760 
761 /**
762  *  \brief Wake the ADS1299 from standby mode.
763  *
764  * This function sends the WAKEUP opcode to the ADS1299. This returns the device to normal operation
765  * after entering standby mode using ads1299_standby(). The host must wait 4 ADS1299 clock cycles
766  * (approximately 2 us at 2.048 MHz) after sending this opcode to allow the device to wake up.
767  *
768  * \pre Requires spi.h from the Atmel Software Framework and ads1299_spi_adapt.h.
769  * \param chip_select The chip select number of the ADS1299 to be initialized.
770  * \return Zero if successful, or an error code if unsuccessful.
771  */
772 static inline ads1299_error_t ads1299_wake(uint8_t chip_select)
773 {
774     //#if UC3
775     ads1299_send_byte(chip_select, ADS1299_OPC_WAKEUP);
776 
777     return ADS1299_STATUS_OK;
778     //#else
779     //#endif
780 }
781 
782 /**
783  *  \brief Start analog-to-digital conversion on the ADS1299 by setting the START pin.
784  *
785  * This function pulls the START pin high, which begins analog-to-digital conversion on the ADS1299.
786  * If conversions are already in progress, this has no effect. Pulling the START pin low
787  * ads1299_hard_stop_conversion() must follow this command by at least 4 ADS1299 clock cycles
788  * (approximately 2 us at 2.048 MHz). This command should not be used if ads1299_soft_start_conversion ↙
       ()
789  * has been used but has not yet been followed by ads1299_soft_stop_conversion().
790  * The START pin is defined at the top of ads1299.h with the macro ADS1299_PIN_START.
791  */
792 static inline void ads1299_hard_start_conversion(void)
793 {
794     //#if UC3
795     #ifdef ADS1299_PIN_START
796     gpio_set_gpio_pin(ADS1299_PIN_START);
797     #endif
798     //#else
799     //#endif
800 }
801 
802 /**
803  *  \brief Stop analog-to-digital conversion on the ADS1299 by clearing the START pin.
804  *
805  * This function pulls the START pin low, which halts analog-to-digital conversion on the ADS1299.
806  * This command must follow pulling the START pin high ads1299_hard_start_conversion() by at least
807  * 4 ADS1299 clock cycles (approximately 2 us at 2.048 MHz).
808  * The START pin is defined at the top of ads1299.h with the macro ADS1299_PIN_START.
809  */
810 static inline void ads1299_hard_stop_conversion(void)
811 {
```

```
812     //#if UC3
813     #ifdef ADS1299_PIN_START
814     gpio_clr_gpio_pin(ADS1299_PIN_START);
815     #endif
816     //#else
817     //#endif
818 }
819
820 /**
821  *  \brief Start analog-to-digital conversion on the ADS1299 using the START opcode.
822  *
823  * This function sends the START opcode, which begins analog-to-digital conversion on the ADS1299.
824  * It is provided in case the START pin is not available for use in the user application.
825  * If conversions are already in progress, this has no effect. The STOP command
       ads1299_soft_stop_conversion()
826  * must follow this command by at least 4 ADS1299 clock cycles (approximately 2 us at 2.048 MHz).
827  * This command should not be used if ads1299_hard_start_conversion() has not yet been followed by
828  * ads1299_hard_stop_conversion().
829  *
830  * \pre Requires spi.h from the Atmel Software Framework and ads1299_spi_adapt.h.
831  * \param chip_select The chip select number of the ADS1299 to be initialized.
832  * \return Zero if successful, or an error code if unsuccessful.
833  */
834 static inline ads1299_error_t ads1299_soft_start_conversion(uint8_t chip_select)
835 {
836     //#if UC3
837     ads1299_send_byte(chip_select, ADS1299_OPC_START);
838
839     return ADS1299_STATUS_OK;
840     //#else
841     //#endif
842 }
843
844 /**
845  *  \brief Stop analog-to-digital conversion on the ADS1299 using the STOP opcode.
846  *
847  * This function sends the STOP opcode, which halts analog-to-digital conversion on the ADS1299.
848  * It is provided in case the START pin is not available for use in the user application.
849  * This command must follow a START opcode ads1299_soft_start_conversion() by at least 4 ADS1299
850  * clock cycles (approximately 2 us at 2.048 MHz). This command should not be used if
851  * ads1299_hard_start_conversion() has not yet been followed by ads1299_hard_stop_conversion().
852  *
853  * \pre Requires spi.h from the Atmel Software Framework and ads1299_spi_adapt.h.
854  * \param chip_select The chip select number of the ADS1299 to be initialized.
855  * \return Zero if successful, or an error code if unsuccessful.
856  */
857 static inline ads1299_error_t ads1299_soft_stop_conversion(uint8_t chip_select)
858 {
859     //#if UC3
860     ads1299_send_byte(chip_select, ADS1299_OPC_STOP);
861
862     return ADS1299_STATUS_OK;
863     //#else
864     //#endif
865 }
866
867 /**
868  *  \brief Enable continuous data output.
869  *
870  * This function sends the RDATAC opcode, which makes conversion data immediately available
871  * to the host as soon as the DRDY pin goes low. The host need only send SCLKs to retrieve
872  * the data, rather than starting with a RDATA command. Registers cannot be read or written
873  * (RREG or WREG) in this mode. Cancel continuous read mode using ads1299_stop_rdatac().
874  *
875  * \pre Requires spi.h from the Atmel Software Framework and ads1299_spi_adapt.h.
876  * \param chip_select The chip select number of the ADS1299 to be initialized.
```

```
877  * \return Zero if successful, or an error code if unsuccessful.
878  */
879 static inline ads1299_error_t ads1299_start_rdatac(uint8_t chip_select)
880 {
881     //#if UC3
882     ads1299_send_byte(chip_select, ADS1299_OPC_RDATAC);
883 
884     return ADS1299_STATUS_OK;
885     //#else
886     //#endif
887 }
888 
889 /**
890  *  \brief Disable continuous data output.
891  *
892  * This function sends the SDATAC opcode, which exits the continuous data mode.
893  *
894  * \pre Requires spi.h from the Atmel Software Framework and ads1299_spi_adapt.h.
895  * \param chip_select The chip select number of the ADS1299 to be initialized.
896  * \return Zero if successful, or an error code if unsuccessful.
897  */
898 static inline ads1299_error_t ads1299_stop_rdatac(uint8_t chip_select)
899 {
900     //#if UC3
901     ads1299_send_byte(chip_select, ADS1299_OPC_SDATAC);
902 
903     return ADS1299_STATUS_OK;
904     //#else
905     //#endif
906 }
907 
908 static inline ads1299_error_t ads1299_powerdn (void)
909 {
910     //#if UC3
911     gpio_clr_gpio_pin(ADS1299_PIN_PWDN);
912     return ADS1299_STATUS_OK;
913 //  #else
914 //  #endif
915 }
916 
917 static inline ads1299_error_t ads1299_powerup (void)
918 {
919 //  #if UC3
920     gpio_set_gpio_pin(ADS1299_PIN_PWDN);
921     return ADS1299_STATUS_OK;
922 //  #else
923 //  #endif
924 }
925 
926 static inline ads1299_error_t ads1299_hard_reset (void)
927 {
928     #if (BRAINBOARD_REV == 0)
929     gpio_clr_gpio_pin(ADS1299_PIN_RESET);
930     delay_us(1);
931     gpio_set_gpio_pin(ADS1299_PIN_RESET);
932     delay_us(10);
933     #endif
934     return ADS1299_STATUS_OK;
935 }
936 
937 static inline ads1299_error_t ads1299_soft_reset (uint8_t chip_select)
938 {
939     ads1299_send_byte(chip_select, ADS1299_OPC_RESET);
940     delay_us(9);
941     return ADS1299_STATUS_OK;
942 }
```

```
943
944 #ifdef __cplusplus
945 }
946 #endif
947
948 #endif // #ifndef _ADS1299_H
949
```

```
/**
 *  \file brainboard.h
 *  \brief Board-specific definitions for current Brainboard revision.
 *  \author Graham Kelly
 *  \version 1.0
 *  \date August 2014
 *
 *  All Atmel Software Framework libraries used herein are copyright Atmel and
 *  subject to their appropriate licenses, which allow free redistribution with
 *  some restrictions. These restrictions are listed in their appropriate files.
 *
 *  \page License
 *
 *  Brainboard firmware code is placed under the MIT license
 *  Copyright (c) 2014 Graham Kelly
 *
 *  Permission is hereby granted, free of charge, to any person obtaining a copy
 *  of this software and associated documentation files (the "Software"), to deal
 *  in the Software without restriction, including without limitation the rights
 *  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 *  copies of the Software, and to permit persons to whom the Software is
 *  furnished to do so, subject to the following conditions:
 *
 *  The above copyright notice and this permission notice shall be included in
 *  all copies or substantial portions of the Software.
 *
 *  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 *  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 *  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 *  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 *  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 *  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 *  THE SOFTWARE.
 *
 */

#ifndef _BRAINBOARD_H_
#define _BRAINBOARD_H_

#include <compiler.h>

#define BRAINBOARD_REV                  2

/* For the below values, clocks are derived from the uC's digital frequency-locked loop (DFLL),
 * divided by various clock dividers */
#define FCPU_HZ                         (DFLL_FREQ_HZ >> CONFIG_SYSCLK_CPU_DIV) // CPU frequency in Hz
#define FPBA_HZ                         (DFLL_FREQ_HZ >> CONFIG_SYSCLK_PBA_DIV) // Peripheral Bus A freq in Hz
#define FPBB_HZ                         (DFLL_FREQ_HZ >> CONFIG_SYSCLK_PBB_DIV) // Peripheral Bus B freq in Hz

/*****************************************************************************************************************
 *              Host Pin Config Definitions                                                                      *
 *****************************************************************************************************************/

#if (BRAINBOARD_REV == 0)

/**
 *  \brief Defines the host processor pin assigned to the RESET pin on the ADS1299.
 */
#define ADS1299_PIN_RESET       AVR32_PIN_PB04

/**
```

```
62  *  \brief Defines the host processor pin assigned to the START pin on the ADS1299.
63  */
64 #define ADS1299_PIN_START       AVR32_PIN_PA17
65
66 #endif
67
68 /**
69  *  \brief Defines the host processor pin assigned to the DRDY pin on the ADS1299.
70  */
71 #define ADS1299_PIN_DRDY        AVR32_PIN_PA18
72 #define ADS1299_DRDY_INT        5
73 #define ADS1299_DRDY_PRIORITY   AVR32_INTC_INT3
74 #define ADS1299_DRDY_IRQ        AVR32_EIC_IRQ_5
75 #define ADS1299_DRDY_INT_FN     AVR32_EIC_EXTINT_5_0_FUNCTION
76
77 /**
78  *  \brief Defines the host processor pin assigned to the PWDN pin on the ADS1299.
79  */
80 #define ADS1299_PIN_PWDN        AVR32_PIN_PA09
81
82 /**
83  *  \brief Defines the host processor pin assigned to the RESET pin on the RN-42.
84  */
85 #define BT_RESET_PIN            AVR32_PIN_PB11
86
87 /**
88  *  \brief Defines the host processor pin assigned to the INT pin on the MPU-6050.
89  */
90 #if (BRAINBOARD_REV == 2)
91 #define MPU6050_INT_PIN         AVR32_PIN_PA07
92 #elif (BRAINBOARD_REV < 2)
93 #define MPU6050_INT_PIN         AVR32_PIN_PA10
94 #endif
95
96 #endif /* #ifndef _BRAINBOARD_H_ */
```

```
/**
 *  \file comms.h
 *  \brief Defines channel counts, communication packets, and data rates for Brainboard.
 *  \author Graham Kelly
 *  \version 1.0
 *  \date August 2014
 *
 *  All Atmel Software Framework libraries used herein are copyright Atmel and
 *  subject to their appropriate licenses, which allow free redistribution with
 *  some restrictions. These restrictions are listed in their appropriate files.
 *
 *  \page License
 *
 *  Brainboard firmware code is placed under the MIT license
 *  Copyright (c) 2014 Graham Kelly
 *
 *  Permission is hereby granted, free of charge, to any person obtaining a copy
 *  of this software and associated documentation files (the "Software"), to deal
 *  in the Software without restriction, including without limitation the rights
 *  to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
 *  copies of the Software, and to permit persons to whom the Software is
 *  furnished to do so, subject to the following conditions:
 *
 *  The above copyright notice and this permission notice shall be included in
 *  all copies or substantial portions of the Software.
 *
 *  THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
 *  IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
 *  FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE
 *  AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER
 *  LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM,
 *  OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN
 *  THE SOFTWARE.
 *
 */
 */

#ifndef _COMMS_H_
#define _COMMS_H_

#include <compiler.h>

#ifdef __cplusplus
extern "C" {
#endif

#define SPI_ADDRESS                     (&AVR32_SPI)
#define DUMMY_BYTE                      0x00

#define SPI_ADS1299_MAIN_CHIPNUM        0
#define SPI_ADS1299_DAISY_CHIPNUM       2

/* For primary USART communicating with RN-42 Bluetooth module */
#define USARTBT_MODULE                  (&AVR32_USART3)
#define USARTBT_DEFAULT_BAUDRATE        115200
/* For secondary USART communicating with Beaglebone (if applicable; R0 and R1 only) */
#define USARTUC_MODULE                  (&AVR32_USART2)
#define USARTUC_BAUDRATE                921600
#define I2C_MODULE                      (&AVR32_TWIM0)

/* Define which USART is used for data streaming and which is used for auxiliary communications
 * (R0 and R1 only)
 */
#define DATA_USART                      USARTBT_MODULE
//#define DATA_USART                    USARTUC_MODULE
#define AUX_USART                       USARTBT_MODULE
```

```
67 //#define AUX_USART                     USARTUC_MODULE
68
69 /* DMA/PDCA definitions */
70 #define DATA_USART_TX_PDCA_CHANNEL      0
71 #define DATA_USART_TX_PDCA_PID          AVR32_PDCA_PID_USART3_TX
72 #define DATA_USART_RX_PDCA_CHANNEL      1
73 #define DATA_USART_RX_PDCA_PID          AVR32_PDCA_PID_USART3_RX
74
75 /**
76  *  \brief Defines the size of the signal portion of transmitted data packets.
77  */
78 #define DATA_USART_TX_BUFFER_SIZE           1
79
80 /**
81  *  \brief Defines the maximum number of EEG input channels available.
82  *
83  *  The maximum number of channels for a single ADS1299 is 8. MAX_CHANNELS may be decreased if this
       many channels
84  *  are not desired. Increasing the number to support multiple ADCs is not currently supported.
85  */
86 #define MAX_EEG_CHANNELS            8
87
88 /**
89  *  \brief Defines the maximum number of degrees of freedom from the inertial measurement chip.
90  *
91  *  The maximum number of channels supported for a single chip is 6 (3 acceleration and 3 rotation).
92  *  Increasing the number to support multiple IMUs is not currently supported.
93  */
94 #define MAX_IMU_DOF                 6
95
96 /**
97  *  \brief PADDED Brainboard packet structure for communicating 24-bit EEG and inertial data.
98  *
99  *  This packet type pre-pads 24-bit data to 32 bits. Easy to work with but least efficient for
       transmission.
100  */
101 typedef struct
102 {
103     uint16_t start;
104     uint8_t packetnum;
105     volatile uint32_t eegstatus;
106     volatile int32_t eegdata[DATA_USART_TX_BUFFER_SIZE][MAX_EEG_CHANNELS];
107     volatile int16_t imudata[DATA_USART_TX_BUFFER_SIZE][MAX_IMU_DOF];
108 } __attribute__((packed)) bboard_data32bit_packet_t;
109
110 /**
111  *  \brief PACKED Brainboard packet structure for communicating 24-bit EEG and inertial data.
112  *
113  *  This packet type stores EEG data as raw bytes. Most compact.
114  */
115 typedef struct
116 {
117     uint16_t start;
118     uint8_t packetnum;
119     volatile uint8_t eegstatus[3];
120     volatile uint8_t eegdata[DATA_USART_TX_BUFFER_SIZE][MAX_EEG_CHANNELS][3];
121     volatile int16_t imudata[DATA_USART_TX_BUFFER_SIZE][MAX_IMU_DOF];
122 } __attribute__((packed)) bboard_data24bit_packet_t;
123
124 /**
125  *  \brief Brainboard packet structure for communicating 16-bit EEG and inertial data.
126  *
127  *  This packet type truncates the EEG signal data, which trades packet size for precision. May lose
       information this way.
128  */
129 typedef struct
```

```
130 {
131     uint16_t start;
132     uint8_t packetnum;
133     volatile uint32_t eegstatus;
134     volatile int16_t eegdata[DATA_USART_TX_BUFFER_SIZE][MAX_EEG_CHANNELS];
135     volatile int16_t imudata[DATA_USART_TX_BUFFER_SIZE][MAX_IMU_DOF];
136 } __attribute__((packed)) bboard_data16bit_packet_t;
137
138 // #define data_packet_t                    bboard_data32bit_packet_t
139 // #define TX_PACKET_SIZE   (7+DATA_USART_TX_BUFFER_SIZE*(4*MAX_EEG_CHANNELS+2*MAX_IMU_DOF))
140
141 #define data_packet_t               bboard_data24bit_packet_t
142 #define TX_PACKET_SIZE  (6+DATA_USART_TX_BUFFER_SIZE*(3*MAX_EEG_CHANNELS+2*MAX_IMU_DOF))
143
144 // #define data_packet_t                bboard_data16bit_packet_t
145 // #define TX_PACKET_SIZE   (7+DATA_USART_TX_BUFFER_SIZE*(2*MAX_EEG_CHANNELS+2*MAX_IMU_DOF))
146
147
148 /**
149  * \brief Compile-time assertion macros for checking packet struct sizes.
150  *
151  *        ct_assert will give a negative array size error if it evaluates false.
152  */
153 #define ASSERT_CONCAT_(a, b) a##b
154 #define ASSERT_CONCAT(a, b) ASSERT_CONCAT_(a, b)
155 #define ct_assert(e) extern char (*ct_assert(void)) [sizeof(char[1 - 2*!(e)])]
156
157 /* If this line gives a compile-time error, byte alignment/packing differs from the intended
158  * implementation on the selected platform and may cause runtime errors.
159  */
160 ct_assert(sizeof(data_packet_t) == TX_PACKET_SIZE);
161
162 #ifdef __cplusplus
163 }
164 #endif
165
166 #endif /* #ifndef _COMMS_H_ */
```

Appendix D

BCI2VR Data Acquisition Source Code (MATLAB)

Brainboard Setup File:

```
function out=brainBoardBt_btv_daq_settings
% Brain-Computer Interface to Virtual reality (BCI2VR)
% Example Setup for EEG amplifier using ADS1299 and MPU6050 accelerometer/gyro via Bluetooth
% Ver. 1.0 02/25/2013 Copyright by Ou Bai
% You may add new variables, but you should not change the current variable names.
out.subject.Name                    =   'BCI2VR';
out.subject.DateOfBirth             =   '01-Jan-2014';
out.subject.Sex                     =   'M';
out.subject.Handness                =   'Right';
out.subject.Investigator            =   'BCI2VR Boy';
out.subject.Physician               =   'VCU';
out.subject.DateOfRecording         =   date;
out.daq.HardwareName                =   'brainBoardBt';                                 %% For
ads1299-based device only
out.daq.DeviceSerials               =   1;                                              %% how
many serial ports (currently only 1 is supported)
out.daq.Ports{1}                    =   1:14;                                           %% Check
firmware: 8 for one ADS1299, 6 for one MPU6050
out.daq.Channels{1}                 =   1:length(out.daq.Ports{1});                     %% Signal
channels
out.daq.UsedChannels{1}             =   logical([1 1 1 1 1 1 1 1 1 1 1 1 1 1]);         %% Used
signal channels for each device: default 1; 0 will be turned off
out.daq.ChannelGain{1}              =   24*ones(1,length(out.daq.Ports{1}));            %% Gain
for each ads1299 channel: 1,2,4,6,8,12,24(default)
out.daq.ElectrodeMontage            =   'differential';                                 %%
'singleEnded' or 'Differential'
out.daq.bbBt.RemoteName             =   'RNBT-B312';                                %% Bluetooth
device name, leave it empty if don't know 'RNBT-2427'
out.daq.bbBt.RemoteID               =   '0006666AB312';                                 %% The
Mac no. CHeck your Bluetooth device; If you know the Bluetooth device name, you may leave this
empty '000666662427'
out.daq.bbBt.BtChannel              =   1;                                              %% You
may use matlab function  instrhwinfo to check this
out.daq.bbBt.InputBufferSize        =   4096;                                           %% must
be larger than PacketLen * daq.SamplesAcquiredFcnCount
out.daq.bbBt.Headers                =   [165 90];                                       %% Check
firmwire: ads device has 3 status types starting with the two headers and one byte of packet
index
out.daq.bbBt.PacketLen              =   42;                                             %% Check
firmwire: 2 header bytes + 1 packet Number + 3 status bytes + 8 (eegChannel) * 3 bytes (24 bits)
+ 6 (ax, ay, az, gx, gy, gz) * 2 bytes
out.daq.bbBt.bitResolution          =   [(4.5/(2^23-1))*(ones(1,8)/24) 3.3/(2^15-1)*ones(1,6)];
%% V_REF/(Analog_gain*(2^(ADC_bits-1)-1)) for each channel
out.chan.locs                       =   btv_readlocs( 'brainBoardBt_14Ch.loc' );        %% Must
be a loc file
```

```
out.chan.Calibration                =   ones(14,1);                                 %% Always
1 for open EEG interface (Hardware has adjusted channel difference), may change in the future
out.daq.SamplingRate                =   250;                                        %%
ads1299 sampling at 250Hz; only 250 Hz is supported
out.daq.SamplesAcquiredFcnCount     =   20;                                         %% Buffer
update when 2000/125 (ads Sampling Rate/minmal sampling rate output) bytes available
out.buf.BufferLength                =   5000;                                       %% Must
be larger than 10*max(xscale)*SamplingRate, a large number will significant affect the processing
speed
out.buf.cntDataLength               =   60*60*out.daq.SamplingRate;                 %%
Reserve space for save continuous data
out.view.UpdateTimeInterval         =   0.5;                                        %% in
second, too short may cause delay for online calculation
out.view.LineColor                  =   ['m';'r';'b';'k'];                          %% 'k'
will be black color for all channels
out.view.Channels                   =   1:14;                                       %%
Combine channels in multiple amplifiers, must be in avaible signal channels
out.view.XScaleUnit                 =   1;                                          %%
Horizontal view: initial view range
out.view.xscale                     =   [0.01:0.01:0.09 0.1:0.1:0.9 1.0:1.0:5.0];   %%
Horizontal view: adjustable range for offline use only
out.view.YUnit                      =   [10^6*ones(1,8) 10^2*ones(1,6)];            %%
Convert to uV, length MUST be the same as view.Channels
out.view.ChannelYAxis               =   200;                                        %% Set Y
axis range
out.trigger.DetectTimeInterval      =   0.1;                                        %%
Detection Interval for Triggers from Stim/VR/Daq
out.daq.NotchFilter.bandStopFreq      = [];                                         %%
bandstop Frequency range (frequency component removal); [] for none.
out.daq.NotchFilter.order             = 2;                                          %%
Butterworth filter order
out.daq.LowpassFilter.Freq            = [];                                         %%
Lowpass frequency; [] for none
out.daq.LowpassFilter.order           = 2;                                          %%
Butterworth filter order;
out.daq.HighpassFilter.Freq           = [];                                         %% High
pass frequency; [] for none.
out.daq.HighpassFilter.order          = 2;                                          %%
Butterworth filter order
return
```

Published with MATLAB® R2013a

Brainboard Control Functions:

```
function btv_brainBoardBtFun(varargin)
% Brain-Computer Interface to Virtual reality (BCI2VR)
% Functions for interfacing with Brainboard LW
% (ADS1299 + MPU6050 Bluetooth device)
% Ver. 1.1 12-18-2013 Copyright by Ou Bai

global btv;
if btv.flag.daq~=1,
    errordlg('You must open DAQ interface before initializing data acquisition objects.');
    return
end

if nargin == 0,
elseif ischar(varargin{1}) % INVOKE NAMED SUBFUNCTION OR CALLBACK
    %     try
    eval([ 'feval(' varargin{:} ');']);
    %     catch
    %         errordlg(lasterr,['BCI2VR: ' mfilename]);
    %         daq_stop;
    %     end
end
return
```

GUI and ADS1299 initialization

```
function deviceInit %#ok<DEFNU>
global btv;
[~,~,endian] = computer;
btv.daq.settings.daq.endian = endian;
```

ADS1299 constants list:

```
btv.daq.settings.daq.adsOpc.ID = uint8(hex2dec('00'));     %% Device ID
btv.daq.settings.daq.adsOpc.RESET = uint8(hex2dec('06'));
btv.daq.settings.daq.adsOpc.START = uint8(hex2dec('08'));
btv.daq.settings.daq.adsOpc.STOP = uint8(hex2dec('0a'));
btv.daq.settings.daq.adsOpc.SDATAC = uint8(hex2dec('11'));
btv.daq.settings.daq.adsOpc.RDATAC = uint8(hex2dec('10'));
% CONFIG1 register
btv.daq.settings.daq.adsOpc.CONFIG1 = uint8(hex2dec('01'));
btv.daq.settings.daq.adsOpc.CONFIG1_const = uint8(hex2dec('90')); %a
btv.daq.settings.daq.adsOpc.DAISY_EN = uint8(hex2dec('40')); %a
btv.daq.settings.daq.adsOpc.CLK_EN = uint8(hex2dec('20')); %a
btv.daq.settings.daq.adsOpc.DR2 = uint8(hex2dec('04'));
btv.daq.settings.daq.adsOpc.DR1 = uint8(hex2dec('02'));
btv.daq.settings.daq.adsOpc.DR0 = uint8(hex2dec('01'));
btv.daq.settings.daq.adsOpc.RES_16k_SPS = uint8(btv.daq.settings.daq.adsOpc.CONFIG1_const); %a
```

```
btv.daq.settings.daq.adsOpc.RES_8k_SPS = uint8(btv.daq.settings.daq.adsOpc.CONFIG1_const+
btv.daq.settings.daq.adsOpc.DR0); %a
btv.daq.settings.daq.adsOpc.RES_4k_SPS = uint8(btv.daq.settings.daq.adsOpc.CONFIG1_const+
btv.daq.settings.daq.adsOpc.DR1); %a
btv.daq.settings.daq.adsOpc.RES_2k_SPS = uint8(btv.daq.settings.daq.adsOpc.CONFIG1_const+
btv.daq.settings.daq.adsOpc.DR1+ btv.daq.settings.daq.adsOpc.DR0); %a
btv.daq.settings.daq.adsOpc.RES_1k_SPS = uint8(btv.daq.settings.daq.adsOpc.CONFIG1_const+
btv.daq.settings.daq.adsOpc.DR2); %a
btv.daq.settings.daq.adsOpc.RES_500_SPS = uint8(btv.daq.settings.daq.adsOpc.CONFIG1_const+
btv.daq.settings.daq.adsOpc.DR2+ btv.daq.settings.daq.adsOpc.DR0); %a
btv.daq.settings.daq.adsOpc.RES_250_SPS = uint8(btv.daq.settings.daq.adsOpc.CONFIG1_const+
btv.daq.settings.daq.adsOpc.DR2+ btv.daq.settings.daq.adsOpc.DR1); %a
% CONFIG2 register
btv.daq.settings.daq.adsOpc.CONFIG2 = uint8(hex2dec('02'));
btv.daq.settings.daq.adsOpc.CONFIG_const = uint8(hex2dec('C0'));
btv.daq.settings.daq.adsOpc.INT_TEST = uint8(hex2dec('10')); %a
btv.daq.settings.daq.adsOpc.TEST_AMP = uint8(hex2dec('04')); %double amplitude of test signal to
2 x -(VREFP-VREFN)/2.4 mV
btv.daq.settings.daq.adsOpc.TEST_FREQ1 = uint8(hex2dec('02'));
btv.daq.settings.daq.adsOpc.TEST_FREQ0 = uint8(hex2dec('01'));
btv.daq.settings.daq.adsOpc.INT_TEST_4HZ_AMP = uint8(btv.daq.settings.daq.adsOpc.CONFIG_const +
btv.daq.settings.daq.adsOpc.INT_TEST + btv.daq.settings.daq.adsOpc.TEST_AMP); %Pulsed at 1Hz
(0.98) %a
btv.daq.settings.daq.adsOpc.INT_TEST_8HZ_AMP = uint8(btv.daq.settings.daq.adsOpc.CONFIG_const +
btv.daq.settings.daq.adsOpc.INT_TEST + btv.daq.settings.daq.adsOpc.TEST_FREQ1 +
btv.daq.settings.daq.adsOpc.TEST_AMP);  %Pulsed at 2 Hz (1.95) %a
btv.daq.settings.daq.adsOpc.INT_TEST_DC_AMP = uint8(btv.daq.settings.daq.adsOpc.CONFIG_const +
btv.daq.settings.daq.adsOpc.INT_TEST + + btv.daq.settings.daq.adsOpc.TEST_FREQ1 +
btv.daq.settings.daq.adsOpc.TEST_FREQ0 + btv.daq.settings.daq.adsOpc.TEST_AMP);  %Pulsed at 2 Hz
(1.95) %a
% CONFIG3 register
btv.daq.settings.daq.adsOpc.CONFIG3 = uint8(hex2dec('03'));
btv.daq.settings.daq.adsOpc.CONFIG3_const = uint8(hex2dec('60'));
btv.daq.settings.daq.adsOpc.PD_REFBUF = uint8(hex2dec('80'));
btv.daq.settings.daq.adsOpc.BIASREF_INT = uint8(hex2dec('08'));
btv.daq.settings.daq.adsOpc.PD_BIAS = uint8(hex2dec('04'));
btv.daq.settings.daq.adsOpc.BIAS_LOFF_SENS = uint8(hex2dec('02'));
btv.daq.settings.daq.adsOpc.EXT_REF_EXT_BIAS = uint8(btv.daq.settings.daq.adsOpc.CONFIG3_const);
% LOFF (lead-off) register
btv.daq.settings.daq.adsOpc.CONFIGLEADOFF = uint8(hex2dec('04'));
btv.daq.settings.daq.adsOpc.LEADOFF_const = uint8(hex2dec('00'));
btv.daq.settings.daq.adsOpc.COMP_TH2 = uint8(hex2dec('80'));
btv.daq.settings.daq.adsOpc.COMP_TH1 = uint8(hex2dec('40'));
btv.daq.settings.daq.adsOpc.COMP_TH0 = uint8(hex2dec('20'));
btv.daq.settings.daq.adsOpc.VLEAD_OFF_EN = uint8(hex2dec('10'));
btv.daq.settings.daq.adsOpc.ILEAD_OFF1 = uint8(hex2dec('08'));
btv.daq.settings.daq.adsOpc.ILEAD_OFF0 = uint8(hex2dec('04'));
btv.daq.settings.daq.adsOpc.FLEAD_OFF1 = uint8(hex2dec('02'));
btv.daq.settings.daq.adsOpc.FLEAD_OFF0 = uint8(hex2dec('01'));
btv.daq.settings.daq.adsOpc.COMP_TH_95 = uint8(hex2dec('00'));
btv.daq.settings.daq.adsOpc.COMP_TH_92_5 = btv.daq.settings.daq.adsOpc.COMP_TH0;
btv.daq.settings.daq.adsOpc.COMP_TH_90 = btv.daq.settings.daq.adsOpc.COMP_TH1;
btv.daq.settings.daq.adsOpc.COMP_TH_87_5 = (btv.daq.settings.daq.adsOpc.COMP_TH1 +
```

```
btv.daq.settings.daq.adsOpc.COMP_TH0);
btv.daq.settings.daq.adsOpc.COMP_TH_85 = btv.daq.settings.daq.adsOpc.COMP_TH2;
btv.daq.settings.daq.adsOpc.COMP_TH_80 = (btv.daq.settings.daq.adsOpc.COMP_TH2 +
btv.daq.settings.daq.adsOpc.COMP_TH0);
btv.daq.settings.daq.adsOpc.COMP_TH_75 = (btv.daq.settings.daq.adsOpc.COMP_TH2 +
btv.daq.settings.daq.adsOpc.COMP_TH1);
btv.daq.settings.daq.adsOpc.COMP_TH_70 = (btv.daq.settings.daq.adsOpc.COMP_TH2 +
btv.daq.settings.daq.adsOpc.COMP_TH1 + btv.daq.settings.daq.adsOpc.COMP_TH0);
btv.daq.settings.daq.adsOpc.ILEAD_OFF_6nA = uint8(hex2dec('00'));
btv.daq.settings.daq.adsOpc.ILEAD_OFF_12nA = btv.daq.settings.daq.adsOpc.ILEAD_OFF0;
btv.daq.settings.daq.adsOpc.ILEAD_OFF_18nA = btv.daq.settings.daq.adsOpc.ILEAD_OFF1;
btv.daq.settings.daq.adsOpc.ILEAD_OFF_24nA = (btv.daq.settings.daq.adsOpc.ILEAD_OFF1 +
btv.daq.settings.daq.adsOpc.ILEAD_OFF0);
btv.daq.settings.daq.adsOpc.FLEAD_OFF_AC = btv.daq.settings.daq.adsOpc.FLEAD_OFF0;
btv.daq.settings.daq.adsOpc.FLEAD_OFF_DC = (btv.daq.settings.daq.adsOpc.FLEAD_OFF1 +
btv.daq.settings.daq.adsOpc.FLEAD_OFF0);
% CHnSET individual channel settings (n = 1:8)
% Address = 05h to 0Ch
btv.daq.settings.daq.adsOpc.CONFIGCHnSET = uint8(hex2dec('05'));
btv.daq.settings.daq.adsOpc.PDn = uint8(hex2dec('80')); %power down amplifier
btv.daq.settings.daq.adsOpc.GAINn2 = uint8(hex2dec('40'));
btv.daq.settings.daq.adsOpc.GAINn1 = uint8(hex2dec('20'));
btv.daq.settings.daq.adsOpc.GAINn0 = uint8(hex2dec('10'));
btv.daq.settings.daq.adsOpc.GAIN_1X = uint8(hex2dec('00'));
btv.daq.settings.daq.adsOpc.GAIN_2X = uint8(btv.daq.settings.daq.adsOpc.GAINn0);
btv.daq.settings.daq.adsOpc.GAIN_4X = uint8(btv.daq.settings.daq.adsOpc.GAINn1);
btv.daq.settings.daq.adsOpc.GAIN_6X = uint8(btv.daq.settings.daq.adsOpc.GAINn1+
btv.daq.settings.daq.adsOpc.GAINn0);
btv.daq.settings.daq.adsOpc.GAIN_8X = uint8(btv.daq.settings.daq.adsOpc.GAINn2);
btv.daq.settings.daq.adsOpc.GAIN_12X = uint8(btv.daq.settings.daq.adsOpc.GAINn2+
btv.daq.settings.daq.adsOpc.GAINn0);
btv.daq.settings.daq.adsOpc.GAIN_24X = uint8(btv.daq.settings.daq.adsOpc.GAINn2+
btv.daq.settings.daq.adsOpc.GAINn1);
btv.daq.settings.daq.adsOpc.MUXn2 = uint8(hex2dec('04'));
btv.daq.settings.daq.adsOpc.MUXn1 = uint8(hex2dec('02'));
btv.daq.settings.daq.adsOpc.MUXn0 = uint8(hex2dec('01'));
btv.daq.settings.daq.adsOpc.TEST_SIGNAL = uint8(btv.daq.settings.daq.adsOpc.MUXn2+
btv.daq.settings.daq.adsOpc.MUXn0);
btv.daq.settings.daq.adsOpc.ELECTRODE_INPUT = uint8(hex2dec('00'));
btv.daq.settings.daq.adsOpc.SHORTED = uint8(btv.daq.settings.daq.adsOpc.MUXn0);
% BIAS_SENSP: Bias Drive Positive Sense Selection
btv.daq.settings.daq.adsOpc.CONFIGBIAS_SENSP = uint8(hex2dec('0D'));
btv.daq.settings.daq.adsOpc.BIAS_SENSP_const = uint8(hex2dec('00'));
btv.daq.settings.daq.adsOpc.BIAS8P = uint8(hex2dec('80'));
btv.daq.settings.daq.adsOpc.BIAS7P = uint8(hex2dec('40'));
btv.daq.settings.daq.adsOpc.BIAS6P = uint8(hex2dec('20'));
btv.daq.settings.daq.adsOpc.BIAS5P = uint8(hex2dec('10'));
btv.daq.settings.daq.adsOpc.BIAS4P = uint8(hex2dec('08'));
btv.daq.settings.daq.adsOpc.BIAS3P = uint8(hex2dec('04'));
btv.daq.settings.daq.adsOpc.BIAS2P = uint8(hex2dec('02'));
btv.daq.settings.daq.adsOpc.BIAS1P = uint8(hex2dec('01'));
% BIAS_SENSN: Bias Drive Negative Sense Selection
btv.daq.settings.daq.adsOpc.CONFIGBIAS_SENSN = uint8(hex2dec('0E'));
```

```
btv.daq.settings.daq.adsOpc.BIAS_SENSN_const = uint8(hex2dec('00'));
btv.daq.settings.daq.adsOpc.BIAS8N = uint8(hex2dec('80'));
btv.daq.settings.daq.adsOpc.BIAS7N = uint8(hex2dec('40'));
btv.daq.settings.daq.adsOpc.BIAS6N = uint8(hex2dec('20'));
btv.daq.settings.daq.adsOpc.BIAS5N = uint8(hex2dec('10'));
btv.daq.settings.daq.adsOpc.BIAS4N = uint8(hex2dec('08'));
btv.daq.settings.daq.adsOpc.BIAS3N = uint8(hex2dec('04'));
btv.daq.settings.daq.adsOpc.BIAS2N = uint8(hex2dec('02'));
btv.daq.settings.daq.adsOpc.BIAS1N = uint8(hex2dec('01'));
% LOFF_SENSP: Lead Off Positive Sense Selection
btv.daq.settings.daq.adsOpc.CONFIGLOFF_SENSP = uint8(hex2dec('0F'));
btv.daq.settings.daq.adsOpc.LOFF_SENSP_const = uint8(hex2dec('00'));
btv.daq.settings.daq.adsOpc.LOFF8P = uint8(hex2dec('80'));
btv.daq.settings.daq.adsOpc.LOFF7P = uint8(hex2dec('40'));
btv.daq.settings.daq.adsOpc.LOFF6P = uint8(hex2dec('20'));
btv.daq.settings.daq.adsOpc.LOFF5P = uint8(hex2dec('10'));
btv.daq.settings.daq.adsOpc.LOFF4P = uint8(hex2dec('08'));
btv.daq.settings.daq.adsOpc.LOFF3P = uint8(hex2dec('04'));
btv.daq.settings.daq.adsOpc.LOFF2P = uint8(hex2dec('02'));
btv.daq.settings.daq.adsOpc.LOFF1P = uint8(hex2dec('01'));
% LOFF_SENSN: Lead Off Negative Sense Selection
btv.daq.settings.daq.adsOpc.CONFIGLOFF_SENSN = uint8(hex2dec('10'));
btv.daq.settings.daq.adsOpc.LOFF_SENSN_const = uint8(hex2dec('00'));
btv.daq.settings.daq.adsOpc.LOFF8N = uint8(hex2dec('80'));
btv.daq.settings.daq.adsOpc.LOFF7N = uint8(hex2dec('40'));
btv.daq.settings.daq.adsOpc.LOFF6N = uint8(hex2dec('20'));
btv.daq.settings.daq.adsOpc.LOFF5N = uint8(hex2dec('10'));
btv.daq.settings.daq.adsOpc.LOFF4N = uint8(hex2dec('08'));
btv.daq.settings.daq.adsOpc.LOFF3N = uint8(hex2dec('04'));
btv.daq.settings.daq.adsOpc.LOFF2N = uint8(hex2dec('02'));
btv.daq.settings.daq.adsOpc.LOFF1N = uint8(hex2dec('01'));
% LOFF_FLIP: Lead Off Current Direction Control
btv.daq.settings.daq.adsOpc.CONFIGLOFF_FLIP = uint8(hex2dec('11'));
btv.daq.settings.daq.adsOpc.LOFF_FLIP_const = uint8(hex2dec('00'));
btv.daq.settings.daq.adsOpc.LOFF_FLIP8 = uint8(hex2dec('80'));
btv.daq.settings.daq.adsOpc.LOFF_FLIP7 = uint8(hex2dec('40'));
btv.daq.settings.daq.adsOpc.LOFF_FLIP6 = uint8(hex2dec('20'));
btv.daq.settings.daq.adsOpc.LOFF_FLIP5 = uint8(hex2dec('10'));
btv.daq.settings.daq.adsOpc.LOFF_FLIP4 = uint8(hex2dec('08'));
btv.daq.settings.daq.adsOpc.LOFF_FLIP3 = uint8(hex2dec('04'));
btv.daq.settings.daq.adsOpc.LOFF_FLIP2 = uint8(hex2dec('02'));
btv.daq.settings.daq.adsOpc.LOFF_FLIP1 = uint8(hex2dec('01'));
% LOFF_STATP: Lead-Off Positive Input Status (read only)
btv.daq.settings.daq.adsOpc.CONFIGLOFF_STATP = uint8(hex2dec('12'));
% LOFF_STATN: Lead-Off Negative Input Status (read only)
btv.daq.settings.daq.adsOpc.CONFIGLOFF_STATN = uint8(hex2dec('13'));
% GPIO: General-Purpose I/O Register
btv.daq.settings.daq.adsOpc.CONFIGGPIO = uint8(hex2dec('14'));   %% For GPIO settting
btv.daq.settings.daq.adsOpc.GPIO_const = uint8(hex2dec('00'));
btv.daq.settings.daq.adsOpc.GPIOD4 = uint8(hex2dec('80'));
btv.daq.settings.daq.adsOpc.GPIOD3 = uint8(hex2dec('40'));
btv.daq.settings.daq.adsOpc.GPIOD2 = uint8(hex2dec('20'));
btv.daq.settings.daq.adsOpc.GPIOD1 = uint8(hex2dec('10'));
```

```
btv.daq.settings.daq.adsOpc.GPIOC4 = uint8(hex2dec('08'));
btv.daq.settings.daq.adsOpc.GPIOC3 = uint8(hex2dec('04'));
btv.daq.settings.daq.adsOpc.GPIOC2 = uint8(hex2dec('02'));
btv.daq.settings.daq.adsOpc.GPIOC1 = uint8(hex2dec('01'));
% MISC1: Miscellaneous 1
btv.daq.settings.daq.adsOpc.CONFIGMISC1 = uint8(hex2dec('15'));
btv.daq.settings.daq.adsOpc.MISC1_const = uint8(hex2dec('00'));
btv.daq.settings.daq.adsOpc.SRB1 = uint8(hex2dec('20'));
% MISC2: Miscellaneous 2 (must be 0)
btv.daq.settings.daq.adsOpc.CONFIGMISC2 = uint8(hex2dec('16'));
btv.daq.settings.daq.adsOpc.MISC2_const = uint8(hex2dec('00'));
% CONFIG4: Configuration Register 4
btv.daq.settings.daq.adsOpc.CONFIG4 = uint8(hex2dec('17'));
btv.daq.settings.daq.adsOpc.CONFIG4_const = uint8(hex2dec('00'));
btv.daq.settings.daq.adsOpc.SINGLE_SHOT = uint8(hex2dec('08'));
btv.daq.settings.daq.adsOpc.PD_LOFF_COMP = uint8(hex2dec('02'));
```

Create and open Bluetooth connection

```
instrVer=ver('instrument');
if ~isempty(instrVer)
    if str2double(instrVer.Version) < 3.0
        errordlg('The MATLAB Instrument Control Toolbox version must be 3.0 (MATLAB 2011b) or
later.');
        return
    end
else
    errordlg('The MATLAB Instrument Control Toolbox is required!');
    return
end

hdl=waitbar(0,'Connecting Bluetooth device, please wait...','Name','BCI2VR');
drawnow;

if ~isempty(btv.daq.settings.daq.bbBt.RemoteName)

btv.hdl.daq.bbBt=Bluetooth(btv.daq.settings.daq.bbBt.RemoteName,btv.daq.settings.daq.bbBt.BtChann
el);
elseif ~isempty(btv.daq.settings.daq.bbBt.RemoteID)
    btv.hdl.daq.bbBt=Bluetooth(['btspp://'
btv.daq.settings.daq.bbBt.RemoteID],btv.daq.settings.daq.bbBt.BtChannel);
end

waitbar(1/3);

if isempty(btv.hdl.daq.bbBt)
    close(hdl);
    error('The assigned device is NOT found!');
end
%
set(btv.hdl.daq.bbBt,'InputBufferSize',btv.daq.settings.daq.bbBt.InputBufferSize);
set(btv.hdl.daq.bbBt,'ReadAsyncMode','continuous'); % the serial port object continuously queries
```

```
the device to determine if data is available to be read
set(btv.hdl.daq.bbBt,'BytesAvailableFcnCount',btv.daq.settings.daq.SamplesAcquiredFcnCount*btv.da
q.settings.daq.bbBt.PacketLen);
set(btv.hdl.daq.bbBt,'BytesAvailableFcnMode','Byte','BytesAvailableFcn',[mfilename
'(''@btvBufUpdate'')']);

waitbar(2/3);

try
    fopen(btv.hdl.daq.bbBt);
    waitbar(3/3);
    close(hdl);
    hdl = waitbar(0,'Configuring data acquisition...','Name','BCI2VR');
catch errCode
    close(hdl);
    errordlg(errCode.message,'BCI2VR');
    return
end
while get(btv.hdl.daq.bbBt,'BytesAvailable') > 0, %Flush serial input buffer
    fread(btv.hdl.daq.bbBt,get(btv.hdl.daq.bbBt,'BytesAvailable'));
end
```

Set up ADS1299 device

```
deviceStop; %stop ADS1299 sampling
pause(0.1); %delay probably not required, but we need to be in SDATAC to change setup

adsWriteReg(btv.daq.settings.daq.adsOpc.CONFIGGPIO, uint8(hex2dec('00')));
waitbar(1/15);
% Register CONFIG1 sets sample rate, daisy-chain, resolution (high vs power saving) and CLK
connection
switch btv.daq.settings.daq.SamplingRate
%     case 16000
%
adsWriteReg(btv.daq.settings.daq.adsOpc.CONFIG1,btv.daq.settings.daq.adsOpc.RES_16k_SPS);
%     case 8000
%
adsWriteReg(btv.daq.settings.daq.adsOpc.CONFIG1,btv.daq.settings.daq.adsOpc.RES_8k_SPS);
%     case 4000
%
adsWriteReg(btv.daq.settings.daq.adsOpc.CONFIG1,btv.daq.settings.daq.adsOpc.RES_4k_SPS);
%     case 2000
%
adsWriteReg(btv.daq.settings.daq.adsOpc.CONFIG1,btv.daq.settings.daq.adsOpc.RES_2k_SPS);
%     case 1000
%
adsWriteReg(btv.daq.settings.daq.adsOpc.CONFIG1,btv.daq.settings.daq.adsOpc.RES_1k_SPS);
%     case 500
%
adsWriteReg(btv.daq.settings.daq.adsOpc.CONFIG1,btv.daq.settings.daq.adsOpc.RES_500_SPS);
    case 250
        adsWriteReg(btv.daq.settings.daq.adsOpc.CONFIG1,btv.daq.settings.daq.adsOpc.RES_250_SPS);
```

```
%Default for wired data acquisition
    otherwise
        close(hdl);
        errordlg('Currently only 250 Hz data rate is supported.');
        deviceClose;
        return
end
waitbar(2/15);
% Register CONFIG2 sets internal/external test signal, test amplifier and test frequency
adsWriteReg(btv.daq.settings.daq.adsOpc.CONFIG2,
uint8(btv.daq.settings.daq.adsOpc.INT_TEST_4HZ_AMP));  % generate internal test signals
waitbar(3/15);
% You would also specify which channels to use for BIAS
adsWriteReg(btv.daq.settings.daq.adsOpc.CONFIGBIAS_SENSP, uint8(hex2dec('FF')));  % Use all input
channels IN1P-IN8P and IN1N-IN8N
waitbar(4/15);
adsWriteReg(btv.daq.settings.daq.adsOpc.CONFIGBIAS_SENSN, uint8(hex2dec('FF')));
waitbar(5/15);
% Register CONFIG3 sets bias operation: Power up the internal bias reference and wait for it to
settle
adsWriteReg(btv.daq.settings.daq.adsOpc.CONFIG3,uint8(btv.daq.settings.daq.adsOpc.CONFIG3_const +
btv.daq.settings.daq.adsOpc.PD_REFBUF + btv.daq.settings.daq.adsOpc.BIASREF_INT +
btv.daq.settings.daq.adsOpc.PD_BIAS )); %PD_REFBUF used for test signal
waitbar(6/15);
% Set single-ended or differential operation
if strcmpi(btv.daq.settings.daq.ElectrodeMontage,'singleEnded')

adsWriteReg(btv.daq.settings.daq.adsOpc.CONFIGMISC1,uint8(btv.daq.settings.daq.adsOpc.MISC1_const
+ btv.daq.settings.daq.adsOpc.SRB1));
else

adsWriteReg(btv.daq.settings.daq.adsOpc.CONFIGMISC1,uint8(btv.daq.settings.daq.adsOpc.MISC1_const
));
end
waitbar(7/15);
% Channel registers
for i=1:8           % Only one ADS1299 device is supported and Ports is always 1:8
    switch btv.daq.settings.daq.ChannelGain{1}(i)
        case 1
            adsWriteReg(uint8(btv.daq.settings.daq.adsOpc.CONFIGCHnSET + i-1),
uint8(btv.daq.settings.daq.adsOpc.ELECTRODE_INPUT+ btv.daq.settings.daq.adsOpc.GAIN_1X));
        case 2
            adsWriteReg(uint8(btv.daq.settings.daq.adsOpc.CONFIGCHnSET + i-1),
uint8(btv.daq.settings.daq.adsOpc.ELECTRODE_INPUT+ btv.daq.settings.daq.adsOpc.GAIN_2X));
        case 4
            adsWriteReg(uint8(btv.daq.settings.daq.adsOpc.CONFIGCHnSET + i-1),
uint8(btv.daq.settings.daq.adsOpc.ELECTRODE_INPUT+ btv.daq.settings.daq.adsOpc.GAIN_4X));
        case 6
            adsWriteReg(uint8(btv.daq.settings.daq.adsOpc.CONFIGCHnSET + i-1),
uint8(btv.daq.settings.daq.adsOpc.ELECTRODE_INPUT+ btv.daq.settings.daq.adsOpc.GAIN_6X));
        case 8
            adsWriteReg(uint8(btv.daq.settings.daq.adsOpc.CONFIGCHnSET + i-1),
uint8(btv.daq.settings.daq.adsOpc.ELECTRODE_INPUT+ btv.daq.settings.daq.adsOpc.GAIN_8X));
```

```
        case 12
            adsWriteReg(uint8(btv.daq.settings.daq.adsOpc.CONFIGCHnSET + i-1),
uint8(btv.daq.settings.daq.adsOpc.ELECTRODE_INPUT+ btv.daq.settings.daq.adsOpc.GAIN_12X));
        case 24
            adsWriteReg(uint8(btv.daq.settings.daq.adsOpc.CONFIGCHnSET + i-1),
uint8(btv.daq.settings.daq.adsOpc.ELECTRODE_INPUT+ btv.daq.settings.daq.adsOpc.GAIN_24X));
        otherwise
            close(hdl);
            errordlg('ADS1299 programmable gain is one of 1, 2, 4, 6, 8, 12 and 24.');
            deviceClose;
            return
    end
    if btv.daq.settings.daq.UsedChannels{1}(i) == 0,  % %turn off unused channels
        adsWriteReg(uint8(btv.daq.settings.daq.adsOpc.CONFIGCHnSET + i-1),
uint8(btv.daq.settings.daq.adsOpc.PDn+ btv.daq.settings.daq.adsOpc.SHORTED));
    end
    waitbar((7+i)/15);
end
close(hdl);
hdl = waitbar(0,'Configuring filter parameters...','Name','BCI2VR');
```

Set up digital filter

```
btv.daq.buf.Filter.zeros=[];
btv.daq.buf.Filter.poles=[];
btv.daq.buf.Filter.gains=1;
numChan=0;
for i=1:length(btv.daq.settings.daq.DeviceSerials)
    numChan=numChan+length(btv.daq.settings.daq.Ports{i});
end
waitbar(1/5);
if isfield(btv.daq.settings.daq,'NotchFilter') &&
~isempty(btv.daq.settings.daq.NotchFilter.bandStopFreq)

[btv.daq.buf.NotchFilter.b,btv.daq.buf.NotchFilter.a]=butter(btv.daq.settings.daq.NotchFilter.ord
er,...

btv.daq.settings.daq.NotchFilter.bandStopFreq/(btv.daq.settings.daq.SamplingRate/2),'stop');

[btv.daq.buf.NotchFilter.zeros,btv.daq.buf.NotchFilter.poles,btv.daq.buf.NotchFilter.gain]=tf2zp(
btv.daq.buf.NotchFilter.b,btv.daq.buf.NotchFilter.a);
    btv.daq.buf.Filter.zeros=[btv.daq.buf.Filter.zeros;btv.daq.buf.NotchFilter.zeros];
    btv.daq.buf.Filter.poles=[btv.daq.buf.Filter.poles;btv.daq.buf.NotchFilter.poles];
    btv.daq.buf.Filter.gains=btv.daq.buf.Filter.gains*btv.daq.buf.NotchFilter.gain;
end
waitbar(2/5);
if isfield(btv.daq.settings.daq,'LowpassFilter') &&
~isempty(btv.daq.settings.daq.LowpassFilter.Freq)

[btv.daq.buf.LowpassFilter.b,btv.daq.buf.LowpassFilter.a]=butter(btv.daq.settings.daq.LowpassFilt
er.order,...
        btv.daq.settings.daq.LowpassFilter.Freq/(btv.daq.settings.daq.SamplingRate/2),'low');
```

```
[btv.daq.buf.LowpassFilter.zeros,btv.daq.buf.LowpassFilter.poles,btv.daq.buf.LowpassFilter.gain]=
tf2zp(btv.daq.buf.LowpassFilter.b,btv.daq.buf.LowpassFilter.a);
    btv.daq.buf.Filter.zeros=[btv.daq.buf.Filter.zeros;btv.daq.buf.LowpassFilter.zeros];
    btv.daq.buf.Filter.poles=[btv.daq.buf.Filter.poles;btv.daq.buf.LowpassFilter.poles];
    btv.daq.buf.Filter.gains=btv.daq.buf.Filter.gains*btv.daq.buf.LowpassFilter.gain;
end
waitbar(3/5);
if isfield(btv.daq.settings.daq,'NotchFilter') &&
~isempty(btv.daq.settings.daq.HighpassFilter.Freq)

[btv.daq.buf.HighpassFilter.b,btv.daq.buf.HighpassFilter.a]=butter(btv.daq.settings.daq.HighpassF
ilter.order,...
        btv.daq.settings.daq.HighpassFilter.Freq/(btv.daq.settings.daq.SamplingRate/2),'high');

[btv.daq.buf.HighpassFilter.zeros,btv.daq.buf.HighpassFilter.poles,btv.daq.buf.HighpassFilter.gai
n]=tf2zp(btv.daq.buf.HighpassFilter.b,btv.daq.buf.HighpassFilter.a);
    btv.daq.buf.Filter.zeros=[btv.daq.buf.Filter.zeros;btv.daq.buf.HighpassFilter.zeros];
    btv.daq.buf.Filter.poles=[btv.daq.buf.Filter.poles;btv.daq.buf.HighpassFilter.poles];
    btv.daq.buf.Filter.gains=btv.daq.buf.Filter.gains*btv.daq.buf.HighpassFilter.gain;
end
waitbar(4/5);
if ~isempty(btv.daq.buf.Filter.zeros)

[btv.daq.buf.Filter.b,btv.daq.buf.Filter.a]=zp2tf(btv.daq.buf.Filter.zeros,btv.daq.buf.Filter.pol
es,btv.daq.buf.Filter.gains);
    btv.daq.buf.Filter.Zf=zeros(max(length(btv.daq.buf.Filter.b),length(btv.daq.buf.Filter.a))-
1,numChan);
else
    btv.daq.buf.Filter=[];
end
```

Add toolbar on daq monitor

```
btv.hdl.daq.tb1.calibration=uitoggletool(btv.hdl.daq.toolbar1,'separator','on','CData',image2icon
data('calibration.gif'),'state','off',...
    'TooltipString','Calibration','ClickedCallback',[mfilename '(''@adsCalibration'')']);
btv.hdl.daq.tb1.checkImpedance=uipushtool(btv.hdl.daq.toolbar1,'CData',image2icondata('checkImped
ance.gif'),...
    'TooltipString','Check Impedance','ClickedCallback',[mfilename '(''@adsImpedancePanel'')']);
waitbar(1,hdl,'Configuration complete.');
pause(0.5);
close(hdl);
```

Start acquiring data

```
function deviceStart
    global btv;
    % Start data streaming
    pause(.5);
    btv.daq.buf.adsResidual=[];
```

```
    btv.daq.buf.packetNumberRead=[];
    adsSendCommand(btv.daq.settings.daq.adsOpc.RDATAC); %restart streaming
    adsSendCommand(btv.daq.settings.daq.adsOpc.START);
```

Update data buffer

```
function btvBufUpdate %#ok<DEFNU>   % Consider to move into btv_daq
global btv;
% Updating data buffer using FIFO
updatedData = adsStream;
if ~isempty(updatedData)
    btv.daq.buf.currentTimePoint=btv.daq.buf.currentTimePoint+size(updatedData,1); %% For using
getdata
    btv.daq.buf.dataTimePoints=(btv.daq.buf.currentTimePoint-
btv.daq.settings.buf.BufferLength+1:btv.daq.buf.currentTimePoint)';
    % Apply digital filters
    if isfield(btv.daq.buf,'Filter')
        if ~isempty(btv.daq.buf.Filter)
            [updatedData,btv.daq.buf.Filter.Zf] = filter(btv.daq.buf.Filter.b,
btv.daq.buf.Filter.a, updatedData, btv.daq.buf.Filter.Zf, 1);
        end
    end

    btv.daq.buf.data=[btv.daq.buf.data;updatedData];
    btv.daq.buf.data(1:end-btv.daq.settings.buf.BufferLength,:)=[];
    if btv.daq.flag.scopeon == 1, btv.daq.buf.viewUpdate=[btv.daq.buf.viewUpdate;updatedData];
end
    btv.daq.buf.socketRDA=updatedData;

    if btv.daq.flag.tcpip == 1,
        if btv.hdl.daq.conRDA == -1,
            btv.hdl.daq.conRDA=pnet(btv.hdl.daq.sockconRDA,'tcplisten');
            if btv.hdl.daq.conRDA~=-1
                pnet(btv.hdl.daq.conRDA,'setwritetimeout',.1);
                btv.daq.buf.nBlock=0;
            end
        end
        if btv.hdl.daq.conRDA ~= -1,
            if pnet(btv.hdl.daq.conRDA,'status') <= 0, %% Remote client is disconnected
                pnet(btv.hdl.daq.conRDA,'close');
                btv.hdl.daq.conRDA=-1;
            elseif ~isempty(btv.daq.buf.socketRDA),
                writeRDAPort;
            end
        end
    end
end
```

Begin recording data to disk

```
function deviceRecord %#ok<DEFNU>
```

```
global btv;
% Stop data streaming
deviceStop;
% Flush serial input buffer
while get(btv.hdl.daq.bbBt,'BytesAvailable') > 0, %there is often some latency with USB
communications
    fread(btv.hdl.daq.bbBt,get(btv.hdl.daq.bbBt,'BytesAvailable'));
    pause(0.01);
end
[tempname,temppath] = uiputfile({'*_bb.bin';'*_bb.txt'},'Save data to...');
if isequal(tempname,0) || isequal(temppath,0), return; end
[spath,sname,sext]=fileparts(fullfile(temppath,tempname));
btv.daq.settings.data.datasavingpathname=spath;
btv.daq.settings.data.datasavingext=sext;
aa=strfind(sname,'_bb');
if ~isempty(aa)
    btv.daq.settings.datasavingfilename=sname(1:aa-1);
else
    btv.daq.settings.datasavingfilename=sname;
end
btv.daq.settings.datasavingfilename=[btv.daq.settings.datasavingfilename '_bb'];
switch btv.daq.settings.data.datasavingext
   case '.bin'
       fid=fopen(fullfile(btv.daq.settings.data.datasavingpathname,...
            [btv.daq.settings.datasavingfilename
btv.daq.settings.data.datasavingext]),'w+','ieee-le');
   case '.txt'
       fid = fopen(fullfile(btv.daq.settings.data.datasavingpathname,...
            [btv.daq.settings.datasavingfilename btv.daq.settings.data.datasavingext]),'w+');
       fprintf(fid,'Recording started on %s\r\n',datestr(clock));
       fprintf(fid,'Device: %s\r\n',btv.daq.settings.daq.HardwareName);
       fprintf(fid,'Sampling Rate: %d Hz\r\n',btv.daq.settings.daq.SamplingRate);
       channels = btv.daq.settings.daq.Ports{1};
       usedChannels = channels(btv.daq.settings.daq.UsedChannels{1});
       fprintf(fid,'CH%d\t',usedChannels(1:end-1));
       fprintf(fid,'CH%d\r\n',usedChannels(end));
end
if fid < 0,
    errordlg('BCI2VR can''t open the file to save data');
    deviceClose;
    return
end
fclose(fid);
deviceStart;
```

Stop recording to disk

```
function deviceRecordOff %#ok<DEFNU>
global btv;
btv.daq.flag.recordon=0;
```

Stop acquiring data

```
function deviceStop
global btv;
% Stop data streaming
adsSendCommand(btv.daq.settings.daq.adsOpc.SDATAC);
adsSendCommand(btv.daq.settings.daq.adsOpc.STOP);

% Flush serial input buffer
while get(btv.hdl.daq.bbBt,'BytesAvailable') > 0,
    fread(btv.hdl.daq.bbBt,get(btv.hdl.daq.bbBt,'BytesAvailable'));

    pause(0.01);
end
```

Close Bluetooth connection

```
function deviceClose
global btv;
deviceStop;
pause(0.1);
fclose(btv.hdl.daq.bbBt);
delete(btv.hdl.daq.bbBt);
```

Send command to the ADS1299

```
function adsSendCommand(cmd)
global btv;
pause(0.1);
% MCU has 3-byte DMA buffer for USART RX; need to fill it to
% trigger an interrupt
fwrite(btv.hdl.daq.bbBt,[cmd 0 0]);
pause(0.1);
```

Write to ADS1299 register

```
function adsWriteReg(reg, val)
global btv;
pause(0.1);
fwrite(btv.hdl.daq.bbBt,[64+reg,0,val]);
pause(0.5);
readval = adsReadReg(reg);
if val ~= readval
    disp(['Wrote ' int2str(val) ' to register ' int2str(reg) ', read back ' int2str(readval)
'.']);
    errordlg('ADS1299 register read/write error! Please try again.');
    deviceClose;
    return
end
```

Read ADS1299 register

```
function [val] =  adsReadReg(reg)
global btv;
val = -1;
% flush buffer
while get(btv.hdl.daq.bbBt,'BytesAvailable') > 0
   fread(btv.hdl.daq.bbBt,get(btv.hdl.daq.bbBt,'BytesAvailable'));
   pause(0.01);
end
% MCU has 3-byte DMA buffer for USART RX; need to fill it to
% trigger an interrupt
fwrite(btv.hdl.daq.bbBt,[32+reg,0,0]);
pause(0.1);
loop=1;
while (get(btv.hdl.daq.bbBt,'BytesAvailable') < 1) && (loop < 100),
   fwrite(btv.hdl.daq.bbBt,[32+reg,0,0]);
   loop=loop+1;
   pause(0.1);
end
if get(btv.hdl.daq.bbBt,'BytesAvailable') > 0
    [data,~] = fread(btv.hdl.daq.bbBt,get(btv.hdl.daq.bbBt,'BytesAvailable'),'uchar');
    val = data(1);
end
```

Streaming data transmission

```
function EEGData=adsStream  % Potentially lossy; data only but fast
global btv;
if get(btv.hdl.daq.bbBt,'BytesAvailable') > 0,
   aa=fread(btv.hdl.daq.bbBt,get(btv.hdl.daq.bbBt,'BytesAvailable'),'uchar');
else
   EEGData=[];
   return
end
aa=[btv.daq.buf.adsResidual; aa];
pos=1;
samples=0;
len=length(aa);
OKpos = 0;
chanData=zeros(length(btv.daq.settings.daq.Ports{1}),1);
while (len-pos+1) >= btv.daq.settings.daq.bbBt.PacketLen
   if (aa(pos) == btv.daq.settings.daq.bbBt.Headers(1)) && (aa(pos+1) ==
btv.daq.settings.daq.bbBt.Headers(2)) %header byte
       samples = samples + 1;
       if ~isempty(btv.daq.buf.packetNumberRead)
           if aa(pos+2) ~= mod(btv.daq.buf.packetNumberRead+1,128)
               disp('Warning: missing packet.');
               disp(['Expected ',num2str(mod(btv.daq.buf.packetNumberRead+1,4)),'; read
',num2str(aa(pos+2)),'.']);
           end
       end
```

```
        btv.daq.buf.packetNumberRead=aa(pos+2);
        %Process channel data; assumes incoming data is
        %big-endian
        for i=1:8
            cc = aa(pos+3+(i*3))*(2^16) + aa(pos+4+(i*3))*(2^8) + aa(pos+5+(i*3));
            if cc >= 8388608, cc = cc - 16777216; end
            chanData(i,samples) =  cc * btv.daq.settings.daq.bbBt.bitResolution(i); % Convert to
[V]
        end
        for i=9:14
            cc = aa(pos+30+(i-9)*2)*(2^8) + aa(pos+30+(i-9)*2+1); % Accel/gyro sensor using uint16
            if cc >= 32768, cc = cc - 65536; end
            chanData(i,samples) =  cc * btv.daq.settings.daq.bbBt.bitResolution(i); % Convert to
[V]
        end
        pos = pos + btv.daq.settings.daq.bbBt.PacketLen;
        OKpos = pos - 1;%last valid byte
    else
        pos = pos + 1;
    end %if command else data   end
end
if (OKpos < len)
    btv.daq.buf.adsResidual = aa((OKpos+1):end); % retain left over bytes
else
    btv.daq.buf.adsResidual = [];
end

if btv.daq.flag.recordon == 1, % % save data when needed
    fid=fopen(fullfile(btv.daq.settings.data.datasavingpathname,...
                [btv.daq.settings.datasavingfilename btv.daq.settings.data.datasavingext]),'a+');
    switch btv.daq.settings.data.datasavingext
        case '.bin'
            fwrite(fid,chanData(:),'double');
        case '.txt'

fprintf(fid,'%+1.6e\t%+1.6e\t%+1.6e\t%+1.6e\t%+1.6e\t%+1.6e\t%+1.6e\t%+1.6e\t%+1.6e\t%+1.6e\t%+1.
6e\t%+1.6e\t%+1.6e\t%+1.6e\r\n',chanData);
    end
    fclose(fid);
end
EEGData=chanData';
```

Enable calibration mode

```
function adsCalibration  %#ok<DEFNU>
global btv;
%
hdl=msgbox('Reset device...wait','BCI2VR');
drawnow;
% Turn of continuous data logging to set channel register
deviceStop;
for i=1:8           % Only one ADS1299 device is supported; only 8 channels available
```

```
    if strcmpi(get(btv.hdl.daq.tb1.calibration,'State'),'on')
        switch btv.daq.settings.daq.ChannelGain{1}(i)
            case 1
                adsWriteReg(uint8(btv.daq.settings.daq.adsOpc.CONFIGCHnSET + i-1),
uint8(btv.daq.settings.daq.adsOpc.TEST_SIGNAL+ btv.daq.settings.daq.adsOpc.GAIN_1X));
            case 2
                adsWriteReg(uint8(btv.daq.settings.daq.adsOpc.CONFIGCHnSET + i-1),
uint8(btv.daq.settings.daq.adsOpc.TEST_SIGNAL+ btv.daq.settings.daq.adsOpc.GAIN_2X));
            case 4
                adsWriteReg(uint8(btv.daq.settings.daq.adsOpc.CONFIGCHnSET + i-1),
uint8(btv.daq.settings.daq.adsOpc.TEST_SIGNAL+ btv.daq.settings.daq.adsOpc.GAIN_4X));
            case 6
                adsWriteReg(uint8(btv.daq.settings.daq.adsOpc.CONFIGCHnSET + i-1),
uint8(btv.daq.settings.daq.adsOpc.TEST_SIGNAL+ btv.daq.settings.daq.adsOpc.GAIN_6X));
            case 8
                adsWriteReg(uint8(btv.daq.settings.daq.adsOpc.CONFIGCHnSET + i-1),
uint8(btv.daq.settings.daq.adsOpc.TEST_SIGNAL+ btv.daq.settings.daq.adsOpc.GAIN_8X));
            case 12
                adsWriteReg(uint8(btv.daq.settings.daq.adsOpc.CONFIGCHnSET + i-1),
uint8(btv.daq.settings.daq.adsOpc.TEST_SIGNAL+ btv.daq.settings.daq.adsOpc.GAIN_12X));
            case 24
                adsWriteReg(uint8(btv.daq.settings.daq.adsOpc.CONFIGCHnSET + i-1),
uint8(btv.daq.settings.daq.adsOpc.TEST_SIGNAL+ btv.daq.settings.daq.adsOpc.GAIN_24X));
            otherwise
                close(hdl);
                errordlg('ADS1299 programmable gain is one of 1, 2, 4, 6, 8, 12 and 24.');
                deviceClose;
                return
        end
    else
        switch btv.daq.settings.daq.ChannelGain{1}(i)
            case 1
                adsWriteReg(uint8(btv.daq.settings.daq.adsOpc.CONFIGCHnSET + i-1),
uint8(btv.daq.settings.daq.adsOpc.ELECTRODE_INPUT+ btv.daq.settings.daq.adsOpc.GAIN_1X));
            case 2
                adsWriteReg(uint8(btv.daq.settings.daq.adsOpc.CONFIGCHnSET + i-1),
uint8(btv.daq.settings.daq.adsOpc.ELECTRODE_INPUT+ btv.daq.settings.daq.adsOpc.GAIN_2X));
            case 4
                adsWriteReg(uint8(btv.daq.settings.daq.adsOpc.CONFIGCHnSET + i-1),
uint8(btv.daq.settings.daq.adsOpc.ELECTRODE_INPUT+ btv.daq.settings.daq.adsOpc.GAIN_4X));
            case 6
                adsWriteReg(uint8(btv.daq.settings.daq.adsOpc.CONFIGCHnSET + i-1),
uint8(btv.daq.settings.daq.adsOpc.ELECTRODE_INPUT+ btv.daq.settings.daq.adsOpc.GAIN_6X));
            case 8
                adsWriteReg(uint8(btv.daq.settings.daq.adsOpc.CONFIGCHnSET + i-1),
uint8(btv.daq.settings.daq.adsOpc.ELECTRODE_INPUT+ btv.daq.settings.daq.adsOpc.GAIN_8X));
            case 12
                adsWriteReg(uint8(btv.daq.settings.daq.adsOpc.CONFIGCHnSET + i-1),
uint8(btv.daq.settings.daq.adsOpc.ELECTRODE_INPUT+ btv.daq.settings.daq.adsOpc.GAIN_12X));
            case 24
                adsWriteReg(uint8(btv.daq.settings.daq.adsOpc.CONFIGCHnSET + i-1),
uint8(btv.daq.settings.daq.adsOpc.ELECTRODE_INPUT+ btv.daq.settings.daq.adsOpc.GAIN_24X));
            otherwise
```

```
                close(hdl);
                errordlg('ADS1299 programmable gain is one of 1, 2, 4, 6, 8, 12 and 24.');
                deviceClose;
                return
        end
        if btv.daq.settings.daq.UsedChannels{1}(i) == 0,  % %turn off unused channels
            adsWriteReg(uint8(btv.daq.settings.daq.adsOpc.CONFIGCHnSET + i-1),
uint8(btv.daq.settings.daq.adsOpc.PDn+ btv.daq.settings.daq.adsOpc.SHORTED));
        end
    end
end
deviceStart;
close(hdl);
```

Published with MATLAB® R2013a