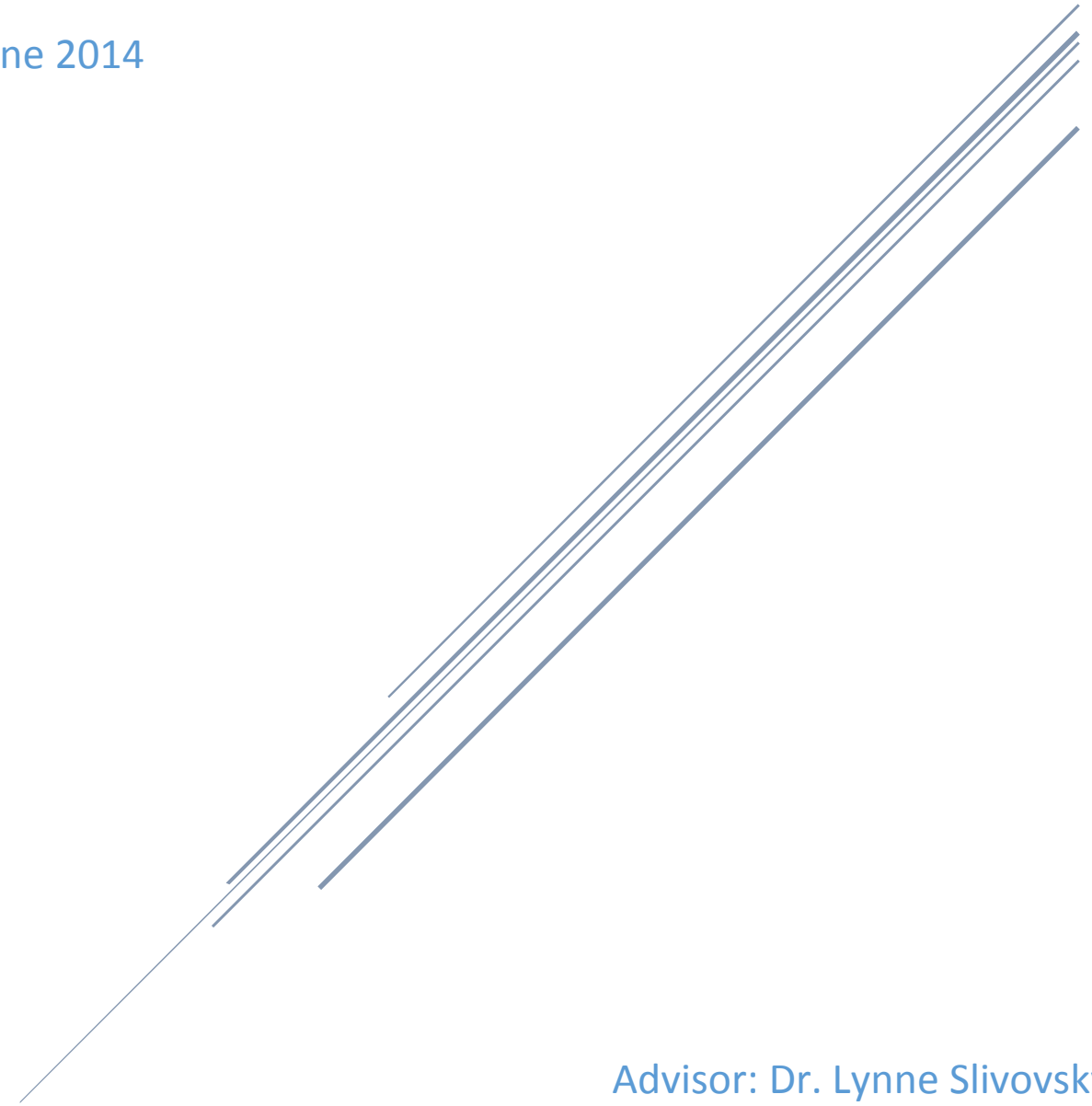
# Alternate Computer Input Device for Individuals with Quadriplegia

June 2014



Advisor: Dr. Lynne Slivovsky



California Polytechnic State University, San Luis Obispo

## Disclaimer

The conceptual designs, final solution, justification and all rationale for this Cal Poly San Luis Obispo Multidisciplinary Senior Project were to the best of our team's knowledge and efforts. All decisions, justifications and rationale for this project were based on a general consensus within the team. The device designed and created in this report is for Senior Project use only and was designed for one person in mind- our user, Brook McCall. This device does not constitute a state-of-the-art device.

No reliance should be made by any user/client on this report but instead users should make their own independent justifications and decisions. Under no circumstances shall the sponsor, advisor, and members of this team be liable for any indirect, incidental, special, punitive, consequential or similar damages that result from the use of this device. Also, sponsors, advisors, and designers on this team will not be liable for any damage whatsoever resulting from misuse of this device.

# Table of Contents

# List of Tables

# List of Figures

# List of Nomenclature

| | |
|---|---|
| DoF | Degrees of Freedom |
| EEG | Electroencephalography-the recording of electrical activity along the scalp |
| FDA | Food and Drug Administration |
| GND | Ground Connection |
| $I^2C$ | Inter-Integrated Circuit |
| IMU | Inertial Measurement Unit |
| KINE | Kinesiology |
| LED | Semiconductor diode that emits light |
| LMB | Left Mouse Button |
| PCB | Printed Circuit Board |
| RAM | Random Access Memory |
| RMB | Right Mouse Button |
| RoHS | Restriction of Hazardous Substances |
| SCL | Serial Clock Line |
| SDA | Serial Data Line |
| USB | Universal Serial Bus |
| $V_{in}$ | Voltage in |
| $V_{out}$ | Voltage out |
| $V_{ref}$ | Reference Voltage |

# Executive Summary

This project details the design development of an alternative computer input system that allows a person with quadriplegia to move a computer's cursor and activate left and right click button inputs. After researching and analyzing possible solutions, an end design was chosen that most appropriately satisfied all user requirements and engineering specifications. This final design employs a head mounted Inertial Measurement Unit (IMU) with 9 DoF (Degrees of Freedom) to track head movements and correlate these motions to computer cursor movements. A Sip-Puff Transducer monitors and interprets a user's application of negative and positive air pressure differentials to a vinyl tube as analog voltages, which are then interpreted over time to trigger left and right click events. An Arduino Due microcontroller is used to interpret and process these inputs and send mouse commands to the user's computer via a USB connection. In addition to the sensing hardware, there are two indicator LEDs which display the state of the left and right mouse buttons. There are also two adjustment potentiometers, which can be turned to adjust the sensitivity of the mouse tracking and the sip-puff click sensing window.

This system improves upon other alternative computer interfaces by allowing the user to more easily perform complex and non-linear tasks such as file organization and digital painting/drawing. Two accelerometers were initially incorporated into the design to be strapped to the upper arms of the user, and upward and downward accelerations caused by the raising and lowering of each shoulder would have corresponded to the activation of the Control and Shift keys. However, due to issues with program timing and computational complexity, these parts of our design that operated the control and shift keys were abandoned.

Of the nineteen engineering specifications derived from our customer requirements, we failed to meet only three of them: the input latency was 30 ms greater than our target value of 50 ms, we were unable to measure our devices' meantime to failure due to inadequate resources for that test, and because we were unable to include the accelerometers in the final product, our last unmet specification was inconclusive for determining whether or not the shift and control buttons could be actuated either independently or simultaneously. This engineering specification verification checklist can be found utilizing the table of contents, above. In conclusion, we believe our foray into accessibility technology to be a success.

# Chapter 1: Introduction

There are millions of people in the world that are diagnosed with different forms of paralysis, such as paraplegia and quadriplegia. These individuals must utilize alternative solutions to complete everyday tasks that the rest of us take for granted. Luckily, the cutting edge of assistive devices and technologies improve the ease with which many of these individuals navigate a world designed for those with full use of their arms and legs.

Our company, MindGames, has designed an assistive, custom computer input device for an individual diagnosed with quadriplegia. People with quadriplegia are capable of little to no movement below the neck due to paralysis of an individual's limbs and torso. Therefore, our team drew a solution through innovative design and background research, to allow our user to operate a computer without the use of the user's hands. We found the user's limits and capabilities before beginning this project in order to design a hands-free device suitable for the user. While the most important task at hand is to design an assistive computer input device for an individual living with quadriplegia- allowing user to interact with computer systems, we must also ensure that the user does not forfeit all physical activity involving their shoulders, neck, and head; as required by the sponsor of this project. Our ultimate goal for this project was at first to design a device that would be suitable for our client to effectively play video games, but has since changed to allowing our client to use a computer without relying entirely on dictation software, enabling her to utilize the freedom of on-screen, non-linear movements to organize computer files and have the opportunity to utilize drawing programs, such as paint, while maintaining some form of physical activity for our user.

Our team is collaboratively working with Kimberly Jones, an undergraduate Kinesiology Student here at Cal Poly, San Luis Obispo that is acting as a liaison between our engineering team and the sponsor of this project, Dr. J. Kevin Taylor - Chair of the Department of Kinesiology here at Cal Poly. Our advisor for this team project is Dr. Lynne Slivovsky, a professor in the Department of Electrical Engineering. Brooke McCall is our client diagnosed with quadriplegia that will give us insight as to the likelihood of our design succeeding in accordance to all of Brooke's design wants and needs. Hopefully, this completed project will be able to assist anyone with limited use of their hands or upper body.

## Objectives and Engineering Specifications

The primary focus of this project was to design a device that would be suitable for our client to operate a computer mouse and a couple modifier keys (shift and control) to organize computer files and have the opportunity to draw, utilizing programs such as paint. This device will be compatible with Windows, Mac, and Linux and will re-map the mouse axes and the shift and control keyboard buttons onto inputs, which can be actuated using only one's head and shoulders.

We have come up with a specification table to demonstrate our engineering specifications and our target values, tolerances, risk of each spec, and the compliances. This table can be viewed in Appendix B. In developing these engineering specifications, we took into account the specific desires and needs of our client, as well as general requirements for a device of this nature.

Our client expressed several specific preferences that will differentiate our device from others already on the market; for example, she was opposed to invasive oral components, such as bite sensors and tongue pressure sensors, being incorporated into the device to use as signal inputs. This desire translated directly to engineering specification 12 detailed in Appendix B. She also expressed that she wished to be able to organize her computer files more efficiently and effectively, thus requiring an engineering spec that ensured simultaneous functionality of button inputs; thus creating the engineering specs 17, 18, and 19. Additionally, our client is also capable of moving her shoulders (but not her arms or hands) unlike other individuals with spinal cord injuries whom may have some limited use of their arms and hands; and thus reflected that in engineering requirement 13.

Aside from our client's preferences, there are other basic safety and fundamental requirements inherent to what this device will do. For instance, specifications 1, 4, 7, 8, 11, and 16 are included because this device needs to be comfortable and safe to use. Specification 15 is also fundamental to the nature of the device so it must be a requirement. Also, requirements 5 and 14 are important to our device's role as a machine input mechanism because high latency between the user's articulation of an input and that action's effect on the system, as well as incorrect detection of user input, can result in difficulty in using the device, as well as a sub-par user experience.

As for the cost, we chose our target prototype manufacturing cost by considering the price of other products that meet the same end, along with the cost of components likely to be used in this device. Competing devices ranged from $800 to upwards of $1500, except for the Quad Controller which costs approximately $300 because it uses primarily mechanical systems (which are cheaper than MEMS inertial measurement devices) and is not sold for a profit.

The relations between these customer requirements and the engineering specifications developed from the user's wants and needs are demonstrated in the Engineering House of Quality, or the QFD, located in Appendix C. All numbers that are present within each appropriate cell, were chosen by our team- depending on how strongly we agreed about the correlation between one specific customer requirement and a correlating engineering specification. A number (1) implied there is not much correlation between that requirement and that specification. However, a number (3) implied that there is a strong correlation between the two and must be taken into consideration. An empty cell box signifies there is no correlation between that specific engineering specification and the correlating customer requirement listed.

On the far right of the house of quality diagram, a series of numbers 0-4 are once again present. This section demonstrates our belief on how strongly a customer requirement is fulfilled by three different competitive products with similar end results we are trying to accomplish with this project. A number (0) signifies that we felt that specific competitive product did not fulfill that correlating customer requirement at all. A number (1) signifies that we felt a slight correlation, a number (2) signifies we felt a decent correlation, and so on with number (4) signifying a strong correlation.

Lastly, at the very bottom of the house of quality diagram are three rows with "Benchmark" written in those cells. The numbers in the consecutive cells after, demonstrate the values for our engineering specifications of each different product (If we were able to find those values listed somewhere; otherwise, they are blank cells). These were used to improve and/or re-evaluate our design.

## Project Management Plan

Through the duration of this project, our team has followed through on time with key milestones for this project. A full, detailed list of our projected timeline for the entire year can be viewed in our Gantt chart located in Appendix D.

As stated in our team contract, throughout the execution of this project, our team continued to collaborate together and help each other when help is desired or needed from another team member. We tried to complete all of our tasks in a timely manner and kept a linear path to completion of this project utilizing the Gantt chart we had devised as a team. All team members continued to give updates at meetings about the progress of the project and the individual sections they were assigned each week. All team members continued to collaborate together whether they were extremely familiar with certain sections in the development of this project or not. For example, Toby Elder was actively involved in the programming development for this project, Michelle Martinez was actively involved in the solid-modeling and prototyping stages, and David Sylvester was actively involved in the circuitry/microcontrollers design and assembly aspect of this project - but all team members are involved in each area and helped perform the tasks necessary to complete this project in a timely manner. Michelle Martinez was the primary correspondent between our sponsor, Dr. Taylor, and the team; as well as keeping contact with Kimberly Jones, our KINE department team member, and our user, Brooke McCall.

As stated above, our team was in accordance with our project schedule for the year, which includes all necessary milestone deadlines. Documents provided to our sponsor as we progressed through the completion of this project included those such as: our Project Requirements, Conceptual Design Report, and Finalization of our prototype. The full list can be referred to in Appendix D with all deadlines included. Throughout the year, the team continued to meet every week for approximately 12 hours per week (shorter meetings resulted from lower necessity to meet) in order to complete milestones prior to deadlines, which enabled us to review all submissions prior to our submission dates.

# Chapter 2: Background

An individual with Quadriplegia has a spinal cord injury located in the upper portion of the spine. This will cause paralysis of all four of the individual's limbs in return. In addition to the arms and legs being paralyzed, the abdominal and chest muscles will also be affected, resulting in difficulty breathing at a normal rate and the inability to properly cough and clear the chest. People with this type of paralysis are referred to as Quadriplegic, or occasionally as Tetraplegic. As defined on mdguidelines.com, paralysis is the loss or impairment of one or more muscle groups that provide voluntary movement, due to lesion of the neuromuscular mechanism. It results from an interruption in the pathway of the brain signaling the muscle groups to move and react. A variation of different forms of paralysis result from where in the spinal cord the patients have caused injury. Individuals diagnosed with quadriplegia have had serious injuries located on the spinal cord above the first thoracic vertebra.

Within the last two years, starting in September 2011, Cal Poly engineering students have had two fairly successful senior projects centered on developing a controller system to give an individual with quadriplegia allowance to play video games- which could be analyzed for potential use with our team's project. The senior project finished in March of 2012 involved a system consisting of a mouth guard with embedded bite sensors and an attached gyroscope and infrared emitter attached to the front of the mouthpiece. This device connected to a module that housed the microprocessor, modified Wii controller hardware, as well as the associated power management hardware via an attached cable. This system was specifically designed to be used with the Nintendo Wii system and emulate most of the functionality of a traditional Wii controller through the use of head movements and bite force.

Following after the first attempt to successfully design and test a prototype for this project, a second round of students attempted to successfully finish the senior project in May 2013. This most recent senior project group created a custom molded mouthpiece with embedded force sensors that would be actuated by the user's tongue. The sensors and associated electronics were connected to a microcontroller, which served to interpret the inputs as movements in a custom computer game that the team created. Although these two previous senior projects were video game related, the same technology and design criteria is still valid for research, observation, and possible implementation for this current, reformed project.

## Products on the Market

To begin this project, our team had to first perform extensive research to find components and devices, either out on the market or developed for personal use, which could possibly assist us in the development of what is essentially a hands-free mouse. After conducting some research for products and devices, which aim to solve similar problems as our project (allowing users to interact with a computer interface hands-free), we have found the following:

A user by the name of "IronChris" on MakeyMakey.com, has drawn a solution to discovering a way for individuals with quadriplegia to interact with their surroundings while playing video games. The author of this post illustrates that he proposes to build a computer input device for gaming that will compose of a wheelchair headrest on which conductive threads will be utilized

to incorporate a number of inputs- the number of inputs would vary depending on how severe the user's case of paralysis is, and it can be changed by simply attaching/ detaching the leads to the headrest (Figure 1). The user can then signal the video game system to do what is wanted by moving their head to activate the respective conductive lead on the headrest. This design would enable users to still be able to communicate with family and friends while playing the video games because it does not interfere with their mouth in any way- which is a positive attribute of this design. We want our user to be able to have a natural experience as close as possible to playing video games like those of us without disabilities do.

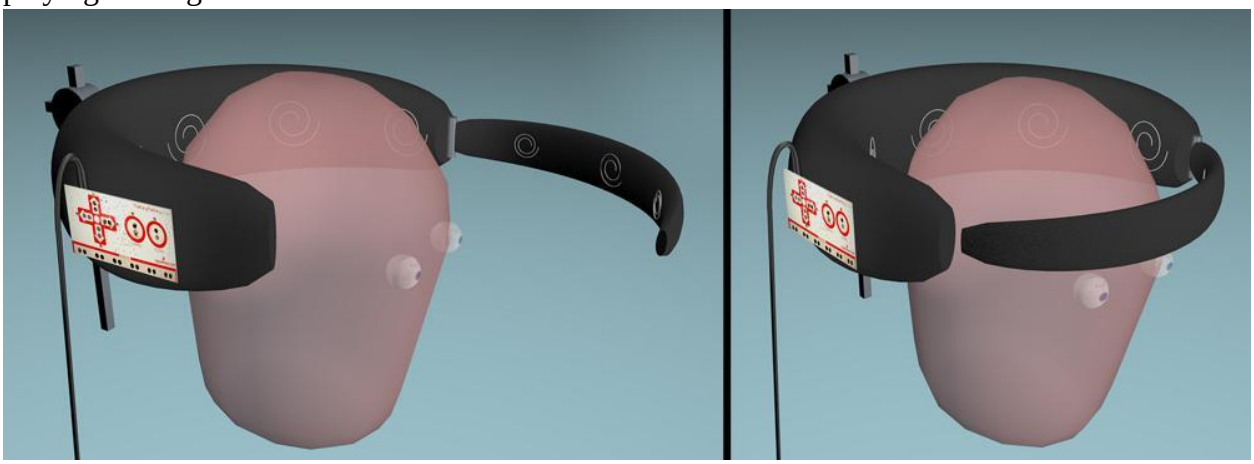


FIGURE 1: HEADREST CONDUCTIVE LEADS DESIGN

A Russian medical company, Gravitonus, has designed an "Alternative Computer Control System" (ACCS) (Figure 2) that will help paralyzed individuals control computers using a customized mouth device. The mouth device contains a tongue controlled directional command module that encompasses 12 additional commands, as well. This wireless device enables the user to sit in Gravitonus' chair, equipped with a computer and monitor, comfortably and enables them to communicate digitally through the computer. According to their website, it does not interfere with breathing, talking or drinking fluids while it is in use.

FIGURE 2: GRAVITONUS SEATING DEVICE FOR USE OF ACCS

Another company, called Nuance, has produced a product solving our problem from a different angle. This company sells a voice recognition system called Dragon® that enables the user to control much of the function of the computer using voice input. With the software the user can perform a variety of tasks including dictating text, sending emails, web browsing, and other computer functions. Since the software allows the computer to be controlled off of voice input alone, it is a viable option for people with quadriplegia to use their computer.

Adding onto the list of companies that currently have end results somewhat compatible to ours, is the company Origin Instruments. This company produces a mouse emulation system called Headmouse® Extreme (Figure 3) that uses a camera mounted on top of a user's computer screen to track a disposable adhesive dot that the user places on their head. The camera tracks the movement of the reflective dot and translates this movement into mouse cursor movement on-screen. The user can click an on-screen item by holding the cursor in one location for a set period of time. The company also sells a variety of other peripherals and software that allow the user to interact with the computer using Sip-Puff switches, on-screen keyboards, and other tools.

Another product which uses a similar screen-mounted camera-dot tracking system is the TrackerPro by Ablenet® (Figure 3).

**FIGURE 3: HEADMOUSE EXTREME (LEFT) AND TRACKERPRO (RIGHT), SCREEN-MOUNTED-CAMERA-BASED DOT-TRACKING SYSTEMS.**

Although this next product is not mass produced for the public, nor is it on the competitive market to be sold for profit, it has components that could aid our design and perhaps can be modified or improved upon. This device was created by a man named Ken Yankelevitz. He has developed a game controller that utilizes a series of Sip-Puff switches, a mouth controlled joystick, and lip switches that work with a modified Xbox 360 or Playstation 3 controller to allow the user to play and interact with an Xbox 360 or PlayStation 3 system (Figure 4). A Sip-Puff switch is essentially a small tube (or straw) that is attached to an electronic pressure sensor and can detect a sip or puff and convert these events to button inputs that will ultimately relay back to the system (Orin Instruments). This system ultimately allows users with quadriplegia to play most console games and compete with other players either nearby or online.


**FIGURE 4: MOUTH-CONTROLLED JOYSTICK FOR XBOX 360, BY KEN YANKELEVITZ**

A wide variety of eye tracking software and hardware are also currently present on the market as well. These systems can translate eye movement into cursor movement. Many utilize one or more cameras to track the user's pupils using visible or infrared light. Products that utilize this concept include the VT2, TM4, and TM4 mini systems by EyeTech™ Digital Systems which are peripherals that connect to an existing computer via USB and mount to the computer monitor. Other products that use a USB-based eye tracker are the Tobii EyeMobile which is designed to work with Windows 8 Pro tablets and the Tobii PCEye Go which works with laptops, desktops, and tablet computers (Figure 5).

FIGURE 5: TOBII EYEMOBILE AND TOBII PCEYE GO, EYETRACKING PERIPHERALS FOR TABLETS AND COMPUTERS

Several other products utilize tablet or desktop computers with integrated eye tracking system like LC Technologies' Eyegaze Edge Tablet and Desktop (Figure 6).


FIGURE 6: EYEGAZE EDGE TABLET AND EYEGAZE EDGE DESKTOP SYSTEMS WITH INTEGRATED EYE TRACKING HARDWARE

Additionally, another approach to eye tracking is currently being developed by Boston College called Eagle Eyes (see Figure 7). The system consists of electrodes placed around the eye which measure the myoelectric signals coming from the muscles used to move the eye. These signals correspond to the orientation of the eye, which can then be used to position the mouse cursor on screen. Even though this system is not a commercial product, it achieves a result that is similar to our design goals for our project.


FIGURE 7: EAGLEEYES MYOELECTRODE SETUP (LEFT) AND CONTROL INTERFACE (RIGHT)

## Standards

When designing any device (whether for personal use or for the public), safety is an important factor to consider for any product that will be in close contact with the human body. Therefore, this device will be in compliance with all relevant safety standards that may apply to either a human interface device or a device designed to be utilized on an individual's head or inside of an individual's mouth. One such safety standard is RoHS, first drafted by the European Union (EU), which restricts the usage of hazardous substances such as Lead and Cadmium. These materials are toxic to humans when ingested or inserted into the human body in any way and can be replaced by non-toxic alternatives. Any components that ship without RoHS compliance will be shielded such that they cannot come into contact and harm the user (DIRECTIVE 2011/65/EU, 2011). Another set of safety standards that our product shall abide by are those relevant to the standards for all medical devices; such as the Food and Drug Administration (FDA). The section with which we shall keep compliance with is located in the Code of Federal Regulations-Title 21(CFR 21), under part 872- Dental Devices.

# Chapter 3: Design Development

## Preliminary Design Concepts

When we began brainstorming how to solve this problem, there was a surprisingly wide variety of technologies that could help us achieve our goal. These included voice control, computer vision, EEG interpretation, mouth-based switches and joysticks, myoelectric muscle twitch sensors, and gesture-recognition from accelerometer and gyroscope data.

Conveniently, the majority of the viable technologies were not mutually exclusive with at least one other technology in the list, and could in fact be combined synergistically for a better end-product. For instance, voice control can be used at the same time as an IMU, even possibly along with myoelectric sensors. This fact gave us exponentially more combinations to consider, from which we selected the best candidates for more detailed analysis.

## Conceptual Designs

Once our team had derived the engineering specifications from our customer requirements, the first step in our design development for this project was to brainstorm as many ideas as possible. Our team held many sessions of brainstorming and came up with a decently extensive list of possible solutions and ideas. For our team's first brainstorming session, we utilized a modified version of the 6-3-5 method (using only three people instead of six). The 6-3-5 method is a quick, creative technique for brainstorming in which six participants individually write down three ideas for solutions to a specific proposed problem, within 5 minutes. Some of these ideas are illustrated below in Figure 8:

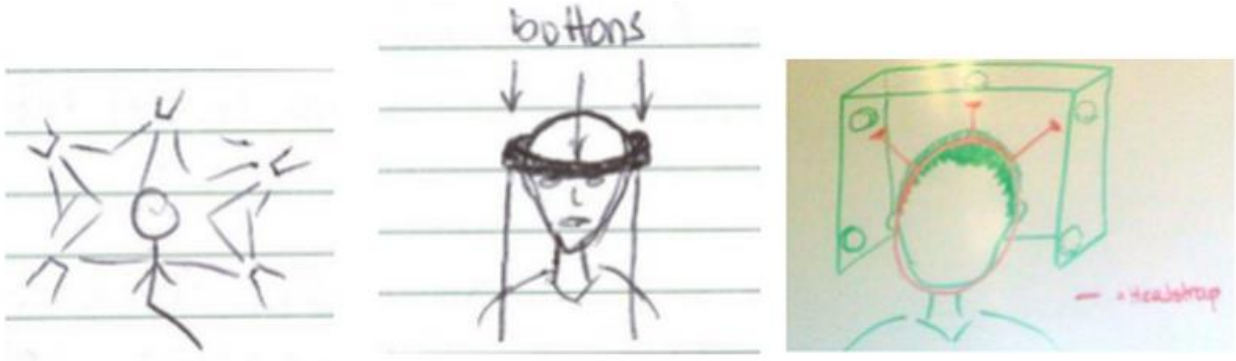



**FIGURE 8: INITIAL DESIGN CONCEPTS FOR QUADRIPLEGIC-ACCESSIBLE INPUT METHODS**

## Concept Selection

To select which concepts to use, our team had a brainstorming session of any and all ideas we could produce and listed them all out. Some of the ideas that we devised as a group included: Eye tracking, facial expression monitoring camera, EEG, IMU, etc. A full list of these initial ideas and their feasibility rankings can be viewed in Appendix E.

Using this ranking as a team, we found that for us, the IMU (inertial Measuring Unit) which consists of an accelerometer and gyroscope combination –and possible magnetometer– was the most feasible for this project; based on the round one concept generation.

After we came up with an extensive list of ideas, we decided to hold one more brainstorming session to generate even more ideas that could possibly spark interest. In this brainstorming session, we applied a few theoretical "what if" questions to help us think out of the box. We devised questions such as "What if we needed to design it cheaply?" and "What if we could not use custom electronics?" and generated a second list with ideas to a possible solution. A full list of these ideas is presented in Appendix F.

Once we had our list of possible solutions for this project, we weeded out those ideas that were of no value, were not feasible, or would just simply not work for this project through a group vote. Using our updated list of ideas, we then generated a morphological matrix that would allow us to list several different components that would perform the function of button pushing or axes movement. This matrix is presented below in Table I. Simultaneously, we also ranked different possible components for this system on their feasibility to perform the function of pressing buttons or moving along the axes. The rankings range from zero signifying no feasibility at all, to the number two signifying high feasibility. These rankings of the different components are presented below in Table I.

| Functions | Components used to satisfy functions | | | | | |
|---|---|---|---|---|---|---|
| Push Buttons | Sticks/Rod attachments | Myoelectric Sensors | Voice command | EEG | Pulley system | IMU |
| Move joystick/Mouse cursor | IMU | Eye tracking | Voice command | Chin Joystick | | |

TABLE I: MORPHOLOGICAL MATRIX FOR POSSIBLE SUBSYSTEMS

Utilizing the morphological matrix and the feasibility of the different components we would like to use in our solution, our team was able to pair the different components to devise different, but ideal solutions. A full list of possible design solutions is presented below in Table II.

| | Feasibility rating (0 - 2) |
|---|---|
| IMU + Myoelectric Sensors | 2 |
| IMU + Voice Command | 2 |
| Eye Tracking + EEG | 0 |
| Eye Tracking + Myoelectric Sensors | 1 |
| Eye Tracking + Voice Command | 1 |
| IMU + EEG | 0 |
| Voice Command + EEG | 0 |

TABLE II: IDEAL SOLUTIONS FROM MORPHOLOGICAL CHART

And finally, to make sure that we chose an ideal solution for our project, we compared each component with design criteria for this project in a decision matrix. Using this decision matrix, we were able to get a quantitative observation of which components seemed more feasible and appropriate for our design problem. Our design criteria can be seen on the left side of the matrix below. Using importance weighting, and a numerical system from zero to four, we then were

able to calculate the total scores for each component and compare their totals to one another. As presented below in Table III, our top three design components for our system are the IMU combo, the myoelectric sensors, and voice command. Also, when ranking the pairs of components (as seen back in Table II), we noticed the only two conceptual designs that received a ranking of 2, were the IMU and Myoelectric sensor combination and the IMU and voice command combination. Therefore, we decided our final design would be chosen from one of these two designs.

| | Weighting | IMU | Myoelectric | Voice | EEG | Eye Tracking | Pulley | Chin Joystick | |
|---|---|---|---|---|---|---|---|---|---|
| cost efficiency | 0.5 | 4 | 4 | 2 | 1 | 3 | 4 | 4 | |
| Input Reliability | 1.5 | 3 | 4 | 3 | 2 | 2 | 2 | 4 | |
| Computational Simplicity | 1.5 | 3 | 4 | 1 | 1 | 0 | 4 | 4 | |
| Mechanical Simplicity | 1.5 | 4 | 4 | 4 | 4 | 4 | 0 | 2 | 0=worst |
| Ease of Setup | 1 | 3 | 3 | 4 | 3 | 4 | 1 | 3 | 4=best |
| Customizability | 1 | 3 | 4 | 3 | 3 | 2 | 0 | 1 | |
| Total | | 23 | 27 | 20 | 17 | 16.5 | 12 | 21 | |

TABLE III: DECISION MATRIX FOR FINAL DESIGN

For us to decide which combination we wanted to implement for our project, we decided to create a simple list of pros and cons for each combination. With this technique, we could compare the two side by side and take into consideration the user preferences and what would be a best fit for the user. Presented below is the list of pros and cons for both combinations for our final design:

**a) IMU and Voice Command Combo:**

Pros:

- Ease of use through speech
- Does not require extra wires

Cons:

- Voice commands too slow
- User will not be able to talk and interact with others while playing video games
- Sometimes unreliable and susceptible to outside interference
- Complicated to customize

**b) IMU and Myoelectric Sensors Combo:**

Pros:

- User still able to communicate with others through speech while playing

- Fast
- Straightforward kinesthetic extension

Cons:

- Requires extra wires attached
- Detection requires sensitive electronics
- Have to replace electrode pads
- Requires application of conductive electrode gel

Finally, as a team we came to the conclusion that the IMU and Myoelectric Sensors Combination was an appropriate fit for our final design. Although three-fourths of the negative aspects of this design were due to the myoelectric sensors, they were still not much of a downfall to the design as the "cons" in the IMU and voice command combination were. We decided that the instability that resulted from the combination of the IMU and voice command software were going to be much more of a problem for our user than the use and replacement of electrode gel and pads. Therefore, our team has decided that our final conceptual design will be an integration of an IMU and myoelectric sensors.

We also brainstormed a variety of ways the different components can be attached to the user. This list that we generated as a team can be found below in Table IV. However, because our user had specified that for the mounting, she would prefer a hat, or the like, if possible, we decided to utilize a simple hat to mount the IMU for our prototype. After consulting her on her preferences for a final mounting solution, we decided to use a service or gaming headset.

| Mounting ideas for IMU: |
|---|
| Hat |
| Headband |
| Rear-wrapped Headpones |
| Hair Clips |
| Sweatband |
| Glasses |
| Visor |
| Google Glass-like |
| Emotive EPOC |

TABLE IV: POSSIBLE IMU MOUNTING SYSTEMS

After deciding on this complete design, we learned that Brook, however, was more interested in using the device for general computing rather than for gaming. Additionally, we learned that myoelectric sensors require three electrodes each, as opposed to the single electrode we had seen in multiple pictures. Shortly after learning about the downside to myoelectric sensors, Brook

informed us that while she wasn't quite comfortable with operating a giant bite switch, a simple Sip / Puff switch was certainly acceptable despite not wanting anything inside her mouth.
That in mind, we decided to replace the myoelectric sensors with a sip-puff switch mounted like a headset microphone and with an accelerometer mounted on each of Brook's shoulders, which would be mapped to the left and right mouse buttons along with the control and shift keys. This design is morphologically compatible with myoelectric sensors and is in fact somewhat simpler to implement, so we deemed that an entire second design phase wasn't necessary to account for the new information.

# Chapter 4: Description of Final Design

## Design Overview

Our final design incorporates input from a head-mounted IMU to track the mouse position. This IMU is hidden inside the headset's earpiece to produce a "cleaner" final product (item B in figure below). Shoulder-mounted accelerometers detect vertical movement that translates into initializing modifier keys (item C in figure below). Lastly, a Sip-Puff circuit will account for the all-important left- and right-click mouse buttons (item B in figure below). The Arduino Due microcontroller will be located in an enclosure near the user's computer (item D in figure below) and utilize wire connections to interface the IMU/Sip-Puff circuitry, which is mounted to a modified chat headset, as well as to the two 3-axis accelerometers located within enclosures mounted with removable Velcro armbands on the user's upper arms. The combined mountings of the design can be seen in Figure 9.

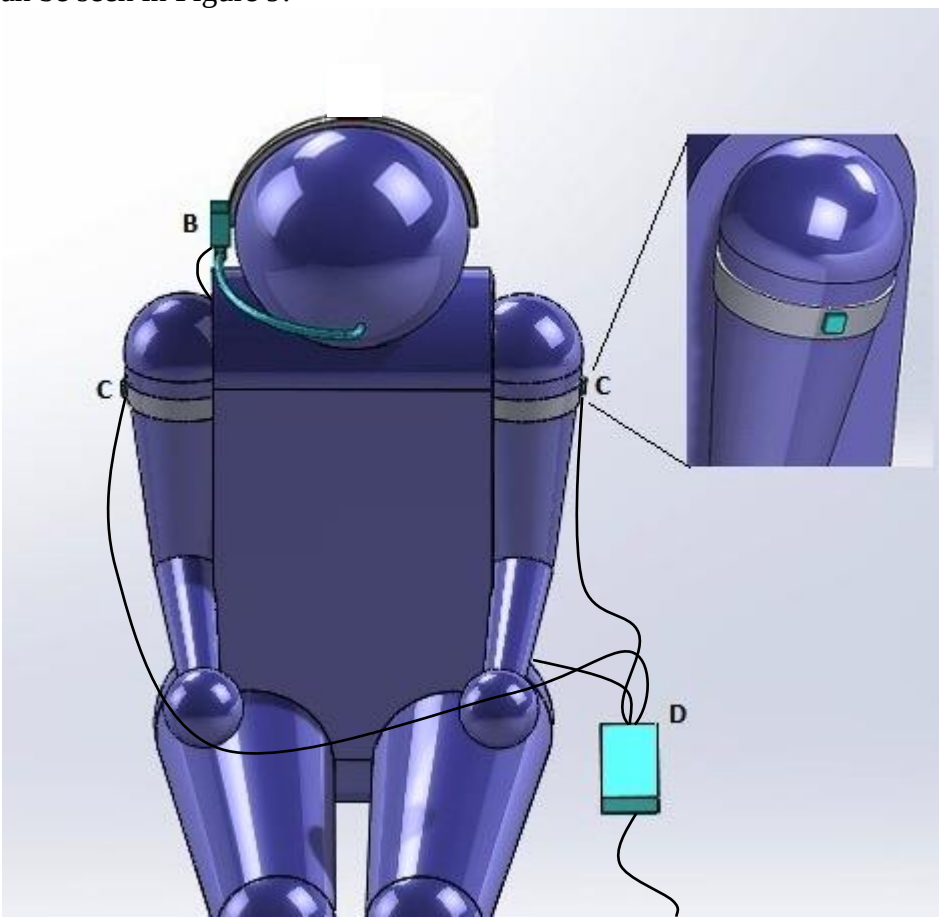



FIGURE 9: FINAL DESIGN SOLUTION

# Detailed Design Description

## Electrical Design

Our design consists of five primary electrical components: a head mounted IMU with 9 Degrees of Freedom (DoF), two analog 3-axis accelerometers, a Sip-Puff Circuit, and an Arduino Due microcontroller that processes the inputs from the aforementioned sensors and sends the appropriate mouse and keyboard inputs to the user's computer via USB. Only the Y-axis of each accelerometer is used. In addition to the sensing hardware, there are four indicator LEDs which display the state of the left and right mouse buttons, the control key, and the shift key. There are also three adjustment potentiometers, which can be turned to adjust the sensitivity of the mouse tracking, the sip-puff click sensing window, and the accelerometers' idle levels. The top-level schematic for the device can be viewed in Figure 10.

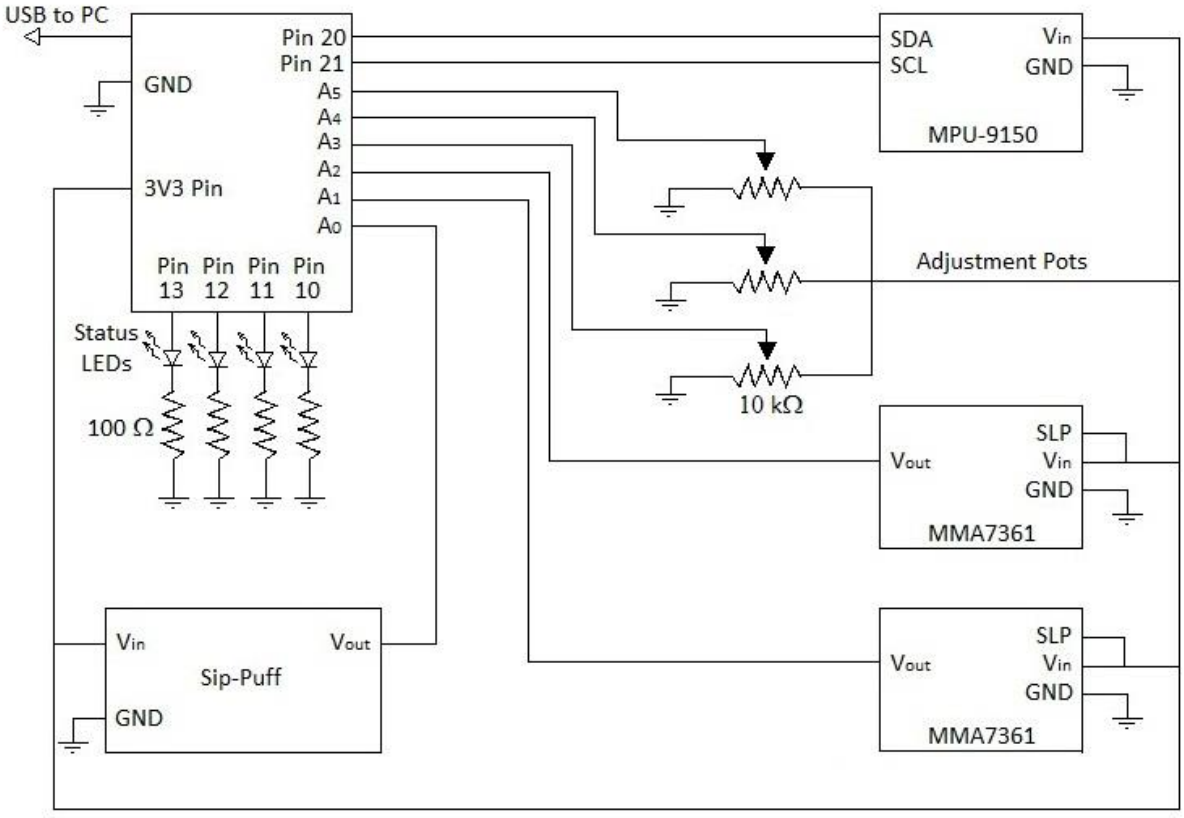



FIGURE 10: TOP-LEVEL SCHEMATIC OF SENSOR SYSTEM

The IMU will transfer information from the onboard accelerometer, gyroscope, and magnetometer to the Arduino Due via an $I^2C$ serial connection. The microcontroller will process these inputs using the onboard program to determine the position and movement of the user's head and correlate this to mouse movement commands that are sent to the computer. The Sip-Puff circuit will register the pressure applied by the user to a 3/16" vinyl tube that runs between the user's mouse and the pressure sensor of the Sip-Puff circuit. The Sip-Puff circuit consists of a pressure sensor that can register both positive and negative air pressures relative to current atmospheric pressure as well as an instrumentation amplifier that differentiates and amplifies the

voltage outputs from the sensor (Figure 11). The amplified signal from the instrumentation amplifier whose gain is set by the gain resistor R1 (see Appendix G for calculations) is then sent to an analog input of the Arduino Due, which interprets this signal as a left-click event when the incoming voltage passes a preset voltage threshold value (user puffs on vinyl straw) and as a right-click event when the signal passes below a preset voltage threshold value (user sips on vinyl straw). The right and left-click events are sent to the user's computer when initialized. For more detailed information on the devices used, see Appendix H for the associated data sheets.

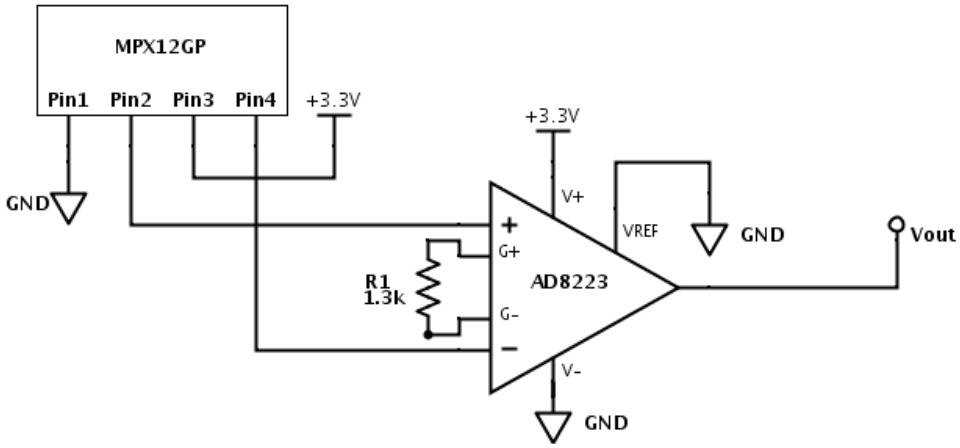



FIGURE 11: DETAILED SCHEMATIC OF SIP-PUFF TRANSDUCER AND INSTRUMENT AMPLIFICATION CIRCUIT

The two accelerometers will be attached to the user's upper arms and will register accelerations when the user raises and lowers their shoulders. These accelerometers will communicate with the microcontroller by outputting analog voltage levels which correspond to an acceleration within the sensor's sensitivity range, which will then be read by the microcontroller's built-in analog-to-digital converters the will monitor for accelerations in the direction of gravity. The raising of the left and right shoulders will be interpreted by the microcontroller as the activation of the Control key and Shift key respectively.

Wiring all of the sensors to the microcontroller will require seven lines: a shared +3.3V line coming from the Arduino Due's +3.3V pin and leading to each of the sensors, a shared ground line connected to one of the Arduino Due's ground pins and leading to each of the sensors, SDA and SCL lines for $I^2C$ serial communication that are used by the IMU connected to pins 20 and 21, respectively, on the Arduino Due microcontroller, two analog inputs connecting the shoulder-mounted accelerometers to the A1 and A2 pins of the microcontroller, and a connection between the $V_{out}$ of the Sip-Puff circuit (which can be seen in Figure 10) and the A0 pin of the Arduino Due.

## Electrical Component Rationale

The Arduino Due was the microcontroller of choice since it is able to emulate a keyboard and mouse through a USB connection with the computer. In addition, the greater amount of RAM, 32-bit processing capabilities, and faster clock frequency over similar microcontrollers like the Leonardo will greatly increase the computational speed and accuracy of our system

The MPU-9150 9 DoF IMU was chosen because of its relatively low cost and integration of a 3-axis magnetometer in conjunction with an accelerometer and gyroscope each with 3-axis of measurement. In addition there are currently usable open access libraries for this sensor that will assist with program development, and the $I^2C$ communication protocol assists with simplifying assembly and wire management.

The MMA7361 3-axis accelerometers were chosen due to their small size, low cost, and simple analog interface.

The Sip-Puff circuit components (MPX12GP Pressure Sensor and AD8223 Instrumentation Amplifier) were chosen due to the success of the open source "openSip+Puff" design by Jason Webb. His designs for a low cost and easily usable Sip-Puff system outclassed all commercial Sip-Puff products on the market during our research in terms of functionality and greatly reduced cost. Thus the Sip-Puff assembly in our project is derived from his work.

## Software Design

The software which will interpret sensor inputs and send mouse and keyboard commands to the computer will follow the flowchart presented in Figure 12, below.

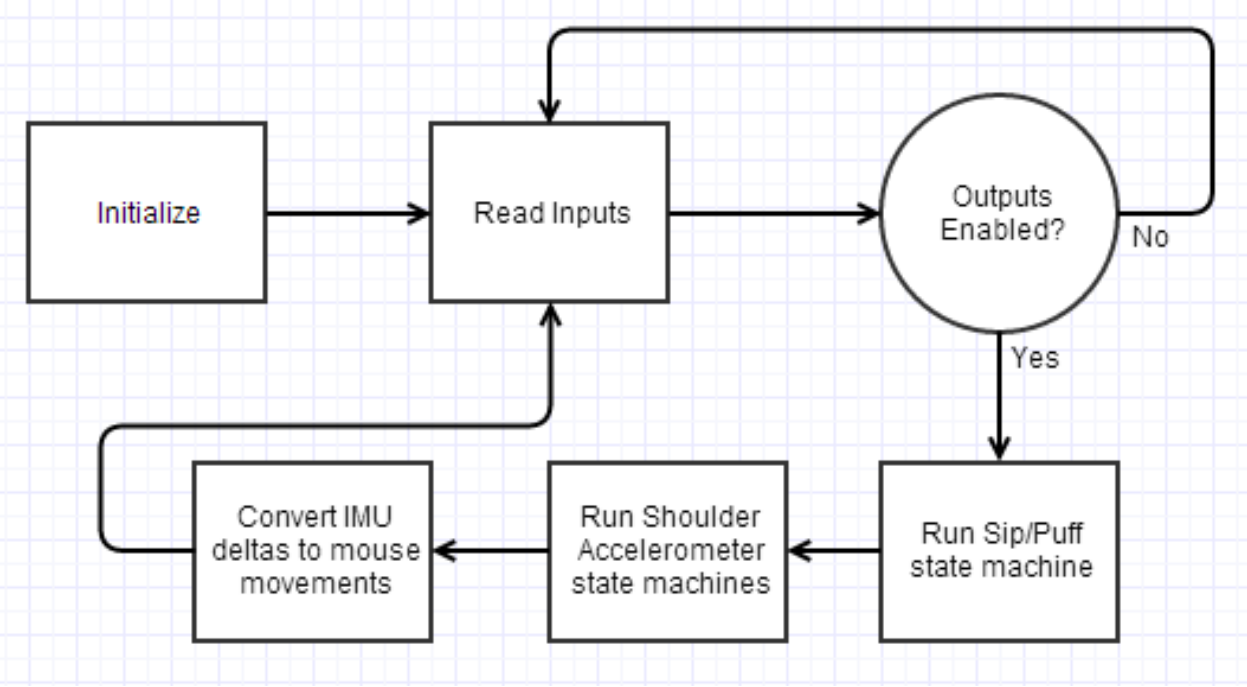



FIGURE 12: SOFTWARE FLOW DIAGRAM FOR SYSTEM TO CONVERT SENSOR DATA TO COMPUTER INPUT EVENTS

As you can see, after initializing, the Arduino Due will essentially loop between reading input values and, if outputs are enabled, converting those inputs to computer input events. This design uses a polling-based sensor reading system rather than an interrupt-based model because, when the device is actively being used, there will be no down-time that can be exploited for program simplicity or power savings. By nature, we want our mouse tracking to be reliable and persistent and not rely specific interrupts being fired.

As for the specific state machines that govern the operation of the Shift and Control keys and of the Left and Right mouse buttons, those are detailed in Figure 13 for the former and Figure 14 for the latter.

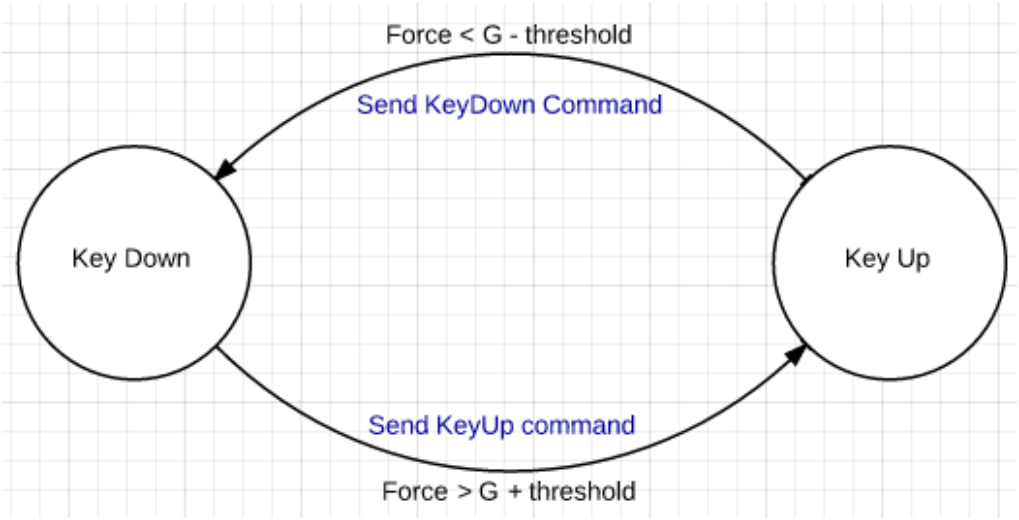



FIGURE 13: SHOULDER ACCELEROMETER INPUT INTERPRETATION STATE MACHINE

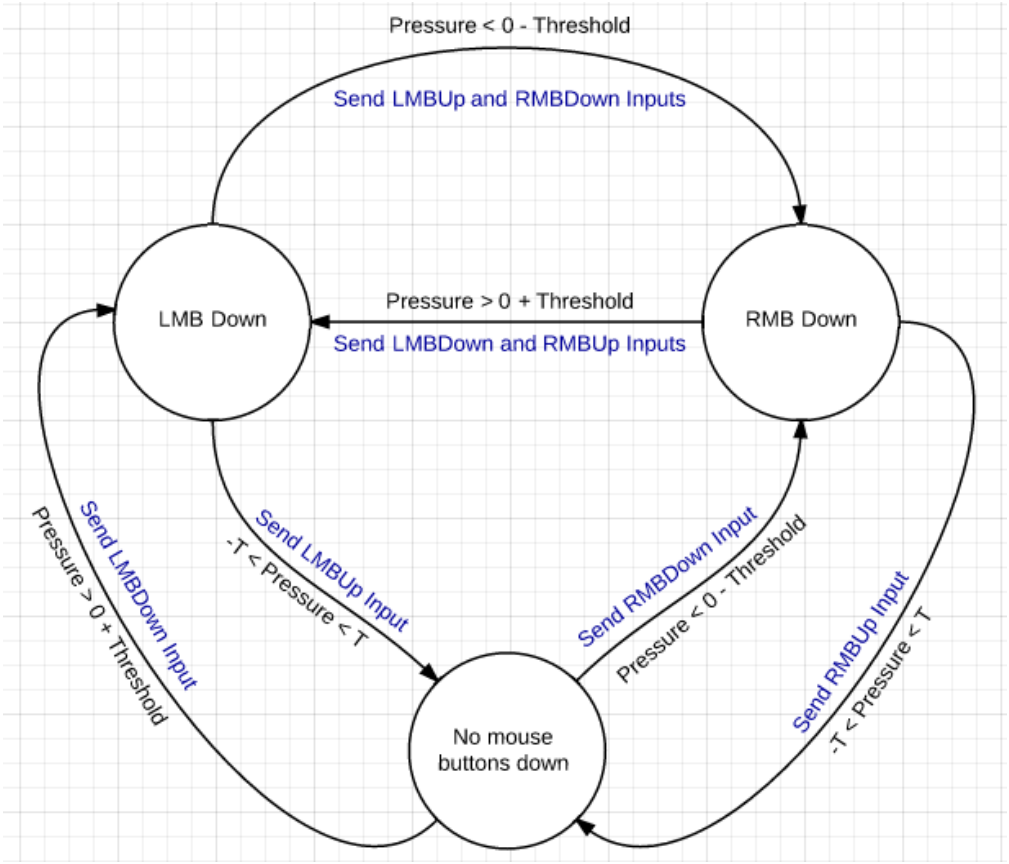



FIGURE 14: SIP-PUFF INPUT INTERPRETATION STATE MACHINE

## Mechanical Design

The basis of the mechanical fixture for the IMU and Sip-Puff electrical components will be inside the earpiece of a video game headset. The headset will allow our device to be adjustable in addition to enhancing the user's comfort when wearing the device. The flexible microphone will serve to support the vinyl tubing running from the Sip-Puff electronics to the user's mouth while allowing for the user to have the position and angle of the vinyl tube be adjusted as needed. Each of the accelerometers will be placed in enclosures attached with Velcro to fabric armbands and are worn on the upper arms during use. The size of the armbands will be variable using Velcro for adjustability as well as to make the armbands easy to secure and remove.

# Analysis Results

To, before we construct it, ensure that our device will function as planned, we have made sure to analyze the compatibility and functionality of relevant components and subsystems. The parts of the design which we have control over are the connections between the components, the amplification circuitry for the Sip-Puff transducer, the code running on the Arduino Due, and the mounting hardware. As such, we needed to analyze the logical connections between components, the operational characteristic of the Sip-Puff transducer, and the mechanical loads that will be imparted on the mounting hardware. As for the program running on the Arduino Due, nobody has attempted to prove an algorithm's correctness by anything other than demonstration since the 1960s, so that section of our design will be refined purely by trial and error.

Ranges for the software was determined and calibrated through iterative testing.

## Logic Circuitry

Analysis of the logic circuitry between our components was trivial: according to the datasheets, all logical connections operate using $I^2C$, and use 3.3v for logical "HIGH" and 0v for logical "LOW," so our sensors and processor will cooperate just fine.

## Sip-Puff Transducer Instrumentation Amplification Circuitry

Through analysis of the works of John Webb on his openSip+Puff design and the documentation for the MPX12GP Pressure Sensor and AD8223 Instrumentation Amplifier, along with analysis of the modified circuit configuration, we verified that the sensor would not cause the instrumentation amplifier to produce voltage outside of the operational parameters of our microcontroller, and would also not cause any erratic internal behavior, either. Detailed circuit analysis and calculations can be found in

# Cost Breakdown

Considering the purpose of our project is to create a usable prototype for our end user Brooke, our system was designed for use by a single individual as opposed to being designed for mass manufacturing. For our device the cost of the primary components used during development can be attributed to the Arduino Due (Amazon, $40), the MPU-9150 9 DoF IMU (Sparkfun $40), the two shoulder mounted MMA8452Q 3-axis accelerometers (Sparkfun $10 each), the MX12GP Pressure Sensor (Digi-Key $8.69), and the AD8223 Instrumentation Amplifier (Digi-Key $1.50). With all other associated connection and mounting components, the estimated total cost of

developing our prototype is $132.74. (See Appendix I for a full breakdown of vendors, contact info, and the component costs)

## Maintenance and Repair Considerations

Almost all of our design is meant to be easy for the user to maintain with little or no upkeep required. Some components that may need to be replaced or cleaned after prolonged use include the 3/16" inner diameter (ID) vinyl tubing that runs from the Sip-Puff sensor to the user's mouth. This component is easy to clean by removing the tubing from the barbed fitting of the pressured sensor and disengaging the mounting features on the adjustable base. Running soap and warm water through the tubing in addition to using a pipe cleaner should be sufficient to maintain the tubing. If needed additional vinyl tubing can be easily purchased at a local hardware store at little cost to the user. Another component that was designed with ease of repair in mind was the MS12GP Pressure Sensor. Since the component utilizes header pins and two screw for electrical and mechanical connections and is not directly soldered to the circuit, this part is easily replaced by the user using only a Philips head screwdriver. Replacement sensors can be purchased from Digi-key for less than $10 per unit and reinstalled into the Sip-Puff circuit.

# Chapter 5: Product Realization

## Manufacturing

The final version of our prototype consisted primarily of three assemblies that were manufactured from raw materials and repurposed products. The main control module which houses the Arduino Due microcontroller, the protoshield with status LED's and adjustment potentiometers, and the headset connector consists primarily of two sheets of machined 1/8" Lexan. These sheets have holes drilled to allow for the routing of wires through the top Lexan sheet to the microcontroller and protoshield below. The holes drilled in the bottom layer allow for the fixation of the Arduino Due via nuts and bolts. Four 1/2" holes were drilled at the corners of each of the sheets of Lexan to allow the use of ½" bolts, nuts, and standoffs to provide the structure for the enclosure.

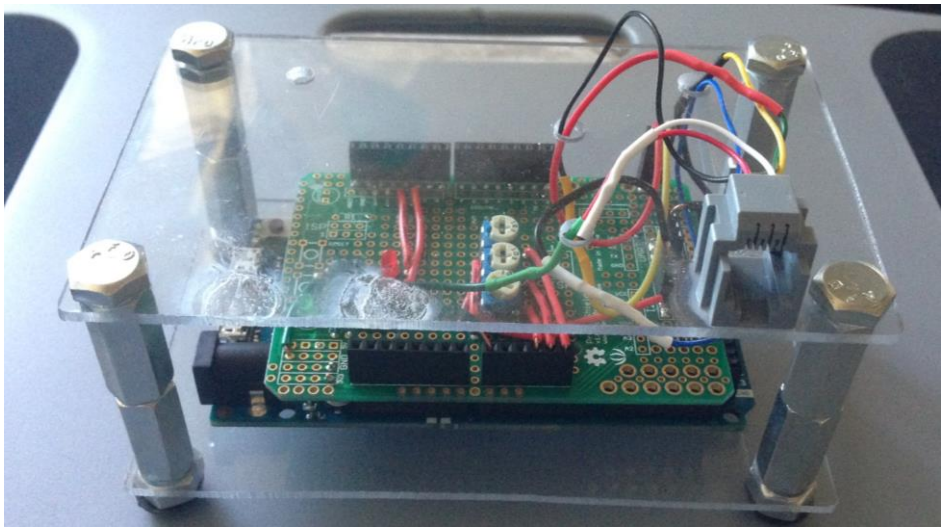


FIGURE 15. ARDUINO DUE WITH ATTACHED PROTOSHIELD WITHIN LEXAN ENCLOSURE.

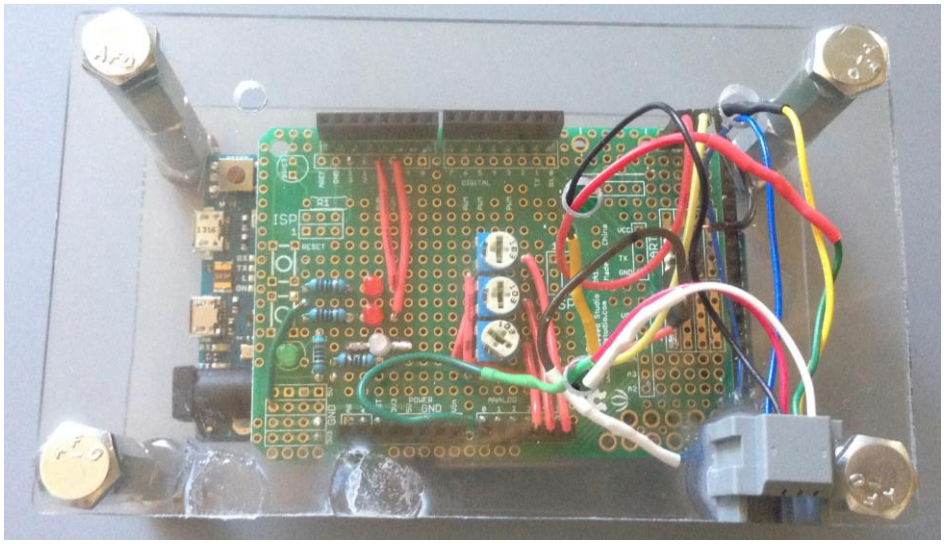


FIGURE 16. TOP VIEW OF MICROCONTROLLER ENCLOSURE. THE FOUR INDICATOR LED'S CAN BE SEEN TO THE LEFT OF THE ADJUSTMENT TRIM POTS. THE MODULAR RJ-25 PHONE JACK CAN BE SEEN ATTACHED TO THE TOP LEXAN LAYER (BOTTOM RIGHT).

The majority of the manufacturing went into the soldering and wiring of the protoshield and headset electronics. For the protoshield, three 10k ohm trimpots and four indicators LED's were used. All Components were soldered in place and wired according to the previously shown electronic schematics.

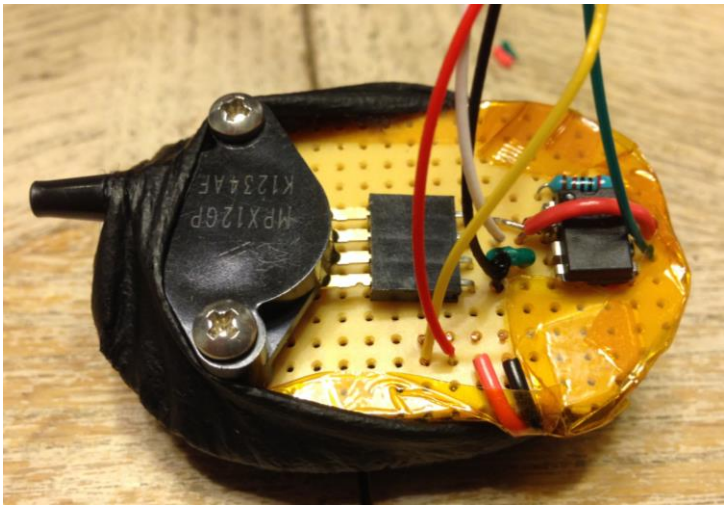


FIGURE 17. HEADSET PROTOBOARD SHOWING (LEFT TO RIGHT) PRESSURE SENSOR, FEMALE HEADER, AND INSTRUMENTATION AMPLIFIER WITH GAIN RESISTOR. THE ASSEMBLY IS PARTIALLY INSERTED INTO THE PLUSH EARPIECE.

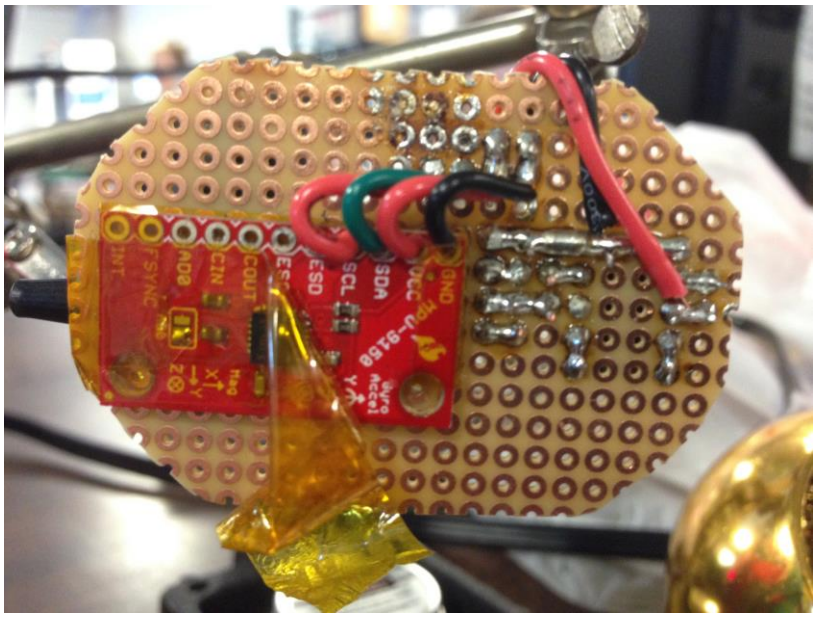



FIGURE 18. REVERSE SIDE OF THE PROTOBOARD SHOWS THE IMU WIRED IN PLACE AND PROTECTED WITH POLYIMIDE TAPE.

A mono headset with microphone purchased from Best Buy was used as the basis of the prototype's headset. The speaker and wiring for the original headset were removed to make room for the prototype's components. A perforated protoboard PCB from Radioshack was trimmed and filed to the approximate size of the earpiece such that the protoboard and electronics would fit within the plush earpiece cover. Once trimmed to size, the instrumentation amplifier, pressure sensor right-angle header, and gain resistor were soldered and wired together on one side of the

protoboard while the IMU was wired to the reverse side of the protoboard. The pressure sensor was inserted into the right-angle female header and two screws were used to fasten the pressure sensor to the protoboard. Cyanoacrylate glue was used to prevent the loosening of the screws over time. A small hole was cut into the plush headset so that the flanged tube of the pressure sensor could fit through to allow the user to connect and disconnect the vinyl sip-puff tubing.

An RJ-25 6-conductor jack with connected wiring was used on the headset to allow an RJ-45 6-conductor telephone cable to connect the headset to the microcontroller. This allowed the cabling between the two devices to be modular and have ample strain-relief to help prevent wire breaks over the lifetime of the device. Five of the wires from the jack were fed through the base of the headset where the original headset wiring had exited from the headset. The five wires (+3.3V, Ground, Serial Data, Serial Clock, and Sip-puff Analog) were wired and soldered into their respective locations on the on the protoboard. The protoboard and electronics were then layered with polyimide tape to protect the electronics from moisture and accidental shorts.

The earpiece of the headset connects to the rest of the headset via a ball and socket joint that was used by the original headset design. A small portion of the plastic earpiece that connects via the ball and socket was removed with a dremel tool to allow room for the pressure sensor. The protoboard with electronics was inserted into the plush earpiece followed by the modified plastic earpiece. Three small applications of cyanoacrylate glue were used to prevent the plush earpiece from becoming unintentionally detached from the modified plastic piece.

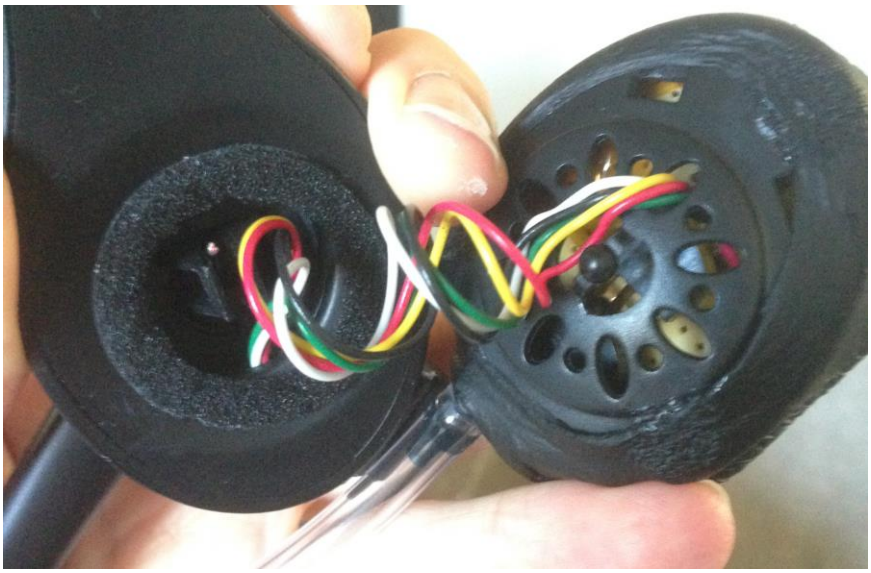


**FIGURE 19. VIEW OF SNAP-FIT BALL AND SOCKET JOINT BETWEEN THE MODIFIED PLASTIC EARPIECE AND THE HEADSET. THE WIRES WERE LOOSELY WRAPPED AROUND THE "SOCKET CLASP" (LEFT) OF THE HEADSET TO HELP PROVIDE EXTRA STRAIN RELIEF.**

The RJ-25 jack was spray painted black to match the headset and a small amount of hot glue was used to firmly adhere it to the base of the headset. A piece of large heat shrink tubing was cut to the approximate length of the flexible arm of the microphone boom and was partially shrunk. This allows for control of the position of the sip-puff tubing while still providing enough space for insertion and removal of the tubing when replacement is desired.

**FIGURE20. COMPLETED HEADSET WITH RJ-25 JACK (BOTTOM LEFT) AND VINYL SIP-PUFF TUBING BEING ROUTED ALONG MICROPHONE BOOM USING HEAT SHRINK TUBING (BOTTOM).**

## Prototype Differentiation from Planned Design

Our final prototype differs from our initial design primarily in that the shoulder-mounted accelerometers were removed. After manufacturing the shoulder-mounted accelerometers, they proved unreliable during testing and would have required too much fine-tuning by the user when strapped on to be practical. Aside from this set back the rest of our final design made it into this prototype.

## Recommendations for Future Manufacturing

Recommendations for future manufacturing include using custom printed circuit boards for the microcontroller and the headset. In theory it may be possible to integrate both the microcontroller and the headset electronics into the headset, such that only a USB cable between the headset and the computer is required. However, this would require a custom headset with ample space to allow for the electronics and wiring while remaining comfortable and lightweight for the user. Further designs could implement a wireless interface between the electronics and the computer, although this would require additional circuitry for the power management, battery, and wireless communication. The additional complications of attempting to implement and troubleshoot this design aspect led the team to forgo that feature in this iteration.

Final production costs would be greatly decreased by using two custom printed circuit boards: one for the headset electronics and one for the microcontroller and associated components. The parts listed below include primarily surface mount components as well as a microcontroller chip (ATmega32u4) and gyroscope/accelerometer (MPU 6050) that better fit a final product due to the reduced number of unused features and cheaper price-point.

## Cost Estimate for Future Production

If this design was to be mass manufactured for future production, then our cost estimate would change quite a bit compared to our current cost of developing this product. A full breakdown of the components needed, the estimated costs of each, and the links on where to obtain these components can be found in Appendix J. Our estimated cost for future production results in approximately $25 per unit (Price based on bulk order of 1000 units).

# Chapter 6: Design Verification (Testing)

## Test Descriptions

1. ***The weight applied to user by the system must be no more than 5 N.***

Test: Each component will be weighed and the combined weight of all user mounted components will be added for verification.

Result: The components were all placed on a sensitive scale to determine what the final weight of the entire device would be: Headset Mass (Weight) = 64 grams (0.63 N), Cable Mass (Weight) = 50 grams (0.49N), Maximum weight on user = 0.112 N. The final weight of the device resulted in 0.112 N which is significantly less than 5N.

2. ***The power consumption of the device will be no greater than 2.5 W.***

Test: The power drawn from the USB port of the computer can be measured using a multimeter. Since the USB port outputs five volts, the maximum allowable current is 500 mA. In addition, the computer's USB ports and the Arduino both have overcurrent protection that will activate and shut down the connection when more than 500 mA is drawn through the USB connection. Multimeter measurements and checking for activated overcurrent circuitry will be used to verify this specification.

Result: Device was connected to laptop and power drawn from the USB port was measured using a multimeter. The device consumes 0.936W with all sensors connected. Activated overcurrent circuitry was not observed.

3. ***The prototype manufacturing cost will be no greater than $500.***

Test: Calculate total cost.

Result: The current cost analysis performed on the component costs of our prototype indicate that the estimated cost is $130.05, which passes this specification. Further analysis will be made after prototype manufacturing has been completed.

4. ***The supply voltage required by the device will be no greater than 12 volts.***

Test: Check the power supply used for the project.

Result: Since our system is powered only by the 5 volt power supply from the USB port of the computer our device has passed this specification.

5. ***The maximum input latency between a physical user input and the receiving of a mouse or keyboard command by the computer will be no greater than 50 ms.***

Test: Using custom testing code integrated into the Arduino Due's programming routine, the time between the microcontroller receiving a user input and sending a command via USB can be measured in milliseconds. This result can then be sent to the computer via a serial connection where the results can be evaluated.

Result: Due to how often we had to read values form the Due's analog inputs, a rather lengthy process, we were unable to reduce the response time below approximately 80ms. However, we believe this is a perfectly acceptable amount, as the original value of the requirement was decided back when the device was intended to be used for fast, reflexive tasks such as video games, rather than basic computer use. 80ms will work fine for the current usage scenario.

6. ***All user wearable components must be able to maintain functionality after being dropped onto a wooden surface from a minimum height of 1 m.***

Test: The user wearable components will be disconnected from the microcontroller and dropped from a height of 1 m onto a wood floor. The sensors will then be reconnected and tested for full functionality. The entire device itself will also be dropped from the height of 1 m if the first test passes.

Result: Device was dropped several times on both hardwood and concrete floor( Cal Poly, SLO Engr IV floor) and did not have any damage and was still working properly. Test was a success 5/5 times.

7. ***The user-mounted components must not pierce, abrade, or lacerate the skin when used for a minimum duration of eight hours.***

Test: The device will be inspected for any protrusion, edges, or surfaces that would harm the user or be felt as uncomfortable prior to testing. Each team member will wear the device and monitor for any discomfort when using the device for the duration. Afterwards, the team member will be inspected for any scratches or abnormal skin discoloration.

Result: Teammates and roommates wore and tested the device for approximately 6-8 hours in a day and concluded there was no harm to the users. Test was a success 4/4 times.

8. ***The device must not expose the user to any hazardous material, edges, or voltages.***

Test: The design and manufacturing of this device will not incorporate or utilize any hazardous materials or dangerous voltages. The device will be thoroughly inspected for any sharp edges that may have been produced during manufacturing. If any nonconforming edges are found they will be filed and rounded.

Result: Users validated that there were no harmful edges, hazardous material, or voltages. Test was a success 5/5 times.

9. ***The system must be able to be set up by the user's assistant in no more than 10 minutes.***

Test: The team and individuals unfamiliar with the product will be asked to set up the system and will be timed in the process.

Result: The device was set up by four different users. The average time to set up the device, by somebody who was not familiar with our device, was 1 minute 49 seconds.

10. ***The mean time to failure must be no less than 7300 hours.***

Test: Although we do not have the resources and equipment to test and verify this engineering specification, we would test this using a fixture that simulates typical usage of the system and through repetitive testing to determine the mean time to failure.

Result: Inconclusive

**11. *The system must be ROHS compliant.***

Test: Research components and materials to be used to verify they are RoHS compliant.

Result: The components we have purchased for our device are from reputable companies and are regulated for sales in the U.S. Also, the soldering for the electrical wire connections is lead free. Yes, all components are RoHS compliant.

***12. The device must have minimal contact with the user's mouth.***

Test: Observe and ask user if approved by her.

Result: The end design has been shown to the end user of our device, and she has deemed the use of the Sip-Puff straw acceptable.

***13. The device must be usable only by articulating only one's head, shoulders, and neck.***

Test: All members, and user, will test the device and verify it can be utilized and function appropriately by articulating only head, shoulders, and neck.

Result: Tested by simply using the device and ensuring that it is working properly and accurately- Passed.

***14. The device must only signal inputs to the computer when intended by the user.***

Test: During the software development phase, iterative testing will be used to develop error checking routines to mitigate any potentials for false inputs by the user. Functional testing of the device by the team members will determine if the system causes any false inputs.

Result: The number of incorrect inputs was approximately one in every 200 actuations.

***15. Must interface with a PC as an input device.***

Test: The device will be connected to and tested with multiple computers to verify that each recognizes the device as an input and functions accordingly

Result: The device is recognized by most, if not all, PCs along with Mac and Linux.

***16. Must be considered comfortable by 80% of users.***

Test: A number of individuals will be asked to try on the device and give feedback on whether or not all components are obstructive, intrusive, or simply uncomfortable.

Result: 7/7 users found the device to be comfortable.

17. ***Cursor input position shall be able to be articulated independently from and simultaneously with button inputs.***

Test: During the software development phase, iterative testing will be used to verify that mouse movement and button inputs are not dependent on one another.

Result: Mouse button inputs and cursor movement are not dependent on one another.

18. ***The user must be able to activate Control, Shift, and Left or Right Click either simultaneously or independently.***

Test: During the software development phase, iterative testing will be used to verify the functionality of these inputs, both simultaneously and independently.

Result: Because the shift and control button inputs are not functional, this test is inconclusive.

19. ***The user must be able to click while holding left or right mouse button inputs (click and drag).***

Test: During the software development phase, iterative testing will be used to verify the functionality of these button inputs.

Result: User was able to click and drag 100% of the time.

## Specification Verification Checklist

| Spec # | Parameter Description | Target Value | Actual Value | Pass/Fail |
|---|---|---|---|---|
| 1 | Weight applied to user | 5 N | 0.112 N | Pass |
| 2 | Power Consumption | 2.5 W | 0.936 W | Pass |
| 3 | Manufacturing Cost per Unit | $500 | ~$130 | Pass |
| 4 | Supply Voltage | 12 V | 5V | Pass |
| 5 | Input Latency | 50 ms | 80 ms | Fail |
| 6 | Must be able to maintain functionality after being dropped onto a wooden surface from a height of | 1 m | 1 m | Pass |
| 7 | Must not pierce, abrade or lacerate skin when used for a duration of | 8 hours | 8 hours | Pass |
| 8 | Must not expose user to any hazardous materials, edges or voltages | None | None | Pass |
| 9 | Must be able to be set up by user's assistant in under | 10 min | 1min49 sec | Pass |
| 10 | Mean time to failure | 7300 hours | inconclusive | Fail |
| 11 | Must be RoHS compliant | TRUE | TRUE | Pass |
| 12 | Must not require invasive contact with inside of user's mouth | TRUE | TRUE | Pass |
| 13 | Must be usable by articulating only one's head, shoulders and neck | TRUE | TRUE | Pass |
| 14 | Must only signal inputs to the computer when intended by the user | TRUE | TRUE | Pass |
| 15 | Must interface with a PC as an input device | TRUE | TRUE | Pass |
| 16 | Must be considered comfortable with _____ % of users | 80% | TRUE | Pass |
| 17 | Cursor input position shall be able to be articulated independently from and simultaneously with button inputs | TRUE | TRUE | Pass |
| 18 | Must be able to activate Control, Shift, and Left or Right Click either simultaneously or independently | TRUE | Inconclusive | Fail |
| 19 | Must be able to click and hold left or right mouse button inputs(click and drag) | TRUE | TRUE | Pass |

# Chapter 7: Conclusions and Recommendations

## Conclusions

In the end, it is always a considerable challenge to allow someone to interact with technology in ways that weren't anticipated or accommodated for. We believe that even though, due to issues with program timing and computational complexity, we abandoned the parts of our design that operated the control and shift keys, all things considered, our final design nonetheless very effectively allows someone to control a computer's mouse without needing to move anything but their head; And that ability is critical for any person with quadriplegia who wants more precise control over their computer. Overall, we believe our foray into accessibility technology to be a success and a user guide to the functioning of this device can be found at the very end of this report in Appendix K.

## Recommendations

If we were to, knowing what we know now, complete this project a second time, we would definitely keep the same general approach incorporating an IMU, accelerometers, and a sip-puff sensor. However, we would definitely make different choices in our implementation of that paradigm.

Firstly, we would use a faster microcontroller. Even though the Arduino Due is the fastest device in the Arduino product line, it still wasn't able to collect and process data as fast as we would have liked, and still experienced some noticeable delay when performing complex calculations. Performance concerns can also be addressed by moving some of the computational complexity to the circuitry, perhaps replacing the accelerometers' averaging routine with a well-chosen filtering capacitor tied to the output. The sip-puff state machine could even, theoretically, have been outsourced to an ATTiny microcontroller, a chip with a very small physical footprint that could have been included in the sip-puff module, allowing the module to send interrupt signals to the Due.

The second area we would do differently a second time is the electrical manufacturing. Although we managed to fare just fine using protoboards and jumper wires, if our group had possessed more time and experience in the field of electrical manufacturing, we might have been able to design, etch, and reflow-solder PCBs for the relevant components, allowing us to directly solder our plugs to the board, as well as fully integrate devices that we could otherwise only access from breakout boards. This would greatly improve the elegance of the design.

# Acknowledgements

We would first like to acknowledge our Advisor, Dr. Lynne Slivovsky, who was of great help throughout the design process of this project. She was always there to lend a helping hand when needed and always tried to assist us in any manner possible.

We would also like to thank Dr. J Kevin Taylor for providing us with the opportunity to create something useful for a person in need and giving us this amazing opportunity; and for providing us with the funds to complete this Senior Project without hesitating too much about cost and whether or not this project was going to be within budget.

We would like to acknowledge our client and user, Brook McCall, for being patient with us and cooperating with us from beginning to end. We had many other clients prior to Brook who fell through, but Brook stayed with us throughout the entire project from start to finish, and for that, we would like to thank her.

We would like to thank Kimberly Jones, who provided us with materials/resources on how to interact with people diagnosed with quadriplegia and for answering any questions regarding our device and its usefulness to a person diagnosed with quadriplegia.

Lastly, we would especially like to thank our friends and families whom also have been impacted by this project with all the cancelled plans, re-scheduled dates and gatherings, and late nights. We thank you all for the friendship and support of each and every one of you throughout the duration of this project.

# Appendix A: References

1. Code of Federal Regulations. Title 21, Volume 8. Revised as of April 1, 2012. CITE: 21CFR872. (http://www.accessdata.fda.gov/scripts/cdrh/cfdocs/cfCFR/CFRSearch.cfm?CFRPart=872&showFR=1)

2. Dragon Speech Recognition Software. Computer software. Dragon. Nuance, 2013. Web. 1 Nov. 2013. <http://www.nuance.com/dragon/index.htm>.

3. "Headmouse Extreme." Shop Origin Instruments. Origin Instruments, 2011. Web. 1 Nov. 2013. <http://shop.orin.com/shop/index.php?main_page=product_info>.

4. Hollingshead, Todd. "'Eagle Eyes' Enable Users to Soar." The Salt Lake Tribune 31 Jan. 2006: n. pag. Boston College. Boston College- Eagle Eyes Project, 27 Sept. 2010. Web. 4 Nov. 2013.

5. Official Journal of the European Union (2011): 23. Print.

6. "Paralysis, Paraplegia, and Quadriplegia." Paralysis Paraplegia And Quadriplegia. American Medical Association, 2013. Web. 18 Oct. 2013.

7. "Quad Control." Quad Control: Video Game Controls for Quadriplegics. QuadControl, 2010. Web. 3 Nov. 2013. <http://quadcontrol.com/joysticks/xbox-360-controller>.

8. "Quasimoto Interactive." Quasimoto Interactive. N.p., 2013. Web. 19 Oct. 2013.

9. "Revolutionary Gaming Device for Quadriplegic People." Revolutionary Gaming Device for Quadriplegic People. Makeymakey.com, 02 June 2012. Web. 16 Oct. 2013.

10. "TrackerPro." Able Net. Able Net, 2013. Web. 3 Nov. 2013. <http://www.ablenetinc.com/Assistive-Technology/Computer-Access/TrackerPro>.

11. Umbehr, Josh. "GRAViTONUS® Gaming System For Quadriplegics." GRAViTONUS® Gaming System For Quadriplegics. MedGadget, 13 Feb. 2007. Web. 19 Oct. 2013.

12. Jason, Webb. N.p., n. d. 13 Feb 2014. <http://jason-webb.info/wiki/index.php?title=OpenSip+Puff>.

# Appendix B: Engineering Specifications Table

| Spec. # | Parameter Description | Target Value | Tolerance | Risk | Compliance |
|---|---|---|---|---|---|
| **1** | Weight applied to user | 5 N | Max | M | A, T, S |
| **2** | Power Consumption | 2.5 W | Max | M | A, T |
| **3** | Manufacturing Cost per Unit | $500 | Max | L | A |
| **4** | Supply Voltage | 12 V | Max | H | A, I |
| **5** | Input Latency | 50 ms | Max | H | T, I, S |
| **6** | Must be able to maintain functionality after being dropped onto a wooden surface from a height of | 1 m | Min | M | A, I |
| **7** | Must not pierce, abrade or lacerate skin when used for a duration of | 8 hours | Minimum | H | T, I |
| **8** | Must not expose user to any hazardous materials, edges or voltages | None | At all | H | A, I |
| **9** | Must be able to be set up by user's assistant in under | 10 min | Max | M | T, A |
| **10** | Mean time to failure | 7300 hours | Min | M | A |
| **11** | Must be RoHS compliant | True | Equals | H | A |
| **12** | Must not require invasive contact with inside of user's | True | Equals | M | A |

| | | | | | |
|---|---|---|---|---|---|
| | mouth | | | | |
| **13** | Must be usable by articulating only one's head, shoulders and neck | True | Equals | H | A, T |
| **14** | Must only signal inputs to the computer when intended by the user | True | Equals | H | A, T |
| **15** | Must interface with a PC as an input device | True | Equals | H | A, T |
| **16** | Must be considered comfortable with _____ % of users | 80% | Min | M | A, T |
| **17.** | Cursor input position shall be able to be articulated independently from and simultaneously with button inputs | True | Equals | M | A, T |
| **18.** | Must be able to activate Control, Shift, and Left or Right Click either simultaneously or independently | True | Equals | M | A, T |
| **19.** | Must be able to click and hold left or right mouse button inputs | True | Equals | M | A, T |

# Appendix C: House of Quality (QFD)

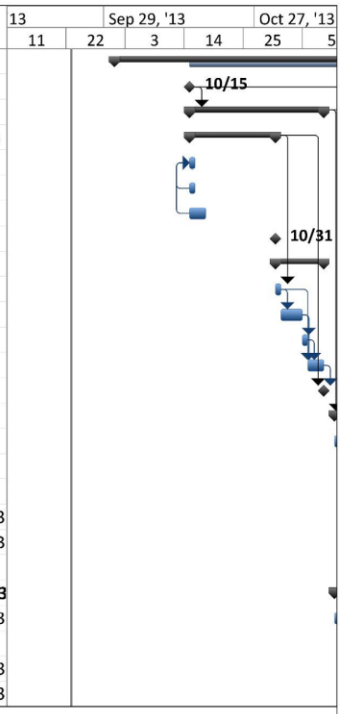
| Quad Gaming<br>Customer (Step #1) Requirements (Whats) | | Weighting (Total 100) | Engineering Specifications (HOWS): Must interface with PC as an input device | Must apply no more than 5 N of weight to the user | Must not use more than 2.5W of power | Must not pierce, abraide, or lacerate skin during after extended use (<8 hours of use) | No power supply required greater than 12 volts | Must be considered comfortable by>= 80% of users | mean time to failure is >= 7300 hours | Must be RoHS compliant | Must not break when dropped from 1 m onto a wooden surface | Must take no more than 10 minutes for an assistant to set up | Must have latency < 50ms | Must cost no more than $500 to manufacture in quantities of 1000 | system will be insulated to protect user from potential shock | System must not contact inside of user's mouth | System usable by persons who can only use their shoulders, neck, and head | Inputs will only be activated when intended by the user | Cursor input position shall be able to be articulated independently from and simultaneously with button inputs. | Must be able to activate Control, Shift, and Left or Right Click either simultaneously or independently. | Must be able to click and hold left or right mouse button inputs | Benchmarks: Dragon Naturally Speaking | HeadMouse® Extreme Deluxe | QuadControl |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Will allow user to draw on screen, other than straight lines | | 3 | | | | | | | | | | 2 | | | | | 2 | 1 | 3 | 3 | 0 | 0 | 0 |
| | Can be used with a computer | | 2 | | | | | | | | | | | | | | | 2 | | | | 4 | 4 | 2 |
| | Safety | | | 3 | 1 | 3 | | | 1 | 3 | | | | | 3 | 2 | | | | | | 4 | 4 | 4 |
| | Comfort | | | 3 | | 3 | | 3 | | | | | | | | 2 | | | | | | 4 | 3 | 4 |
| | Durability | | | 1 | | | | | 3 | | 3 | | | | | 1 | | | | | | 4 | 4 | 3 |
| | Must allow user to organize computer files | | 3 | | | | | | | | | | 1 | | | | | 2 | 3 | 3 | 3 | 1 | 0 | 0 |
| | Portable | | | 2 | 2 | | 2 | | | | | 3 | | | | | | | | | | 4 | 4 | 2 |
| | Easy to set up | | | 1 | 1 | | 2 | | | | | 3 | | | | | | | | | | 2 | 3 | 3 |
| | Must allow user to "click and drag" simultaneously | | 3 | | | | | | | | | | 1 | | | | | 3 | 3 | 3 | 3 | 1 | 0 | 0 |
| | Sufficient accuracy to use as mouse and/or joystick | | | | | | | | | | | | | | | | | 3 | | | | 1 | 3 | 4 |
| | Reliable inputs (no false + or -) | | | | | | | | | | | | | | | | | 3 | | | | 3 | 3 | 4 |
| | Functions using only shoulders, neck, and head | | | | | | | | | | | | | | | | 3 | | | | | 4 | 4 | 4 |
| | No invasive mouth-based components | | | | | | | 2 | | | | | | | | 3 | | | | | | 4 | 4 | 0 |
| | Must not cause muscle strain when used as intended | | | 3 | | | | | | | | | | | | | | | | | | 4 | 4 | 4 |
| | Must be usable for basic computer tasks besides gaming | | 2 | | | | | | | | | | 2 | | | | | 3 | | | | 4 | 4 | 1 |
| | Affordable | | | | | | | | | | | | | 3 | | | | | | | | 3 | 0 | 4 |
| | Units | | Yes/No | Newtons | Watts | Yes/No | Volts | % | Thousand Hours | Yes/No | Meters | Minutes | Milliseconds | USD | Yes/No | Yes/No | Yes/No | Yes/No | | | | | | |
| | Targets | | Yes | 10 | 5 | Yes | 12 | 90 | 7.3 | Yes | 1.5 | 10 | 50 | 500 | Yes | No | Yes | Yes | | | | | | |
| | Benchmark #1 | | Yes | | | | | | | | | | | 75 | | Yes | Yes | | | | | | | |
| | Benchmark #2 | | Yes | 13 | | | | | | | | | | 995 | | Yes | Yes | | | | | | | |
| | Benchmark #3 | | No | | | | | | | | | | | 322 | | No | Yes | | | | | | | |

| | |
|---|---|
| ● = 3 | Strong Correlation |
| ○ = 2 | Medium Correlation |
| Δ = 1 | Small Correlation |
| Blank | No Correlation |
| benchmarks: | |
| | |
| 0 = poor | |
| 4 = great | |

# Appendix D: Gannt Chart

Note that since Microsoft Project is not very compatible with most sane printing configurations, the exported view of the Gannt chart is rather scattered.

| ID | Task Name | Duration | Start | Finish |
|---|---|---|---|---|
| 1 | **Quad Gaming Project** | **179 days** | **Tue 10/1/13** | **Fri 6/6/14** |
| 2 | Team Contract | 0 days | Tue 10/15/13 | Tue 10/15/13 |
| 3 | **Preliminary Research** | **19 days** | **Tue 10/15/13** | **Fri 11/8/13** |
| 4 | **Background Research** | **12 days** | **Tue 10/15/13** | **Thu 10/31/13** |
| 5 | Research previous senior projects | 1 day | Tue 10/15/13 | Tue 10/15/13 |
| 6 | Patent Search | 1 day | Tue 10/15/13 | Tue 10/15/13 |
| 7 | Products with similar outcomes on market | 3 days | Tue 10/15/13 | Thu 10/17/13 |
| 8 | Meet with Client | 0 days | Thu 10/31/13 | Thu 10/31/13 |
| 9 | **Project Requirements** | **7 days** | **Thu 10/31/13** | **Fri 11/8/13** |
| 10 | Understand Customer Requirements | 1 day | Thu 10/31/13 | Thu 10/31/13 |
| 11 | Derive Engineering Specifications | 2 days | Fri 11/1/13 | Mon 11/4/13 |
| 12 | Complete QFD | 1 day | Tue 11/5/13 | Tue 11/5/13 |
| 13 | Write Project Requirements Report | 3 days | Wed 11/6/13 | Fri 11/8/13 |
| 14 | Project Requirements Due | 0 days | Fri 11/8/13 | Fri 11/8/13 |
| 15 | **Conceptual Design** | **56 days** | **Mon 11/11/13** | **Tue 1/28/14** |
| 16 | Brainstorm Possible Solutions | 2 days | Mon 11/11/13 | Tue 11/12/13 |
| 17 | Utilize Morphological Matrices | 2 days | Wed 11/13/13 | Thu 11/14/13 |
| 18 | **Utilize Decision Matrices and Morphological Matrices** | **6 days** | **Fri 11/15/13** | **Fri 11/22/13** |
| 19 | Hardware | 2 days | Fri 11/15/13 | Mon 11/18/13 |
| 20 | Mounting | 2 days | Tue 11/19/13 | Wed 11/20/13 |
| 21 | Final Design | 2 days | Thu 11/21/13 | Fri 11/22/13 |
| 22 | **Material Decisions** | **3 days** | **Mon 11/11/13** | **Wed 11/13/13** |
| 23 | Background Research on Materials | 1 day | Mon 11/11/13 | Mon 11/11/13 |
| 24 | Decision Matrices for Materials | 1 day | Tue 11/12/13 | Tue 11/12/13 |
| 25 | Cost Analysis for Top Design Ideas | 1 day | Wed 11/13/13 | Wed 11/13/13 |
| 26 | Write Conceptual Design Report | 1 day | Mon 11/25/13 | Mon 11/25/13 |

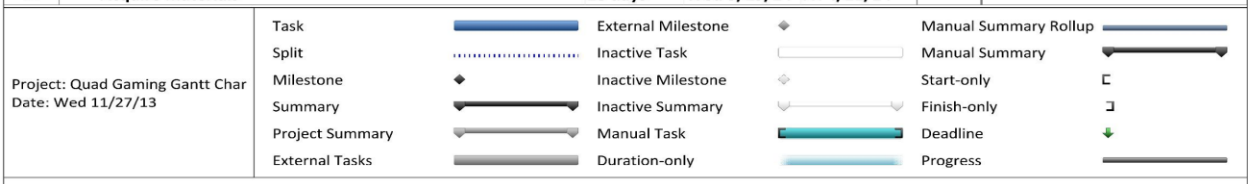



Project: Quad Gaming Gantt Char
Date: Wed 11/27/13

| | | |
|---|---|---|
| Task | External Milestone | Manual Summary Rollup |
| Split | Inactive Task | Manual Summary |
| Milestone | Inactive Milestone | Start-only |
| Summary | Inactive Summary | Finish-only |
| Project Summary | Manual Task | Deadline |
| External Tasks | Duration-only | Progress |

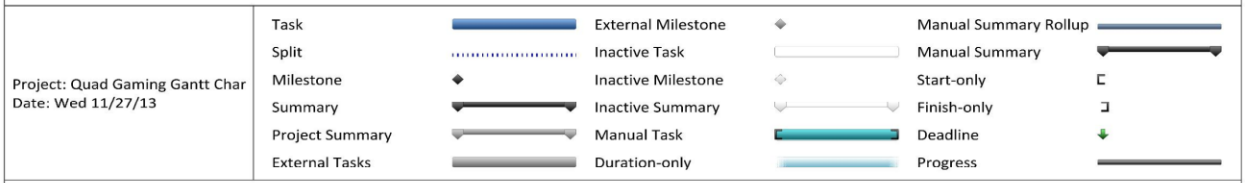
Page 1

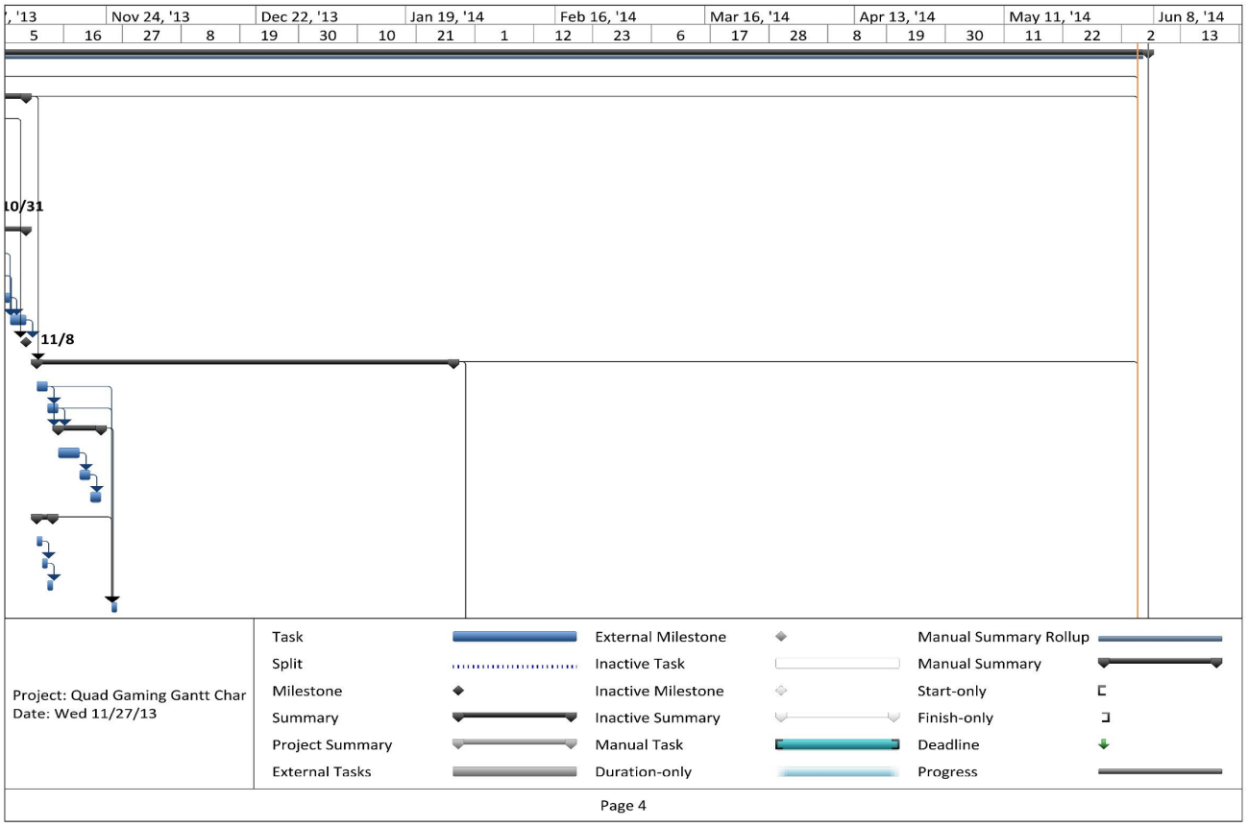
| ID | Task Name | Duration | Start | Finish |
|---|---|---|---|---|
| 27 | Conceptual Design Presentation | 3 days | Mon 11/11/13 | Wed 11/13/13 |
| 28 | Conceptual Design Report Due | 0 days | Tue 12/10/13 | Tue 12/10/13 |
| 29 | Write Critical Design Review Report | 10 days | Tue 1/14/14 | Mon 1/27/14 |
| 30 | Critical Design Report Due | 0 days | Tue 1/28/14 | Tue 1/28/14 |
| 31 | **Prototype Development** | **31 days** | **Tue 1/28/14** | **Tue 3/11/14** |
| 32 | **Electronics Hardware** | **14 days** | **Tue 1/28/14** | **Fri 2/14/14** |
| 33 | Design Needed Circuits | 4 days | Tue 1/28/14 | Fri 1/31/14 |
| 34 | Acquire Materials | 8 days | Mon 2/3/14 | Wed 2/12/14 |
| 35 | Assemble Components | 2 days | Thu 2/13/14 | Fri 2/14/14 |
| 36 | **Software Development and Implementation** | **7 days** | **Tue 1/28/14** | **Wed 2/5/14** |
| 37 | Design Programming for Microcontroller | 4 days | Tue 1/28/14 | Fri 1/31/14 |
| 38 | Upload and Troubleshoot Hardware Programing | 3 days | Mon 2/3/14 | Wed 2/5/14 |
| 39 | **Develop Mounting System** | **11 days** | **Mon 2/17/14** | **Mon 3/3/14** |
| 40 | Design Mount | 2 days | Mon 2/17/14 | Tue 2/18/14 |
| 41 | Aquire Materials | 6 days | Wed 2/19/14 | Wed 2/26/14 |
| 42 | Assemble Mount | 3 days | Thu 2/27/14 | Mon 3/3/14 |
| 43 | Patent Disclosure | 0 days | Tue 2/11/14 | Tue 2/11/14 |
| 44 | Assemble Hardware and Mounting System | 2 days | Tue 3/4/14 | Wed 3/5/14 |
| 45 | Configure PC Software to Correctly Interpret Input for Various Applications | 2 days | Thu 3/6/14 | Fri 3/7/14 |
| 46 | Review success of prototype | 2 days | Mon 3/10/14 | Tue 3/11/14 |
| 47 | Project Update Presentation | 0 days | Tue 3/11/14 | Tue 3/11/14 |
| 48 | Project Update Memo to Sponsor | 0 days | Tue 3/11/14 | Tue 3/11/14 |
| 49 | **Research company with best prices for materials** | **29 days** | **Wed 3/12/14** | **Tue 4/22/14** |
| 50 | Research Manufacturing Methods | 5 days | Wed 3/12/14 | Tue 3/18/14 |
| 51 | **Acquire materials** | **18 days** | **Wed 3/19/14** | **Fri 4/11/14** |

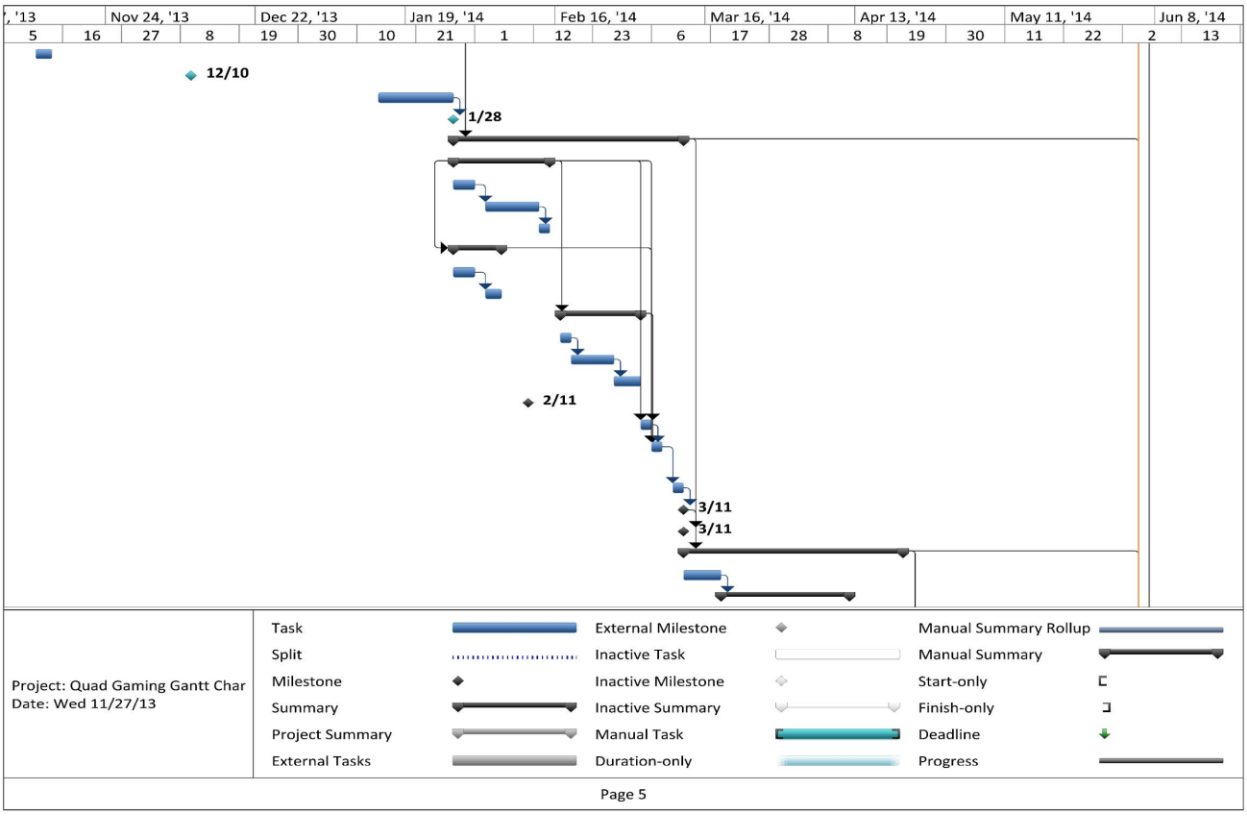


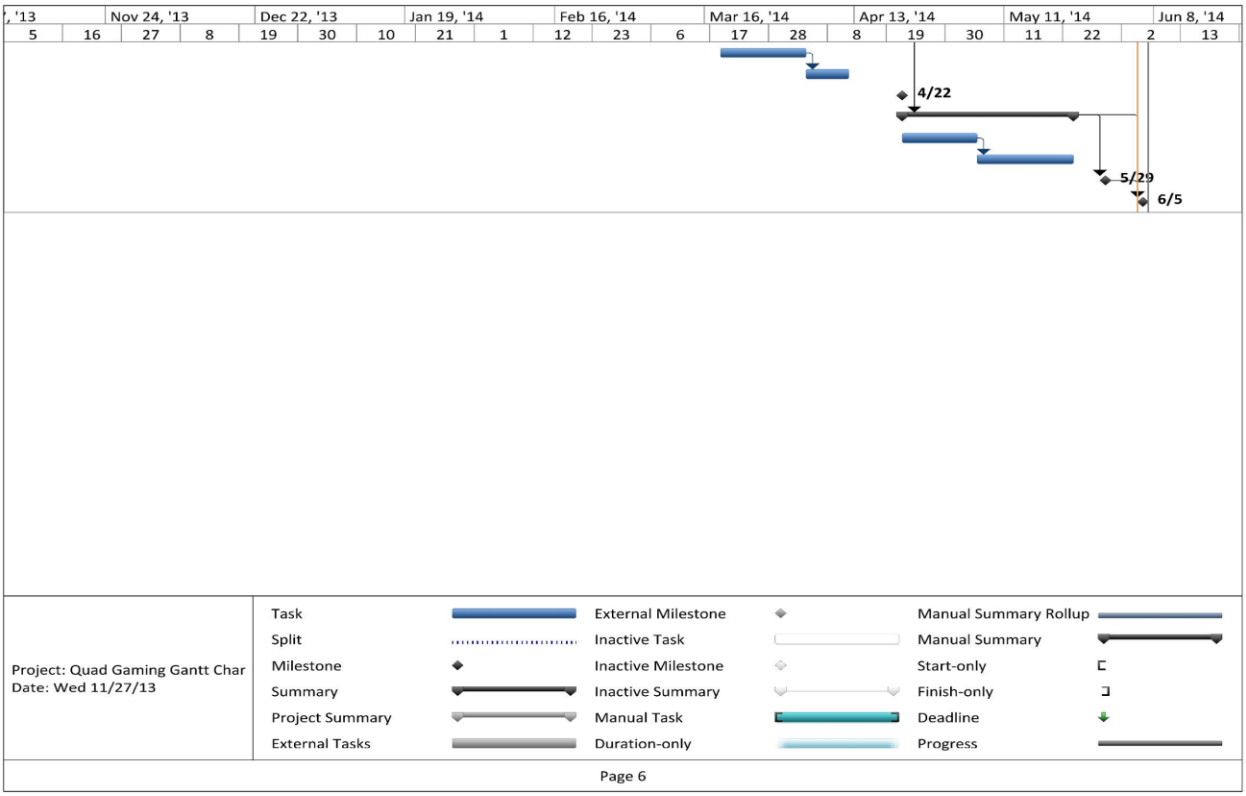



| ID | Task Name | Duration | Start | Finish |
|---|---|---|---|---|
| 52 | Obtain Components | 12 days | Wed 3/19/14 | Thu 4/3/14 |
| 53 | Manufacture Assemblies | 6 days | Fri 4/4/14 | Fri 4/11/14 |
| 54 | Hardware and Software Review | 0 days | Tue 4/22/14 | Tue 4/22/14 |
| 55 | **Checkpoints for failures** | **24 days** | **Tue 4/22/14** | **Fri 5/23/14** |
| 56 | Hardware and Software Testing and Troubleshooting | 10 days | Tue 4/22/14 | Mon 5/5/14 |
| 57 | Final Product Testing | 14 days | Tue 5/6/14 | Fri 5/23/14 |
| 58 | Senior Project Expo | 0 days | Thu 5/29/14 | Thu 5/29/14 |
| 59 | Final Project Report Due | 0 days | Thu 6/5/14 | Thu 6/5/14 |




, '13
Nov 24, '13
Dec 22, '13
Jan 19, '14
Feb 16, '14
Mar 16, '14
Apr 13, '14
May 11, '14
Jun 8, '14
5 16 27 8 19 30 10 21 1 12 23 6 17 28 8 19 30 11 22 2 13
10/31
11/8
Project: Quad Gaming Gantt Char
Date: Wed 11/27/13
Task
Split
Milestone
Summary
Project Summary
External Tasks
External Milestone
Inactive Task
Inactive Milestone
Inactive Summary
Manual Task
Duration-only
Manual Summary Rollup
Manual Summary
Start-only
Finish-only
Deadline
Progress
Page 4


, '13
Nov 24, '13
Dec 22, '13
Jan 19, '14
Feb 16, '14
Mar 16, '14
Apr 13, '14
May 11, '14
Jun 8, '14
5 16 27 8 19 30 10 21 1 12 23 6 17 28 8 19 30 11 22 2 13
12/10
1/28
2/11
3/11
3/11
Project: Quad Gaming Gantt Char
Date: Wed 11/27/13
Task
Split
Milestone
Summary
Project Summary
External Tasks
External Milestone
Inactive Task
Inactive Milestone
Inactive Summary
Manual Task
Duration-only
Manual Summary Rollup
Manual Summary
Start-only
Finish-only
Deadline
Progress
Page 5


', '13
Nov 24, '13
Dec 22, '13
Jan 19, '14
Feb 16, '14
Mar 16, '14
Apr 13, '14
May 11, '14
Jun 8, '14
5 16 27 8 19 30 10 21 1 12 23 6 17 28 8 19 30 11 22 2 13
4/22
5/29
6/5
Project: Quad Gaming Gantt Char
Date: Wed 11/27/13
Task
Split
Milestone
Summary
Project Summary
External Tasks
External Milestone
Inactive Task
Inactive Milestone
Inactive Summary
Manual Task
Duration-only
Manual Summary Rollup
Manual Summary
Start-only
Finish-only
Deadline
Progress
Page 6

# Appendix E: Brainstorming Session #1

| | Ideas | feasibility (0-3) |
|---|---|---|
| 1 | eye tracking | 2 |
| 2 | concept similar to makey makey product with buttons/headgear | 1 |
| 3 | more accurate voice command system | 2 |
| 4 | myoelectric sensors | 2 |
| 5 | EEG | 2 |
| 6 | IMU | 3 |
| 7 | Portable FMRI | 0 |
| 8 | shoulder control with buttons,sensors | 2 |
| 9 | chin straps/headstraps with myoelectrodes | 2 |
| 10 | Facial expression monitoring camera | 1 |
| 11 | surgery to implant mind reading chip | 0 |
| 12 | robot controlled through voice command | 1 |
| 13 | Rube Goldberg device | 0 |
| 14 | target tracking using reflective dots | 2 |
| 15 | chin joystick/buttons | 2 |
| 16 | myoelectric face mask (breatheable) | 2 |

# Appendix F: Brainstorming session #2

| | |
|---|---|
| What if we needed to design it cheaply? | |
| | utilize free eye tracking software for joystick and mouse |
| | use normal unadaptive controller |
| | utilize free eye tracking software along with ~$10 accelerometer + ~$50 gyroscope |
| | strings to pull with face and shoulders and chin band with rod glued to it to push buttons on controller |
| What if we had $1,000,000 to make this? | |
| | Research + Funds for mind reading (or EEG) |
| | attach wires to user's arms to detect nerve signals and move arms for them using metal encasings for "gloves" attached to user |
| | robot that listnes through voice commmand |
| | hire someone to play games for them |
| What if we can't use custom electronics? | |
| | Free voice command software paired with chin joysticks |
| | strings to pull with face and shoulders and chin band with rod glued to it to push buttons |
| | utiliza gyro. Mouse + attach to top of head + string and rods attached to joystick/buttons |
| What if we could use tongue,lip,bite interface? | |
| | bite switches combined with tongue pressure sensor (previous senior project) |
| | mouth joystick with sip 'n' puff switches |
| | negates voice commands |
| | mouth joystick with myoelectric sensors electrodes |

# Appendix G: Detailed Supporting Analysis

## Sip-Puff Transducer Instrument Amplification Circuit

We analyzed the circuit with a gain resistor of 1.3 kΩ, performing the math by hand as follows.

$$\frac{V_{out}}{\Delta V} = \left(1 + \frac{2 \times R_{gain}}{R_1}\right)\frac{R_3}{R_2}$$

The instrumentation amplifier we employed is the AD8223, which has in-built values of 8kΩ, 10kΩ, and 50kΩ for $R_1$, $R_2$, and $R_3$, respectively. We inserted those values into the equation and simplified it to arrive at the following equation relating the amplifier's output voltage to the voltage differential between the two inputs.

$$V_{out} = 62.5 \times \Delta V$$

Using this equation, we can then find the range of input voltage differentials that will result in valid, defined outputs.

$$\Delta V_{min} = 0\ mV$$

$$\Delta V_{max} = 52.8\ mV$$

Referencing the datasheet of the MPX12GP, we see that the voltage differential idles around 26 mV, and tends to vary by about ±20 mV, which fits it perfectly within our desired input voltage range.

# Appendix H: Vendor Supplied Component Specifications and Data Sheets

Arduino Due:

Atmel SAM3X8E ARM Cortex-M3 CPU:

http://www.atmel.com/Images/doc11057.pdf

Board Schematics:

http://arduino.cc/en/uploads/Main/arduino-Due-schematic.pdf

9 DoF IMU (MPU-9150):

MPU-9150 Chip Documentation:

http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/IMU/PS-MPU-9150A.pdf

MPU-9150 Registry Maps:

http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/IMU/RM-MPU-9150A-00.pdf

Breakout Board Schematics:

http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/IMU/mpu-9150_breakout.pdf

**3-axis Accelerometer (MMA8452Q) Documentation:**

**MMA8452Q Chip Documentation:**

http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/Accelerometers/MMA8452Q.pdf

Breakout Board Schematics:

http://dlnmh9ip6v2uc.cloudfront.net/datasheets/Sensors/Accelerometers/MMA8452Q-Breakout-v11-fixed.pdf

**Pressure Sensor (MX12GP) Documentation:**

http://www.freescale.com/files/sensors/doc/data_sheet/MPX12.pdf

Instumentation Amplifier (AD8223) Documentation:

http://www.analog.com/static/imported-files/data_sheets/AD8223.pdf

# Appendix I: List of Vendors, Contact Info, and Pricing

| Primary Components | Qty | Unit | Price/Unit | Supplier |
|---|---|---|---|---|
| **Arduino Due** | 1 | each | 40.00 | Amazon |
| **9 DoF IMU (MPU-9150)** | 1 | each | 40.00 | Sparkfun |
| **3-axis Accelerometer (MMA8452Q)** | 1 | each | 10.00 | Sparkfun |
| **Pressure Sensor (MX12GP)** | 1 | each | 8.69 | Digi-Key |
| **Instumentation Amplifier (AD8223)** | 1 | each | 1.50 | Digi-Key |
| **Pressure Sensor Attachment (1x4 female header)** | 1 | each | 0.73 | Digi-Key |
| **1.3k Ohm Gain Resistor** | 1 | each | 0.10 | Digi-Key |
| **Velcro Armbands for Accelerometers** | 2 | each | Estimated: $3.00 | Manufactured by Team |
| **Main Electronics Enclosure** | 1 | each | Free+Shipping ($3.95) | Takachi-Enclosure |
| **IMU and Sip-Puff Enclosure** | 1 | each | Free+Shipping ($3.95) | Takachi-Enclosure |
| **Schmartboard Protoboard for Sip-Puff Electronics** | 1 | each | $10.00 | Radio Shack |
| **3/16" Clear Vinyl Tubing** | 10 | ft | $0.44 | Home Depot |
| **Xbox 360 Chat Headset** | 1 | each | Free | Supplied from Group Member |
| **Jumpers and Connection Wire** | 20 | ft | Free | Supplied from Group Member |
| **Solder** | 0.25 | oz | Free | Supplied from Group Member |
| | Approximate Total | | $130.05 | |

# Appendix J: Estimated Production Cost

(Price based on bulk order of 1000 units)

**Headset Electronics**

1x MPU 6050 IMU - $4.62/ea (http://store.invensense.com/ProductDetail/MPU6050-InvenSense-Inc/422200&pid=1135)

1x MPX12GP Pressure Sensor - $6.28/ea (http://www.digikey.com/product-search/en?vendor=0&keywords=mpx12gp)

1x AD8223 Instrumentation Amplifier - $1.52/ea (http://www.digikey.com/product-detail/en/AD8223ARMZ-R7/AD8223ARMZ-R7TR-ND/1979373)

1x 2k ohm 5% SMD Resistor - $0.002/ea (http://www.digikey.com/product-detail/en/RC0603JR-072KL/311-2.0KGRCT-ND/729673)

1x Female Connector 1x4 Right Angle Connector - $0.309 (http://www.digikey.com/product-detail/en/PPTC041LGBN-RC/S5440-ND/775898)

**Microcontroller**

ATmega32u4 Microcontroller - $3.54/ea (http://www.digikey.com/product-detail/en/ATMEGA32U4-MU/ATMEGA32U4-MU-ND/1914603)

2x 22 ohm 5% SMD Resistor - $0.005/ea (http://www.digikey.com/product-detail/en/ERJ-6GEYJ220V/P22ACT-ND/87316)

1x 82 ohm 5% SMD Resistor - $0.005/ea (http://www.digikey.com/product-detail/en/ERJ-6GEYJ820V/P82ACT-ND/42830)

3x 10k ohm 5% SMD Resistor - $0.001/ea (http://www.digikey.com/product-detail/en/RC0603JR-0710KL/311-10KGRTR-ND/726700)

1x 1.0µF 10% SMD Capacitor - $0.025/ea (http://www.digikey.com/product-detail/en/C2012X7R1C105K125AA/445-1358-1-ND/567583)

3x 0.1µF 10% SMD Capacitor - $0.037/ea (http://www.digikey.com/product-detail/en/C2012X8R1H104K125AA/445-2514-1-ND/927171)

1x 10µF 10% SMD Capacitor - $0.119/ea (http://www.digikey.com/product-detail/en/T491A106M016AT/399-3687-1-ND/819012)

1x 16MHz PTH Resonator - $0.224/ea (http://www.digikey.com/product-detail/en/CSTLS16M0X51-A0/490-5998-1-ND/3845198)

1x 500mA PTC Resettable SMD Fuse - $0.165/ea (http://www.digikey.com/product-detail/en/PTS120616V025/283-3137-1-ND/2639169)

1x USB mini B PTH - $0.828/ea (http://www.digikey.com/product-detail/en/0548190519/WM17115-ND/773802)

1x Green SMD LED - $0.056/ea (http://www.digikey.com/product-detail/en/LTST-C171GKT/160-1423-1-ND/386792)

2x 10k PTH Trimpot - $0.533/ea (http://www.digikey.com/product-detail/en/3362P-1-103LF/3362P-103LF-ND/1088412)

2x 6-conductor RJ-25 Modular Jack - $0.858/ea (http://www.digikey.com/product-search/en?WT.z_header=search_go&lang=en&site=us&keywords=A31406-ND&x=0&y=0&formaction=on)

1x 6-conductor RJ-25 7 ft. Cable - $1.22/ea (http://www.digikey.com/product-search/en?WT.z_header=search_go&lang=en&site=us&keywords=A1662R-07-ND&x=0&y=0&formaction=on)

2 Custom Printed Circuit boards (Dual Layered at $1/sq. in. from OSH Park)

Headset PCB: approximately 1.5” by 2” = $3.50

Minimized Microcontroller PCB: approximately 2” by 3” = $6.00

Estimated Cost of Custom Injection Molded Microcontroller Enclosure: $2.00

Estimated Cost of Custom Headset - $5.00

**Total Estimated Cost of Parts: $25.51**

# Appendix K: User Guide

User Manual for Hands-Free Computer Mouse

I. Installation

A. Install Software

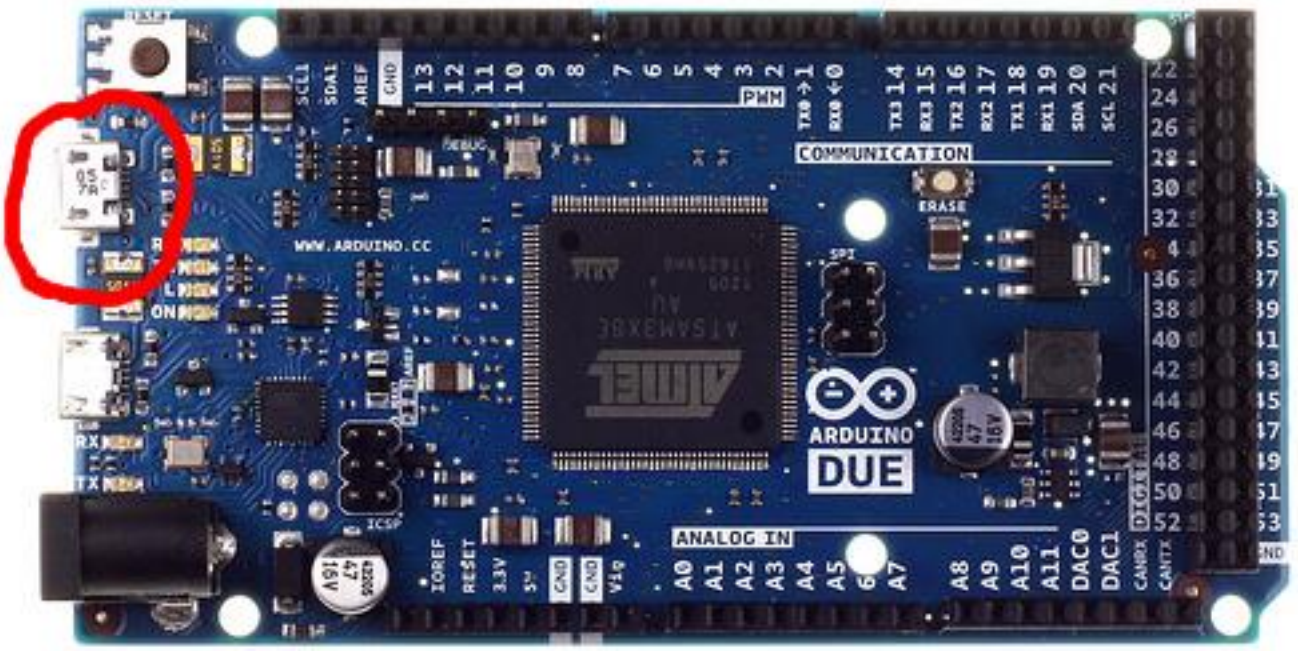
1. Go to http://downloads.arduino.cc/arduino-1.5.6-r2-windows.exe. A file will download.

2. Run the installer you just downloaded. Click “Yes” on all prompts.

II. Set-Up

A. Plug in Device

1. Plug the small end of the supplied USB cable to the port nearest the Reset button (circled in red the picture below)




2. Plug the other end of the cable into any USB port on your computer.

B. Put on Headset

1. Put the headset on your head as though it were a pair of headphones.

2. Adjust the tube attached to the microphone boom so that it is easily accessible form the corner of your mouth.

3. Connect the headset to the device using the supplied phone cable.

C. Press the Reset button.

1. The device should now be working.

2. To move the mouse, move your head.

3. To left-click, puff on the tube.

4. To right-click, sip on the tube.

5. To turn off the device, simply un-plug it.

III. Configuration

A. Adjusting mouse sensitivity.

1. To adjust how sensitive the mouse movement is to movement if your head, use a small flat-head screwdriver to turn Adjustment Knob 1.

a. Turning it clockwise will increase sensitivity

b. Turning it counter-clockwise will decrease sensitivity.

2. If the range of adjustment from the knob is not sufficient, you can further adjust the sensitivity in your PC's mouse settings panel.

B. Adjusting click sensitivity.

1. To adjust how hard you must puff or sip on the tube to register a left or right click, use a small flat-head screwdriver to turn Adjustment Knob 2.

a. Turning it clockwise will increase sensitivity

b. Turning it counter-clockwise will decrease sensitivity.

C. Adjusting shoulder sensor sensitivity.

1. Depending on the angle your arms make with respect to the ground when you are sitting down, you may have to adjust the resting level of the sensor system. To do so, use a small flat-head screwdriver to turn Adjustment Knob 3.

a. Adjust and test the setting until you are able to consistently press and release the Control and Shift keys.

# Appendix L: Program Code

Main Program:

```
//////////////////////////////////////////////////////////////////////////
// quad_computing.ino

#include <address.h>
#include <adk.h>
#include <confdescparser.h>
#include <hid.h>
#include <hidboot.h>
#include <hidusagestr.h>
#include <KeyboardController.h>
#include <MouseController.h>
#include <parsetools.h>
#include <Usb.h>
#include <usb_ch9.h>
#include "Wire.h"
#include "I2Cdev.h"
#include "MPU6050.h"

#define SHIFT_ENABLE 1

#define SIP_APIN 0
#define CTRL_APIN 2
#define SHIFT_APIN 1

#define ADJ1_APIN 3
#define ADJ2_APIN 4
#define ADJ3_APIN 5

#define LED1_PIN 13
#define LED2_PIN 12
#define LED3_PIN 11
#define LED4_PIN 10

#define INTERVAL_US 200
#define ACCEL_INTERVAL_US 2000

#define DEAD_ZONE 35

#define ACCEL_NUMSAMPLES 8 // makes averaging easier
#define ACCEL_RANGE 25
```

```
#define ACCEL_REST 273 // tuner 2

#define SIP_RANGE 150 // tuner 3
#define SIP_REST 525

#define DEBUG 0

#if DEBUG == 1
  #define Print(x) Serial.print((x))
  #define Println(x) Serial.println((x))
#else
  #define Print(x) 0
  #define Println(x) 0
#endif

MPU6050 imu;

unsigned short ctrlData[ACCEL_NUMSAMPLES];
unsigned short shiftData[ACCEL_NUMSAMPLES];

void die() {
  digitalWrite(LED1_PIN, HIGH);
  digitalWrite(LED2_PIN, HIGH);
  digitalWrite(LED3_PIN, HIGH);
  digitalWrite(LED4_PIN, HIGH);
  while(1) {delay(1000);}
}

void setup() {
  Wire.begin();
  #if DEBUG == 1
    Serial.begin(19200);
  #endif

  pinMode(LED1_PIN, OUTPUT);
  pinMode(LED2_PIN, OUTPUT);
  pinMode(LED3_PIN, OUTPUT);
  pinMode(LED4_PIN, OUTPUT);

  digitalWrite(LED1_PIN, LOW);
  digitalWrite(LED2_PIN, LOW);
  digitalWrite(LED3_PIN, LOW);
  digitalWrite(LED4_PIN, LOW);

  // initialize imu
  Println("Initializing I2C devices...");
```

```
  imu.initialize();
  // verify connection
  Println("Testing device connections...");
  bool connectionSuccess = imu.testConnection();
  Println(connectionSuccess ? "MPU6050 connection successful" : "MPU6050
connection failed");
  if (!connectionSuccess) {
    die();
  }

  imu.setFullScaleGyroRange(MPU6050_GYRO_FS_2000);
  if (imu.getFullScaleGyroRange() != MPU6050_GYRO_FS_2000) {
    die();
  }

  for (int i = 0; i < ACCEL_NUMSAMPLES; ++i) {
    ctrlData[i] = 0;
    shiftData[i] = 0;
  }

  Mouse.begin();
  Keyboard.begin();
}

void Accel();
unsigned short Average(unsigned short *ptr, int len);
void SipPuff();
void Gyro();

static int dt;

void loop() {
  static int us = 0;
  int tmp = micros();
  dt = tmp - us;
  us = tmp;
  Print("\t\tDT (us): "); Println(dt);

  Println("GYRO:");
  tmp = micros();
  Gyro();
  Print("\t\t\tGyro time: "); Println(micros() - tmp);
  Println("SIP/PUFF:");
  tmp = micros();
  SipPuff();
  Print("\t\t\tSipPuff time: "); Println(micros() - tmp);
```

```
  //Println("ACCEL:");
  //tmp = micros();
  //Accel();
  //Print("\t\t\tAccel time: "); Println(micros() - tmp);

  //delayMicroseconds(INTERVAL_US);
}
//Check performance, fix performance

void Gyro()
{
  signed int dx, dy;
  int16_t gx, gy, gz;
  unsigned short tweak = (analogRead(ADJ1_APIN) >> 8) + 1;
  Print("Gyro tweak: "); Println(tweak);

  // read raw accel/gyro measurements from device
  imu.getRotation(&gx, &gy, &gz);

  Print("\tx: "); Print(gx);
  Print("\ty: "); Print(gy);
  Print("\tz: "); Println(gz);

  gz = abs(gz) > DEAD_ZONE ? gz : 0;
  gy = abs(gy) > DEAD_ZONE ? gy : 0;
  Print("gy: "); Print(gy);

  dx = -1 * ((gy * tweak) >> 2);
  Print(", dx: "); Println(dx);
  dy = (gz * tweak) >> 2;

  //Mouse.move(abs(dx) > 2 ? dx : 0, abs(dy) > 2 ? dy : 0, 0);
  Mouse.move(dx, dy, 0);
}

#define NOMBDOWN 0
#define LMBDOWN 1
#define RMBDOWN 2
void SipPuff()
{
  int p; // pressure
  static int mbState = 0; // 0: no buttons down, 1: lmb down, 2: rmb down

  short tweak = ((signed short)analogRead(ADJ2_APIN) - 511) >> 5;
  unsigned short p_high = SIP_REST + SIP_RANGE + tweak; // add adjustment
  unsigned short p_low = SIP_REST - SIP_RANGE - tweak;
```

```
p = analogRead(SIP_APIN);
Print("Pressure = "); Print(p);
Print(" / p_high = "); Print(p_high);
Print(" / p_low = "); Println(p_low);

switch(mbState)
{
 case NOMBDOWN:
  if (p > p_high) {
   mbState = LMBDOWN;
   Mouse.press(MOUSE_LEFT);
   digitalWrite(LED1_PIN, HIGH);
  } else if (p < p_low) {
   mbState = RMBDOWN;
   Mouse.press(MOUSE_RIGHT);
   digitalWrite(LED2_PIN, HIGH);
  }
  break;

 case LMBDOWN:
  if (p < p_low) {
   mbState = RMBDOWN;
   Mouse.release(MOUSE_LEFT);
   digitalWrite(LED1_PIN, LOW);
   Mouse.press(MOUSE_RIGHT);
   digitalWrite(LED2_PIN, HIGH);
  } else if (p < p_high) {
   mbState = NOMBDOWN;
   Mouse.release(MOUSE_LEFT);
   digitalWrite(LED1_PIN, LOW);
  }
  break;

 case RMBDOWN:
  if (p > p_high) {
   mbState = LMBDOWN;
   Mouse.release(MOUSE_RIGHT);
   digitalWrite(LED2_PIN, LOW);
   Mouse.press(MOUSE_LEFT);
   digitalWrite(LED1_PIN, HIGH);
  } else if (p > p_low) {
   mbState = NOMBDOWN;
   Mouse.release(MOUSE_RIGHT);
   digitalWrite(LED2_PIN, LOW);
  }
```

```
    break;
  }
}

void Accel()
{
  //static int sclk = 0;
  static int dataNdx = 0;
  //static const int clkDiv = (int)((float)ACCEL_INTERVAL_US /
(float)INTERVAL_US);
  //static const int clkDiv = 5;
  static int ctrlCooldown = 0;
  static int shiftCooldown = 0;
  unsigned short tmp, accel, atweak, t_high, t_low;
  static int ctrlState = 0, shiftState = 0;
  static bool ctrlDown = false;
  static bool shiftDown = false;
  short tweak;

  //if (++sclk < clkDiv) {
   //Serial.print("SCLK: "); Serial.println(sclk);
   //return;
  //}
  //sclk = 0;

  atweak = analogRead(ADJ3_APIN);
  tweak = ((signed short)atweak - 511) >> 2;
  Print("Accel tweak: "); Print(atweak);
  Print(" => "); Println(tweak);
  t_high = ACCEL_REST + tweak + ACCEL_RANGE;
  t_low = ACCEL_REST + tweak - ACCEL_RANGE;

  tmp = analogRead(CTRL_APIN);
  Print("Ctrl Sample["); Print(dataNdx);
  Print("]: "); Println(tmp);
  ctrlData[dataNdx] = tmp;

  tmp = analogRead(SHIFT_APIN);
  shiftData[dataNdx] = tmp;
  //Serial.println("wololo");

  ++dataNdx;
  dataNdx %= ACCEL_NUMSAMPLES;
  //Serial.println("wololo");

  if (ctrlCooldown <= 0) {
```

```
//Serial.println("wololo");
  accel = Average(ctrlData, ACCEL_NUMSAMPLES);
  Print("Ctrl Average: "); Println(accel);

  switch (ctrlState) {
    case 0:
      if (accel < t_low + 20) {
        if (ctrlDown) {
          digitalWrite(LED3_PIN, LOW);
          Keyboard.release(KEY_LEFT_CTRL);
          Println("DING");
          ctrlDown = false;
        } else {
          digitalWrite(LED3_PIN, HIGH);
          Keyboard.press(KEY_LEFT_CTRL);
          Println("DONG");
          ctrlDown = true;
        }
        ctrlState = 1;
      }
      break;
    case 1:
      if (accel > t_high + 20) {
        ctrlState = 0;
      }
      break;
  }
} else {
  Print("CTRL Cooldown: "); Println(ctrlCooldown);
  --ctrlCooldown;
}

if (SHIFT_ENABLE && shiftCooldown <= 0) {
  accel = Average(shiftData, ACCEL_NUMSAMPLES);
  Print("Shift Average: "); Println(accel);

  switch (shiftState) {
    case 0:
      if (accel < t_low) {
        if (shiftDown) {
          digitalWrite(LED4_PIN, LOW);
          Keyboard.release(KEY_LEFT_SHIFT);
          Println("DING");
          shiftDown = false;
        } else {
          digitalWrite(LED4_PIN, HIGH);
```

```
        Keyboard.press(KEY_LEFT_SHIFT);
        Println("DONG");
        shiftDown = true;
       }
       shiftState = 1;
      }
      break;
    case 1:
      if (accel > t_high) {
       shiftState = 0;
      }
      break;
   }
 } else if (SHIFT_ENABLE) {
   Print("SHIFT Cooldown: "); Println(shiftCooldown);
   --shiftCooldown;
 }
}

unsigned short Average(unsigned short *ptr, int len) {
 unsigned int sum = 0;
 int num = len;

 for (; len > 0; --len) {
  sum += *++ptr;
 }

 return (unsigned short)(sum >> 3); // hardcoded numsamples at 8 so this work
}
////////////////////////////////////////////////////////////////////////////
```

Libraries:

```
////////////////////////////////////////////////////////////////////////////
//I2Cdev.cpp
// I2Cdev library collection - Main I2C device class
// Abstracts bit and byte I2C R/W functions into a convenient class
// 6/9/2012 by Jeff Rowberg <jeff@rowberg.net>
//
// Changelog:
//     2012-06-09 - fix major issue with reading > 32 bytes at a time with Arduino Wire
//                - add compiler warnings when using outdated or IDE or limited I2Cdev
implementation
//     2011-11-01 - fix write*Bits mask calculation (thanks sasquatch @ Arduino forums)
//     2011-10-03 - added automatic Arduino version detection for ease of use
//     2011-10-02 - added Gene Knight's NBWire TwoWire class implementation with
```

```
small modifications
//     2011-08-31 - added support for Arduino 1.0 Wire library (methods are different from
0.x)
//     2011-08-03 - added optional timeout parameter to read* methods to easily change
from default
//     2011-08-02 - added support for 16-bit registers
//                - fixed incorrect Doxygen comments on some methods
//                - added timeout value for read operations (thanks mem @ Arduino forums)
//     2011-07-30 - changed read/write function structures to return success or byte
counts
//                - made all methods static for multi-device memory savings
//     2011-07-28 - initial release

/* ============================================
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.
===============================================
*/

#include "I2Cdev.h"

#if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE

    #ifdef I2CDEV_IMPLEMENTATION_WARNINGS
        #if ARDUINO < 100
            #warning Using outdated Arduino IDE with Wire library is functionally limiting.
            #warning Arduino IDE v1.0.1+ with I2Cdev Fastwire implementation is
```

```
recommended.
            #warning This I2Cdev implementation does not support:
            #warning - Repeated starts conditions
            #warning - Timeout detection (some Wire requests block forever)
        #elif ARDUINO == 100
            #warning Using outdated Arduino IDE with Wire library is functionally limiting.
            #warning Arduino IDE v1.0.1+ with I2Cdev Fastwire implementation is
recommended.
            #warning This I2Cdev implementation does not support:
            #warning - Repeated starts conditions
            #warning - Timeout detection (some Wire requests block forever)
        #elif ARDUINO > 100
            /*
            #warning Using current Arduino IDE with Wire library is functionally limiting.
            #warning Arduino IDE v1.0.1+ with I2CDEV_BUILTIN_FASTWIRE
implementation is recommended.
            #warning This I2Cdev implementation does not support:
            #warning - Timeout detection (some Wire requests block forever)
            */
        #endif
    #endif

#elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE

    #error The I2CDEV_BUILTIN_FASTWIRE implementation is known to be broken right
now. Patience, Iago!

#elif I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_NBWIRE

    #ifdef I2CDEV_IMPLEMENTATION_WARNINGS
        #warning Using I2CDEV_BUILTIN_NBWIRE implementation may adversely affect
interrupt detection.
        #warning This I2Cdev implementation does not support:
        #warning - Repeated starts conditions
    #endif

    // NBWire implementation based heavily on code by Gene Knight
<Gene@Telobot.com>
    // Originally posted on the Arduino forum at
http://arduino.cc/forum/index.php/topic,70705.0.html
    // Originally offered to the i2cdevlib project at
http://arduino.cc/forum/index.php/topic,68210.30.html
    TwoWire Wire;

#endif

/** Default constructor.
 */
I2Cdev::I2Cdev() {
}
```

```
/** Read a single bit from an 8-bit device register.
 * @param devAddr I2C slave device address
 * @param regAddr Register regAddr to read from
 * @param bitNum Bit position to read (0-7)
 * @param data Container for single bit value
 * @param timeout Optional read timeout in milliseconds (0 to disable, leave off to use
default class value in I2Cdev::readTimeout)
 * @return Status of read operation (true = success)
 */
int8_t I2Cdev::readBit(uint8_t devAddr, uint8_t regAddr, uint8_t bitNum, uint8_t *data,
uint16_t timeout) {
    uint8_t b;
    uint8_t count = readByte(devAddr, regAddr, &b, timeout);
    *data = b & (1 << bitNum);
    return count;
}

/** Read a single bit from a 16-bit device register.
 * @param devAddr I2C slave device address
 * @param regAddr Register regAddr to read from
 * @param bitNum Bit position to read (0-15)
 * @param data Container for single bit value
 * @param timeout Optional read timeout in milliseconds (0 to disable, leave off to use
default class value in I2Cdev::readTimeout)
 * @return Status of read operation (true = success)
 */
int8_t I2Cdev::readBitW(uint8_t devAddr, uint8_t regAddr, uint8_t bitNum, uint16_t
*data, uint16_t timeout) {
    uint16_t b;
    uint8_t count = readWord(devAddr, regAddr, &b, timeout);
    *data = b & (1 << bitNum);
    return count;
}

/** Read multiple bits from an 8-bit device register.
 * @param devAddr I2C slave device address
 * @param regAddr Register regAddr to read from
 * @param bitStart First bit position to read (0-7)
 * @param length Number of bits to read (not more than 8)
 * @param data Container for right-aligned value (i.e. '101' read from any bitStart
position will equal 0x05)
 * @param timeout Optional read timeout in milliseconds (0 to disable, leave off to use
default class value in I2Cdev::readTimeout)
 * @return Status of read operation (true = success)
 */
int8_t I2Cdev::readBits(uint8_t devAddr, uint8_t regAddr, uint8_t bitStart, uint8_t length,
uint8_t *data, uint16_t timeout) {
    // 01101001 read byte
    // 76543210 bit numbers
    //    xxx   args: bitStart=4, length=3
    //    010   masked
```

```
    //   -> 010 shifted
    uint8_t count, b;
    if ((count = readByte(devAddr, regAddr, &b, timeout)) != 0) {
        uint8_t mask = ((1 << length) - 1) << (bitStart - length + 1);
        b &= mask;
        b >>= (bitStart - length + 1);
        *data = b;
    }
    return count;
}

/** Read multiple bits from a 16-bit device register.
 * @param devAddr I2C slave device address
 * @param regAddr Register regAddr to read from
 * @param bitStart First bit position to read (0-15)
 * @param length Number of bits to read (not more than 16)
 * @param data Container for right-aligned value (i.e. '101' read from any bitStart
position will equal 0x05)
 * @param timeout Optional read timeout in milliseconds (0 to disable, leave off to use
default class value in I2Cdev::readTimeout)
 * @return Status of read operation (1 = success, 0 = failure, -1 = timeout)
 */
int8_t I2Cdev::readBitsW(uint8_t devAddr, uint8_t regAddr, uint8_t bitStart, uint8_t
length, uint16_t *data, uint16_t timeout) {
    // 1101011001101001 read byte
    // fedcba9876543210 bit numbers
    //    xxx           args: bitStart=12, length=3
    //    010           masked
    //           -> 010 shifted
    uint8_t count;
    uint16_t w;
    if ((count = readWord(devAddr, regAddr, &w, timeout)) != 0) {
        uint16_t mask = ((1 << length) - 1) << (bitStart - length + 1);
        w &= mask;
        w >>= (bitStart - length + 1);
        *data = w;
    }
    return count;
}

/** Read single byte from an 8-bit device register.
 * @param devAddr I2C slave device address
 * @param regAddr Register regAddr to read from
 * @param data Container for byte value read from device
 * @param timeout Optional read timeout in milliseconds (0 to disable, leave off to use
default class value in I2Cdev::readTimeout)
 * @return Status of read operation (true = success)
 */
int8_t I2Cdev::readByte(uint8_t devAddr, uint8_t regAddr, uint8_t *data, uint16_t
timeout) {
    return readBytes(devAddr, regAddr, 1, data, timeout);
```

```
}

/** Read single word from a 16-bit device register.
 * @param devAddr I2C slave device address
 * @param regAddr Register regAddr to read from
 * @param data Container for word value read from device
 * @param timeout Optional read timeout in milliseconds (0 to disable, leave off to use
default class value in I2Cdev::readTimeout)
 * @return Status of read operation (true = success)
 */
int8_t I2Cdev::readWord(uint8_t devAddr, uint8_t regAddr, uint16_t *data, uint16_t
timeout) {
    return readWords(devAddr, regAddr, 1, data, timeout);
}

/** Read multiple bytes from an 8-bit device register.
 * @param devAddr I2C slave device address
 * @param regAddr First register regAddr to read from
 * @param length Number of bytes to read
 * @param data Buffer to store read data in
 * @param timeout Optional read timeout in milliseconds (0 to disable, leave off to use
default class value in I2Cdev::readTimeout)
 * @return Number of bytes read (-1 indicates failure)
 */
int8_t I2Cdev::readBytes(uint8_t devAddr, uint8_t regAddr, uint8_t length, uint8_t *data,
uint16_t timeout) {
    #ifdef I2CDEV_SERIAL_DEBUG
        Serial.print("I2C (0x");
        Serial.print(devAddr, HEX);
        Serial.print(") reading ");
        Serial.print(length, DEC);
        Serial.print(" bytes from 0x");
        Serial.print(regAddr, HEX);
        Serial.print("...");
    #endif

    int8_t count = 0;
    uint32_t t1 = millis();

    #if (I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE)

        #if (ARDUINO < 100)
            // Arduino v00xx (before v1.0), Wire library

            // I2C/TWI subsystem uses internal buffer that breaks with large data requests
            // so if user requests more than BUFFER_LENGTH bytes, we have to do it in
            // smaller chunks instead of all at once
            for (uint8_t k = 0; k < length; k += min(length, BUFFER_LENGTH)) {
                Wire.beginTransmission(devAddr);
                Wire.send(regAddr);
                Wire.endTransmission();
```

```
            Wire.beginTransmission(devAddr);
            Wire.requestFrom(devAddr, (uint8_t)min(length - k, BUFFER_LENGTH));

            for (; Wire.available() && (timeout == 0 || millis() - t1 < timeout); count++) {
                data[count] = Wire.receive();
                #ifdef I2CDEV_SERIAL_DEBUG
                    Serial.print(data[count], HEX);
                    if (count + 1 < length) Serial.print(" ");
                #endif
            }

            Wire.endTransmission();
        }
    #elif (ARDUINO == 100)
        // Arduino v1.0.0, Wire library
        // Adds standardized write() and read() stream methods instead of send() and
receive()

        // I2C/TWI subsystem uses internal buffer that breaks with large data requests
        // so if user requests more than BUFFER_LENGTH bytes, we have to do it in
        // smaller chunks instead of all at once
        for (uint8_t k = 0; k < length; k += min(length, BUFFER_LENGTH)) {
            Wire.beginTransmission(devAddr);
            Wire.write(regAddr);
            Wire.endTransmission();
            Wire.beginTransmission(devAddr);
            Wire.requestFrom(devAddr, (uint8_t)min(length - k, BUFFER_LENGTH));

            for (; Wire.available() && (timeout == 0 || millis() - t1 < timeout); count++) {
                data[count] = Wire.read();
                #ifdef I2CDEV_SERIAL_DEBUG
                    Serial.print(data[count], HEX);
                    if (count + 1 < length) Serial.print(" ");
                #endif
            }

            Wire.endTransmission();
        }
    #elif (ARDUINO > 100)
        // Arduino v1.0.1+, Wire library
        // Adds official support for repeated start condition, yay!

        // I2C/TWI subsystem uses internal buffer that breaks with large data requests
        // so if user requests more than BUFFER_LENGTH bytes, we have to do it in
        // smaller chunks instead of all at once
        for (uint8_t k = 0; k < length; k += min(length, BUFFER_LENGTH)) {
            Wire.beginTransmission(devAddr);
            Wire.write(regAddr);
            Wire.endTransmission();
            Wire.beginTransmission(devAddr);
            Wire.requestFrom(devAddr, (uint8_t)min(length - k, BUFFER_LENGTH));
```

```
            for (; Wire.available() && (timeout == 0 || millis() - t1 < timeout); count++) {
                data[count] = Wire.read();
                #ifdef I2CDEV_SERIAL_DEBUG
                    Serial.print(data[count], HEX);
                    if (count + 1 < length) Serial.print(" ");
                #endif
            }

            Wire.endTransmission();
        }
    #endif

#elif (I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE)
    // Fastwire library (STILL UNDER DEVELOPMENT, NON-FUNCTIONAL!)

    // no loop required for fastwire
    uint8_t status = Fastwire::readBuf(devAddr, regAddr, data, length);
    if (status == 0) {
        count = length; // success
    } else {
        count = -1; // error
    }

#endif

// check for timeout
if (timeout > 0 && millis() - t1 >= timeout && count < length) count = -1; // timeout

#ifdef I2CDEV_SERIAL_DEBUG
    Serial.print(". Done (");
    Serial.print(count, DEC);
    Serial.println(" read).");
#endif

return count;
}

/** Read multiple words from a 16-bit device register.
 * @param devAddr I2C slave device address
 * @param regAddr First register regAddr to read from
 * @param length Number of words to read
 * @param data Buffer to store read data in
 * @param timeout Optional read timeout in milliseconds (0 to disable, leave off to use
default class value in I2Cdev::readTimeout)
 * @return Number of words read (0 indicates failure)
 */
int8_t I2Cdev::readWords(uint8_t devAddr, uint8_t regAddr, uint8_t length, uint16_t
*data, uint16_t timeout) {
    #ifdef I2CDEV_SERIAL_DEBUG
        Serial.print("I2C (0x");
```

```
        Serial.print(devAddr, HEX);
        Serial.print(") reading ");
        Serial.print(length, DEC);
        Serial.print(" words from 0x");
        Serial.print(regAddr, HEX);
        Serial.print("...");
    #endif

    int8_t count = 0;
    uint32_t t1 = millis();

    #if (I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE)

        #if (ARDUINO < 100)
            // Arduino v00xx (before v1.0), Wire library

            // I2C/TWI subsystem uses internal buffer that breaks with large data requests
            // so if user requests more than BUFFER_LENGTH bytes, we have to do it in
            // smaller chunks instead of all at once
            for (uint8_t k = 0; k < length * 2; k += min(length * 2, BUFFER_LENGTH)) {
                Wire.beginTransmission(devAddr);
                Wire.send(regAddr);
                Wire.endTransmission();
                Wire.beginTransmission(devAddr);
                Wire.requestFrom(devAddr, (uint8_t)(length * 2)); // length=words, this wants
bytes

                bool msb = true; // starts with MSB, then LSB
                for (; Wire.available() && count < length && (timeout == 0 || millis() - t1 <
timeout);) {
                    if (msb) {
                        // first byte is bits 15-8 (MSb=15)
                        data[count] = Wire.receive() << 8;
                    } else {
                        // second byte is bits 7-0 (LSb=0)
                        data[count] |= Wire.receive();
                        #ifdef I2CDEV_SERIAL_DEBUG
                            Serial.print(data[count], HEX);
                            if (count + 1 < length) Serial.print(" ");
                        #endif
                        count++;
                    }
                    msb = !msb;
                }

                Wire.endTransmission();
            }
        #elif (ARDUINO == 100)
            // Arduino v1.0.0, Wire library
            // Adds standardized write() and read() stream methods instead of send() and
receive()
```

```
            // I2C/TWI subsystem uses internal buffer that breaks with large data requests
            // so if user requests more than BUFFER_LENGTH bytes, we have to do it in
            // smaller chunks instead of all at once
            for (uint8_t k = 0; k < length * 2; k += min(length * 2, BUFFER_LENGTH)) {
                Wire.beginTransmission(devAddr);
                Wire.write(regAddr);
                Wire.endTransmission();
                Wire.beginTransmission(devAddr);
                Wire.requestFrom(devAddr, (uint8_t)(length * 2)); // length=words, this wants
bytes

                bool msb = true; // starts with MSB, then LSB
                for (; Wire.available() && count < length && (timeout == 0 || millis() - t1 <
timeout);) {
                    if (msb) {
                        // first byte is bits 15-8 (MSb=15)
                        data[count] = Wire.read() << 8;
                    } else {
                        // second byte is bits 7-0 (LSb=0)
                        data[count] |= Wire.read();
                        #ifdef I2CDEV_SERIAL_DEBUG
                            Serial.print(data[count], HEX);
                            if (count + 1 < length) Serial.print(" ");
                        #endif
                        count++;
                    }
                    msb = !msb;
                }

                Wire.endTransmission();
            }
        #elif (ARDUINO > 100)
            // Arduino v1.0.1+, Wire library
            // Adds official support for repeated start condition, yay!

            // I2C/TWI subsystem uses internal buffer that breaks with large data requests
            // so if user requests more than BUFFER_LENGTH bytes, we have to do it in
            // smaller chunks instead of all at once
            for (uint8_t k = 0; k < length * 2; k += min(length * 2, BUFFER_LENGTH)) {
                Wire.beginTransmission(devAddr);
                Wire.write(regAddr);
                Wire.endTransmission();
                Wire.beginTransmission(devAddr);
                Wire.requestFrom(devAddr, (uint8_t)(length * 2)); // length=words, this wants
bytes

                bool msb = true; // starts with MSB, then LSB
                for (; Wire.available() && count < length && (timeout == 0 || millis() - t1 <
timeout);) {
                    if (msb) {
```

```
                    // first byte is bits 15-8 (MSb=15)
                    data[count] = Wire.read() << 8;
                } else {
                    // second byte is bits 7-0 (LSb=0)
                    data[count] |= Wire.read();
                    #ifdef I2CDEV_SERIAL_DEBUG
                        Serial.print(data[count], HEX);
                        if (count + 1 < length) Serial.print(" ");
                    #endif
                    count++;
                }
                msb = !msb;
            }

            Wire.endTransmission();
        }
    #endif

#elif (I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE)
    // Fastwire library (STILL UNDER DEVELOPMENT, NON-FUNCTIONAL!)

    // no loop required for fastwire
    uint16_t intermediate[(uint8_t)length];
    uint8_t status = Fastwire::readBuf(devAddr, regAddr, (uint8_t *)intermediate,
(uint8_t)(length * 2));
    if (status == 0) {
        count = length; // success
        for (uint8_t i = 0; i < length; i++) {
            data[i] = (intermediate[2*i] << 8) | intermediate[2*i + 1];
        }
    } else {
        count = -1; // error
    }

#endif

if (timeout > 0 && millis() - t1 >= timeout && count < length) count = -1; // timeout

#ifdef I2CDEV_SERIAL_DEBUG
    Serial.print(". Done (");
    Serial.print(count, DEC);
    Serial.println(" read).");
#endif

return count;
}

/** write a single bit in an 8-bit device register.
 * @param devAddr I2C slave device address
 * @param regAddr Register regAddr to write to
 * @param bitNum Bit position to write (0-7)
```

```
 * @param value New bit value to write
 * @return Status of operation (true = success)
 */
bool I2Cdev::writeBit(uint8_t devAddr, uint8_t regAddr, uint8_t bitNum, uint8_t data) {
    uint8_t b;
    readByte(devAddr, regAddr, &b);
    b = (data != 0) ? (b | (1 << bitNum)) : (b & ~(1 << bitNum));
    return writeByte(devAddr, regAddr, b);
}

/** write a single bit in a 16-bit device register.
 * @param devAddr I2C slave device address
 * @param regAddr Register regAddr to write to
 * @param bitNum Bit position to write (0-15)
 * @param value New bit value to write
 * @return Status of operation (true = success)
 */
bool I2Cdev::writeBitW(uint8_t devAddr, uint8_t regAddr, uint8_t bitNum, uint16_t data) {
    uint16_t w;
    readWord(devAddr, regAddr, &w);
    w = (data != 0) ? (w | (1 << bitNum)) : (w & ~(1 << bitNum));
    return writeWord(devAddr, regAddr, w);
}

/** Write multiple bits in an 8-bit device register.
 * @param devAddr I2C slave device address
 * @param regAddr Register regAddr to write to
 * @param bitStart First bit position to write (0-7)
 * @param length Number of bits to write (not more than 8)
 * @param data Right-aligned value to write
 * @return Status of operation (true = success)
 */
bool I2Cdev::writeBits(uint8_t devAddr, uint8_t regAddr, uint8_t bitStart, uint8_t length,
uint8_t data) {
    //      010 value to write
    // 76543210 bit numbers
    //    xxx   args: bitStart=4, length=3
    // 00011100 mask byte
    // 10101111 original value (sample)
    // 10100011 original & ~mask
    // 10101011 masked | value
    uint8_t b;
    if (readByte(devAddr, regAddr, &b) != 0) {
        uint8_t mask = ((1 << length) - 1) << (bitStart - length + 1);
        data <<= (bitStart - length + 1); // shift data into correct position
        data &= mask; // zero all non-important bits in data
        b &= ~(mask); // zero all important bits in existing byte
        b |= data; // combine data with existing byte
        return writeByte(devAddr, regAddr, b);
    } else {
        return false;
```

```
    }
}

/** Write multiple bits in a 16-bit device register.
 * @param devAddr I2C slave device address
 * @param regAddr Register regAddr to write to
 * @param bitStart First bit position to write (0-15)
 * @param length Number of bits to write (not more than 16)
 * @param data Right-aligned value to write
 * @return Status of operation (true = success)
 */
bool I2Cdev::writeBitsW(uint8_t devAddr, uint8_t regAddr, uint8_t bitStart, uint8_t length,
uint16_t data) {
    //              010 value to write
    // fedcba9876543210 bit numbers
    //    xxx           args: bitStart=12, length=3
    // 0001110000000000 mask byte
    // 1010111110010110 original value (sample)
    // 1010001110010110 original & ~mask
    // 1010101110010110 masked | value
    uint16_t w;
    if (readWord(devAddr, regAddr, &w) != 0) {
        uint8_t mask = ((1 << length) - 1) << (bitStart - length + 1);
        data <<= (bitStart - length + 1); // shift data into correct position
        data &= mask; // zero all non-important bits in data
        w &= ~(mask); // zero all important bits in existing word
        w |= data; // combine data with existing word
        return writeWord(devAddr, regAddr, w);
    } else {
        return false;
    }
}

/** Write single byte to an 8-bit device register.
 * @param devAddr I2C slave device address
 * @param regAddr Register address to write to
 * @param data New byte value to write
 * @return Status of operation (true = success)
 */
bool I2Cdev::writeByte(uint8_t devAddr, uint8_t regAddr, uint8_t data) {
    return writeBytes(devAddr, regAddr, 1, &data);
}

/** Write single word to a 16-bit device register.
 * @param devAddr I2C slave device address
 * @param regAddr Register address to write to
 * @param data New word value to write
 * @return Status of operation (true = success)
 */
bool I2Cdev::writeWord(uint8_t devAddr, uint8_t regAddr, uint16_t data) {
    return writeWords(devAddr, regAddr, 1, &data);
```

```
}

/** Write multiple bytes to an 8-bit device register.
 * @param devAddr I2C slave device address
 * @param regAddr First register address to write to
 * @param length Number of bytes to write
 * @param data Buffer to copy new data from
 * @return Status of operation (true = success)
 */
bool I2Cdev::writeBytes(uint8_t devAddr, uint8_t regAddr, uint8_t length, uint8_t* data) {
    #ifdef I2CDEV_SERIAL_DEBUG
        Serial.print("I2C (0x");
        Serial.print(devAddr, HEX);
        Serial.print(") writing ");
        Serial.print(length, DEC);
        Serial.print(" bytes to 0x");
        Serial.print(regAddr, HEX);
        Serial.print("...");
    #endif
    uint8_t status = 0;
    #if ((I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE && ARDUINO <
100) || I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_NBWIRE)
        Wire.beginTransmission(devAddr);
        Wire.send((uint8_t) regAddr); // send address
    #elif (I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE && ARDUINO >=
100)
        Wire.beginTransmission(devAddr);
        Wire.write((uint8_t) regAddr); // send address
    #endif
    for (uint8_t i = 0; i < length; i++) {
        #if ((I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE && ARDUINO <
100) || I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_NBWIRE)
            Wire.send((uint8_t) data[i]);
        #elif (I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE && ARDUINO
>= 100)
            Wire.write((uint8_t) data[i]);
        #elif (I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE)
            status = Fastwire::write(devAddr, regAddr, data[i]);
            Serial.println(status);
        #endif
        #ifdef I2CDEV_SERIAL_DEBUG
            Serial.print(data[i], HEX);
            if (i + 1 < length) Serial.print(" ");
        #endif
    }
    #if ((I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE && ARDUINO <
100) || I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_NBWIRE)
        Wire.endTransmission();
    #elif (I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE && ARDUINO >=
100)
        status = Wire.endTransmission();
```

```
    #endif
    #ifdef I2CDEV_SERIAL_DEBUG
        Serial.println(". Done.");
    #endif
    return status == 0;
}

/** Write multiple words to a 16-bit device register.
 * @param devAddr I2C slave device address
 * @param regAddr First register address to write to
 * @param length Number of words to write
 * @param data Buffer to copy new data from
 * @return Status of operation (true = success)
 */
bool I2Cdev::writeWords(uint8_t devAddr, uint8_t regAddr, uint8_t length, uint16_t* data)
{
    #ifdef I2CDEV_SERIAL_DEBUG
        Serial.print("I2C (0x");
        Serial.print(devAddr, HEX);
        Serial.print(") writing ");
        Serial.print(length, DEC);
        Serial.print(" words to 0x");
        Serial.print(regAddr, HEX);
        Serial.print("...");
    #endif
    uint8_t status = 0;
    #if ((I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE && ARDUINO <
100) || I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_NBWIRE)
        Wire.beginTransmission(devAddr);
        Wire.send(regAddr); // send address
    #elif (I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE && ARDUINO >=
100)
        Wire.beginTransmission(devAddr);
        Wire.write(regAddr); // send address
    #endif
    for (uint8_t i = 0; i < length * 2; i++) {
        #if ((I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE && ARDUINO <
100) || I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_NBWIRE)
            Wire.send((uint8_t)(data[i++] >> 8)); // send MSB
            Wire.send((uint8_t)data[i]);          // send LSB
        #elif (I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE && ARDUINO
>= 100)
            Wire.write((uint8_t)(data[i++] >> 8)); // send MSB
            Wire.write((uint8_t)data[i]);          // send LSB
        #elif (I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE)
            status = Fastwire::write(devAddr, regAddr, (uint8_t)(data[i++] >> 8));
            status = Fastwire::write(devAddr, regAddr + 1, (uint8_t)data[i]);
        #endif
        #ifdef I2CDEV_SERIAL_DEBUG
            Serial.print(data[i], HEX);
            if (i + 1 < length) Serial.print(" ");
```

```
        #endif
    }
    #if ((I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE && ARDUINO <
100) || I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_NBWIRE)
        Wire.endTransmission();
    #elif (I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE && ARDUINO >=
100)
        status = Wire.endTransmission();
    #endif
    #ifdef I2CDEV_SERIAL_DEBUG
        Serial.println(". Done.");
    #endif
    return status == 0;
}

/** Default timeout value for read operations.
 * Set this to 0 to disable timeout detection.
 */
uint16_t I2Cdev::readTimeout = I2CDEV_DEFAULT_READ_TIMEOUT;

#if I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
    /*
    FastWire 0.2
    This is a library to help faster programs to read I2C devices.
    Copyright(C) 2011 Francesco Ferrara
    occhiobello at gmail dot com
    */

    boolean Fastwire::waitInt() {
        int l = 250;
        while (!(TWCR & (1 << TWINT)) && l-- > 0);
        return l > 0;
    }

    void Fastwire::setup(int khz, boolean pullup) {
        TWCR = 0;
        #if defined(__AVR_ATmega168__) || defined(__AVR_ATmega8__) ||
defined(__AVR_ATmega328P__)
            // activate internal pull-ups for twi (PORTC bits 4 & 5)
            // as per note from atmega8 manual pg167
            if (pullup) PORTC |= ((1 << 4) | (1 << 5));
            else        PORTC &= ~((1 << 4) | (1 << 5));
        #elif defined(__AVR_ATmega644P__) || defined(__AVR_ATmega644__)
            // activate internal pull-ups for twi (PORTC bits 0 & 1)
            if (pullup) PORTC |= ((1 << 0) | (1 << 1));
            else        PORTC &= ~((1 << 0) | (1 << 1));
        #else
            // activate internal pull-ups for twi (PORTD bits 0 & 1)
            // as per note from atmega128 manual pg204
            if (pullup) PORTD |= ((1 << 0) | (1 << 1));
            else        PORTD &= ~((1 << 0) | (1 << 1));
```

```
    #endif

    TWSR = 0; // no prescaler => prescaler = 1
    TWBR = ((16000L / khz) - 16) / 2; // change the I2C clock rate
    TWCR = 1 << TWEN; // enable twi module, no interrupt
}

byte Fastwire::write(byte device, byte address, byte value) {
    byte twst, retry;

    retry = 2;
    do {
        TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO) | (1 << TWSTA);
        if (!waitInt()) return 1;
        twst = TWSR & 0xF8;
        if (twst != TW_START && twst != TW_REP_START) return 2;

        TWDR = device & 0xFE; // send device address without read bit (1)
        TWCR = (1 << TWINT) | (1 << TWEN);
        if (!waitInt()) return 3;
        twst = TWSR & 0xF8;
    } while (twst == TW_MT_SLA_NACK && retry-- > 0);
    if (twst != TW_MT_SLA_ACK) return 4;

    TWDR = address; // send data to the previously addressed device
    TWCR = (1 << TWINT) | (1 << TWEN);
    if (!waitInt()) return 5;
    twst = TWSR & 0xF8;
    if (twst != TW_MT_DATA_ACK) return 6;

    TWDR = value; // send data to the previously addressed device
    TWCR = (1 << TWINT) | (1 << TWEN);
    if (!waitInt()) return 7;
    twst = TWSR & 0xF8;
    if (twst != TW_MT_DATA_ACK) return 8;

    return 0;
}

byte Fastwire::readBuf(byte device, byte address, byte *data, byte num) {
    byte twst, retry;

    retry = 2;
    do {
        TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO) | (1 << TWSTA);
        if (!waitInt()) return 16;
        twst = TWSR & 0xF8;
        if (twst != TW_START && twst != TW_REP_START) return 17;

        TWDR = device & 0xfe; // send device address to write
        TWCR = (1 << TWINT) | (1 << TWEN);
```

```
        if (!waitInt()) return 18;
        twst = TWSR & 0xF8;
      } while (twst == TW_MT_SLA_NACK && retry-- > 0);
      if (twst != TW_MT_SLA_ACK) return 19;

      TWDR = address; // send data to the previously addressed device
      TWCR = (1 << TWINT) | (1 << TWEN);
      if (!waitInt()) return 20;
      twst = TWSR & 0xF8;
      if (twst != TW_MT_DATA_ACK) return 21;

      /***/

      retry = 2;
      do {
        TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWSTO) | (1 << TWSTA);
        if (!waitInt()) return 22;
        twst = TWSR & 0xF8;
        if (twst != TW_START && twst != TW_REP_START) return 23;

        TWDR = device | 0x01; // send device address with the read bit (1)
        TWCR = (1 << TWINT) | (1 << TWEN);
        if (!waitInt()) return 24;
        twst = TWSR & 0xF8;
      } while (twst == TW_MR_SLA_NACK && retry-- > 0);
      if (twst != TW_MR_SLA_ACK) return 25;

      for(uint8_t i = 0; i < num; i++) {
        if (i == num - 1)
          TWCR = (1 << TWINT) | (1 << TWEN);
        else
          TWCR = (1 << TWINT) | (1 << TWEN) | (1 << TWEA);
        if (!waitInt()) return 26;
        twst = TWSR & 0xF8;
        if (twst != TW_MR_DATA_ACK && twst != TW_MR_DATA_NACK) return twst;
        data[i] = TWDR;
      }

      return 0;
    }
#endif

#if I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_NBWIRE
    // NBWire implementation based heavily on code by Gene Knight
<Gene@Telobot.com>
    // Originally posted on the Arduino forum at
http://arduino.cc/forum/index.php/topic,70705.0.html
    // Originally offered to the i2cdevlib project at
http://arduino.cc/forum/index.php/topic,68210.30.html

    /*
```

```
    call this version 1.0

    Offhand, the only funky part that I can think of is in nbrequestFrom, where the buffer
    length and index are set *before* the data is actually read. The problem is that these
    are variables local to the TwoWire object, and by the time we actually have read the
    data, and know what the length actually is, we have no simple access to the object's
    variables. The actual bytes read *is* given to the callback function, though.

    The ISR code for a slave receiver is commented out. I don't have that setup, and can't
    verify it at this time. Save it for 2.0!

    The handling of the read and write processes here is much like in the demo sketch
code:
    the process is broken down into sequential functions, where each registers the next
as a
    callback, essentially.

    For example, for the Read process, twi_read00 just returns if TWI is not yet in a
    ready state. When there's another interrupt, and the interface *is* ready, then it
    sets up the read, starts it, and registers twi_read01 as the function to call after
    the *next* interrupt. twi_read01, then, just returns if the interface is still in a
    "reading" state. When the reading is done, it copies the information to the buffer,
    cleans up, and calls the user-requested callback function with the actual number of
    bytes read.

    The writing is similar.

    Questions, comments and problems can go to Gene@Telobot.com.

    Thumbs Up!
    Gene Knight

    */

    uint8_t TwoWire::rxBuffer[NBWIRE_BUFFER_LENGTH];
    uint8_t TwoWire::rxBufferIndex = 0;
    uint8_t TwoWire::rxBufferLength = 0;

    uint8_t TwoWire::txAddress = 0;
    uint8_t TwoWire::txBuffer[NBWIRE_BUFFER_LENGTH];
    uint8_t TwoWire::txBufferIndex = 0;
    uint8_t TwoWire::txBufferLength = 0;

    //uint8_t TwoWire::transmitting = 0;
    void (*TwoWire::user_onRequest)(void);
    void (*TwoWire::user_onReceive)(int);

    static volatile uint8_t twi_transmitting;
    static volatile uint8_t twi_state;
    static uint8_t twi_slarw;
    static volatile uint8_t twi_error;
```

```
static uint8_t twi_masterBuffer[TWI_BUFFER_LENGTH];
static volatile uint8_t twi_masterBufferIndex;
static uint8_t twi_masterBufferLength;
static uint8_t twi_rxBuffer[TWI_BUFFER_LENGTH];
static volatile uint8_t twi_rxBufferIndex;
//static volatile uint8_t twi_Interrupt_Continue_Command;
static volatile uint8_t twi_Return_Value;
static volatile uint8_t twi_Done;
void (*twi_cbendTransmissionDone)(int);
void (*twi_cbreadFromDone)(int);

void twi_init() {
  // initialize state
  twi_state = TWI_READY;

  // activate internal pull-ups for twi
  // as per note from atmega8 manual pg167
  sbi(PORTC, 4);
  sbi(PORTC, 5);

  // initialize twi prescaler and bit rate
  cbi(TWSR, TWPS0); // TWI Status Register - Prescaler bits
  cbi(TWSR, TWPS1);

  /* twi bit rate formula from atmega128 manual pg 204
  SCL Frequency = CPU Clock Frequency / (16 + (2 * TWBR))
  note: TWBR should be 10 or higher for master mode
  It is 72 for a 16mhz Wiring board with 100kHz TWI */

  TWBR = ((CPU_FREQ / TWI_FREQ) - 16) / 2; // bitrate register
  // enable twi module, acks, and twi interrupt

  TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWEA);

  /* TWEN - TWI Enable Bit
  TWIE - TWI Interrupt Enable
  TWEA - TWI Enable Acknowledge Bit
  TWINT - TWI Interrupt Flag
  TWSTA - TWI Start Condition
  */
}

typedef struct {
  uint8_t address;
  uint8_t* data;
  uint8_t length;
  uint8_t wait;
  uint8_t i;
} twi_Write_Vars;

twi_Write_Vars *ptwv = 0;
```

```
static void (*fNextInterruptFunction)(void) = 0;

void twi_Finish(byte bRetVal) {
    if (ptwv) {
        free(ptwv);
        ptwv = 0;
    }
    twi_Done = 0xFF;
    twi_Return_Value = bRetVal;
    fNextInterruptFunction = 0;
}

uint8_t twii_WaitForDone(uint16_t timeout) {
    uint32_t endMillis = millis() + timeout;
    while (!twi_Done && (timeout == 0 || millis() < endMillis)) continue;
    return twi_Return_Value;
}

void twii_SetState(uint8_t ucState) {
    twi_state = ucState;
}

void twii_SetError(uint8_t ucError) {
    twi_error = ucError ;
}

void twii_InitBuffer(uint8_t ucPos, uint8_t ucLength) {
    twi_masterBufferIndex = 0;
    twi_masterBufferLength = ucLength;
}

void twii_CopyToBuf(uint8_t* pData, uint8_t ucLength) {
    uint8_t i;
    for (i = 0; i < ucLength; ++i) {
        twi_masterBuffer[i] = pData[i];
    }
}

void twii_CopyFromBuf(uint8_t *pData, uint8_t ucLength) {
    uint8_t i;
    for (i = 0; i < ucLength; ++i) {
        pData[i] = twi_masterBuffer[i];
    }
}

void twii_SetSlaRW(uint8_t ucSlaRW) {
    twi_slarw = ucSlaRW;
}

void twii_SetStart() {
    TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWEA) | _BV(TWINT) | _BV(TWSTA);
```

```
  }

  void twi_write01() {
    if (TWI_MTX == twi_state) return; // blocking test
    twi_transmitting = 0 ;
    if (twi_error == 0xFF)
      twi_Finish (0);    // success
    else if (twi_error == TW_MT_SLA_NACK)
      twi_Finish (2);    // error: address send, nack received
    else if (twi_error == TW_MT_DATA_NACK)
      twi_Finish (3);    // error: data send, nack received
    else
      twi_Finish (4);    // other twi error
    if (twi_cbendTransmissionDone) return
twi_cbendTransmissionDone(twi_Return_Value);
    return;
  }


  void twi_write00() {
    if (TWI_READY != twi_state) return; // blocking test
    if (TWI_BUFFER_LENGTH < ptwv -> length) {
      twi_Finish(1); // end write with error 1
      return;
    }
    twi_Done = 0x00; // show as working
    twii_SetState(TWI_MTX); // to transmitting
    twii_SetError(0xFF); // to No Error
    twii_InitBuffer(0, ptwv -> length); // pointer and length
    twii_CopyToBuf(ptwv -> data, ptwv -> length); // get the data
    twii_SetSlaRW((ptwv -> address << 1) | TW_WRITE); // write command
    twii_SetStart(); // start the cycle
    fNextInterruptFunction = twi_write01; // next routine
    return twi_write01();
  }

  void twi_writeTo(uint8_t address, uint8_t* data, uint8_t length, uint8_t wait) {
    uint8_t i;
    ptwv = (twi_Write_Vars *)malloc(sizeof(twi_Write_Vars));
    ptwv -> address = address;
    ptwv -> data = data;
    ptwv -> length = length;
    ptwv -> wait = wait;
    fNextInterruptFunction = twi_write00;
    return twi_write00();
  }

  void twi_read01() {
    if (TWI_MRX == twi_state) return; // blocking test
    if (twi_masterBufferIndex < ptwv -> length) ptwv -> length = twi_masterBufferIndex;
    twii_CopyFromBuf(ptwv -> data, ptwv -> length);
```

```
    twi_Finish(ptwv -> length);
    if (twi_cbreadFromDone) return twi_cbreadFromDone(twi_Return_Value);
    return;
}

void twi_read00() {
    if (TWI_READY != twi_state) return; // blocking test
    if (TWI_BUFFER_LENGTH < ptwv -> length) twi_Finish(0); // error return
    twi_Done = 0x00; // show as working
    twii_SetState(TWI_MRX); // reading
    twii_SetError(0xFF); // reset error
    twii_InitBuffer(0, ptwv -> length - 1); // init to one less than length
    twii_SetSlaRW((ptwv -> address << 1) | TW_READ); // read command
    twii_SetStart(); // start cycle
    fNextInterruptFunction = twi_read01;
    return twi_read01();
}

void twi_readFrom(uint8_t address, uint8_t* data, uint8_t length) {
    uint8_t i;

    ptwv = (twi_Write_Vars *)malloc(sizeof(twi_Write_Vars));
    ptwv -> address = address;
    ptwv -> data = data;
    ptwv -> length = length;
    fNextInterruptFunction = twi_read00;
    return twi_read00();
}

void twi_reply(uint8_t ack) {
    // transmit master read ready signal, with or without ack
    if (ack){
        TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWINT) | _BV(TWEA);
    } else {
        TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWINT);
    }
}

void twi_stop(void) {
    // send stop condition
    TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWEA) | _BV(TWINT) | _BV(TWSTO);

    // wait for stop condition to be exectued on bus
    // TWINT is not set after a stop condition!
    while (TWCR & _BV(TWSTO)) {
        continue;
    }

    // update twi state
    twi_state = TWI_READY;
}
```

```
void twi_releaseBus(void) {
  // release bus
  TWCR = _BV(TWEN) | _BV(TWIE) | _BV(TWEA) | _BV(TWINT);

  // update twi state
  twi_state = TWI_READY;
}

SIGNAL(TWI_vect) {
  switch (TW_STATUS) {
    // All Master
    case TW_START:     // sent start condition
    case TW_REP_START: // sent repeated start condition
      // copy device address and r/w bit to output register and ack
      TWDR = twi_slarw;
      twi_reply(1);
      break;

    // Master Transmitter
    case TW_MT_SLA_ACK:  // slave receiver acked address
    case TW_MT_DATA_ACK: // slave receiver acked data
      // if there is data to send, send it, otherwise stop
      if (twi_masterBufferIndex < twi_masterBufferLength) {
        // copy data to output register and ack
        TWDR = twi_masterBuffer[twi_masterBufferIndex++];
        twi_reply(1);
      } else {
        twi_stop();
      }
      break;

    case TW_MT_SLA_NACK:  // address sent, nack received
      twi_error = TW_MT_SLA_NACK;
      twi_stop();
      break;

    case TW_MT_DATA_NACK: // data sent, nack received
      twi_error = TW_MT_DATA_NACK;
      twi_stop();
      break;

    case TW_MT_ARB_LOST: // lost bus arbitration
      twi_error = TW_MT_ARB_LOST;
      twi_releaseBus();
      break;

    // Master Receiver
    case TW_MR_DATA_ACK: // data received, ack sent
      // put byte into buffer
      twi_masterBuffer[twi_masterBufferIndex++] = TWDR;
```

```
    case TW_MR_SLA_ACK:  // address sent, ack received
      // ack if more bytes are expected, otherwise nack
      if (twi_masterBufferIndex < twi_masterBufferLength) {
        twi_reply(1);
      } else {
        twi_reply(0);
      }
      break;

    case TW_MR_DATA_NACK: // data received, nack sent
      // put final byte into buffer
      twi_masterBuffer[twi_masterBufferIndex++] = TWDR;

    case TW_MR_SLA_NACK: // address sent, nack received
      twi_stop();
      break;

// TW_MR_ARB_LOST handled by TW_MT_ARB_LOST case

// Slave Receiver (NOT IMPLEMENTED YET)
/*
    case TW_SR_SLA_ACK:   // addressed, returned ack
    case TW_SR_GCALL_ACK: // addressed generally, returned ack
    case TW_SR_ARB_LOST_SLA_ACK:   // lost arbitration, returned ack
    case TW_SR_ARB_LOST_GCALL_ACK: // lost arbitration, returned ack
      // enter slave receiver mode
      twi_state = TWI_SRX;

      // indicate that rx buffer can be overwritten and ack
      twi_rxBufferIndex = 0;
      twi_reply(1);
      break;

    case TW_SR_DATA_ACK:       // data received, returned ack
    case TW_SR_GCALL_DATA_ACK: // data received generally, returned ack
      // if there is still room in the rx buffer
      if (twi_rxBufferIndex < TWI_BUFFER_LENGTH) {
        // put byte in buffer and ack
        twi_rxBuffer[twi_rxBufferIndex++] = TWDR;
        twi_reply(1);
      } else {
        // otherwise nack
        twi_reply(0);
      }
      break;

    case TW_SR_STOP: // stop or repeated start condition received
      // put a null char after data if there's room
      if (twi_rxBufferIndex < TWI_BUFFER_LENGTH) {
        twi_rxBuffer[twi_rxBufferIndex] = 0;
```

```
    }

    // sends ack and stops interface for clock stretching
    twi_stop();

    // callback to user defined callback
    twi_onSlaveReceive(twi_rxBuffer, twi_rxBufferIndex);

    // since we submit rx buffer to "wire" library, we can reset it
    twi_rxBufferIndex = 0;

    // ack future responses and leave slave receiver state
    twi_releaseBus();
    break;

case TW_SR_DATA_NACK:       // data received, returned nack
case TW_SR_GCALL_DATA_NACK: // data received generally, returned nack
    // nack back at master
    twi_reply(0);
    break;

// Slave Transmitter
case TW_ST_SLA_ACK:          // addressed, returned ack
case TW_ST_ARB_LOST_SLA_ACK: // arbitration lost, returned ack
    // enter slave transmitter mode
    twi_state = TWI_STX;

    // ready the tx buffer index for iteration
    twi_txBufferIndex = 0;

    // set tx buffer length to be zero, to verify if user changes it
    twi_txBufferLength = 0;

    // request for txBuffer to be filled and length to be set
    // note: user must call twi_transmit(bytes, length) to do this
    twi_onSlaveTransmit();

    // if they didn't change buffer & length, initialize it
    if (0 == twi_txBufferLength) {
        twi_txBufferLength = 1;
        twi_txBuffer[0] = 0x00;
    }

    // transmit first byte from buffer, fall through

case TW_ST_DATA_ACK: // byte sent, ack returned
    // copy data to output register
    TWDR = twi_txBuffer[twi_txBufferIndex++];

    // if there is more to send, ack, otherwise nack
    if (twi_txBufferIndex < twi_txBufferLength) {
```

```
            twi_reply(1);
          } else {
            twi_reply(0);
          }
          break;

        case TW_ST_DATA_NACK: // received nack, we are done
        case TW_ST_LAST_DATA: // received ack, but we are done already!
          // ack future responses
          twi_reply(1);
          // leave slave receiver state
          twi_state = TWI_READY;
          break;
        */

        // all
        case TW_NO_INFO:   // no state information
          break;

        case TW_BUS_ERROR: // bus error, illegal stop/start
          twi_error = TW_BUS_ERROR;
          twi_stop();
          break;
    }

    if (fNextInterruptFunction) return fNextInterruptFunction();
}

TwoWire::TwoWire() { }

void TwoWire::begin(void) {
    rxBufferIndex = 0;
    rxBufferLength = 0;

    txBufferIndex = 0;
    txBufferLength = 0;

    twi_init();
}

void TwoWire::beginTransmission(uint8_t address) {
    //beginTransmission((uint8_t)address);

    // indicate that we are transmitting
    twi_transmitting = 1;

    // set address of targeted slave
    txAddress = address;

    // reset tx buffer iterator vars
    txBufferIndex = 0;
```

```
    txBufferLength = 0;
}

uint8_t TwoWire::endTransmission(uint16_t timeout) {
    // transmit buffer (blocking)
    //int8_t ret =
    twi_cbendTransmissionDone = NULL;
    twi_writeTo(txAddress, txBuffer, txBufferLength, 1);
    int8_t ret = twii_WaitForDone(timeout);

    // reset tx buffer iterator vars
    txBufferIndex = 0;
    txBufferLength = 0;

    // indicate that we are done transmitting
    // twi_transmitting = 0;
    return ret;
}

void TwoWire::nbendTransmission(void (*function)(int)) {
    twi_cbendTransmissionDone = function;
    twi_writeTo(txAddress, txBuffer, txBufferLength, 1);
    return;
}

void TwoWire::send(uint8_t data) {
    if (twi_transmitting) {
        // in master transmitter mode
        // don't bother if buffer is full
        if (txBufferLength >= NBWIRE_BUFFER_LENGTH) {
            return;
        }

        // put byte in tx buffer
        txBuffer[txBufferIndex] = data;
        ++txBufferIndex;

        // update amount in buffer
        txBufferLength = txBufferIndex;
    } else {
        // in slave send mode
        // reply to master
        //twi_transmit(&data, 1);
    }
}

uint8_t TwoWire::receive(void) {
    // default to returning null char
    // for people using with char strings
    uint8_t value = 0;
```

```
    // get each successive byte on each call
    if (rxBufferIndex < rxBufferLength) {
        value = rxBuffer[rxBufferIndex];
        ++rxBufferIndex;
    }

    return value;
}

uint8_t TwoWire::requestFrom(uint8_t address, int quantity, uint16_t timeout) {
    // clamp to buffer length
    if (quantity > NBWIRE_BUFFER_LENGTH) {
        quantity = NBWIRE_BUFFER_LENGTH;
    }

    // perform blocking read into buffer
    twi_cbreadFromDone = NULL;
    twi_readFrom(address, rxBuffer, quantity);
    uint8_t read = twii_WaitForDone(timeout);

    // set rx buffer iterator vars
    rxBufferIndex = 0;
    rxBufferLength = read;

    return read;
}

void TwoWire::nbrequestFrom(uint8_t address, int quantity, void (*function)(int)) {
    // clamp to buffer length
    if (quantity > NBWIRE_BUFFER_LENGTH) {
        quantity = NBWIRE_BUFFER_LENGTH;
    }

    // perform blocking read into buffer
    twi_cbreadFromDone = function;
    twi_readFrom(address, rxBuffer, quantity);
    //uint8_t read = twii_WaitForDone();

    // set rx buffer iterator vars
    //rxBufferIndex = 0;
    //rxBufferLength = read;

    rxBufferIndex = 0;
    rxBufferLength = quantity; // this is a hack

    return; //read;
}

uint8_t TwoWire::available(void) {
    return rxBufferLength - rxBufferIndex;
}
```

```
#endif

//end I2Cdev.cpp
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
//I2Cdev.h
// I2Cdev library collection - Main I2C device class header file
// Abstracts bit and byte I2C R/W functions into a convenient class
// 6/9/2012 by Jeff Rowberg <jeff@rowberg.net>
//
// Changelog:
//     2012-06-09 - fix major issue with reading > 32 bytes at a time with Arduino Wire
//                - add compiler warnings when using outdated or IDE or limited I2Cdev
implementation
//     2011-11-01 - fix write*Bits mask calculation (thanks sasquatch @ Arduino forums)
//     2011-10-03 - added automatic Arduino version detection for ease of use
//     2011-10-02 - added Gene Knight's NBWire TwoWire class implementation with
small modifications
//     2011-08-31 - added support for Arduino 1.0 Wire library (methods are different from
0.x)
//     2011-08-03 - added optional timeout parameter to read* methods to easily change
from default
//     2011-08-02 - added support for 16-bit registers
//                - fixed incorrect Doxygen comments on some methods
//                - added timeout value for read operations (thanks mem @ Arduino forums)
//     2011-07-30 - changed read/write function structures to return success or byte
counts
//                - made all methods static for multi-device memory savings
//     2011-07-28 - initial release

/* ============================================
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 _I2CDEV_H_
#define _I2CDEV_H_

// -----------------------------------------------------------------------------
// I2C interface implementation setting
// -----------------------------------------------------------------------------
#define I2CDEV_IMPLEMENTATION       I2CDEV_ARDUINO_WIRE

// comment this out if you are using a non-optimal IDE/implementation setting
// but want the compiler to shut up about it
#define I2CDEV_IMPLEMENTATION_WARNINGS

// -----------------------------------------------------------------------------
// I2C interface implementation options
// -----------------------------------------------------------------------------
#define I2CDEV_ARDUINO_WIRE         1 // Wire object from Arduino
#define I2CDEV_BUILTIN_NBWIRE       2 // Tweaked Wire object from Gene Knight's NBWire project
                                    // ^^^ NBWire implementation is still buggy w/some interrupts!
#define I2CDEV_BUILTIN_FASTWIRE     3 // FastWire object from Francesco Ferrara's project
                                    // ^^^ FastWire implementation in I2Cdev is INCOMPLETE!

// -----------------------------------------------------------------------------
// Arduino-style "Serial.print" debug constant (uncomment to enable)
// -----------------------------------------------------------------------------
//#define I2CDEV_SERIAL_DEBUG

#ifdef ARDUINO
    #if ARDUINO < 100
        #include "WProgram.h"
    #else
        #include "Arduino.h"
    #endif
    #if I2CDEV_IMPLEMENTATION == I2CDEV_ARDUINO_WIRE
        #include <Wire.h>
    #endif
#else
    #include "ArduinoWrapper.h"
```

```
#endif

// 1000ms default read timeout (modify with "I2Cdev::readTimeout = [ms];")
#define I2CDEV_DEFAULT_READ_TIMEOUT     1000

class I2Cdev {
    public:
        I2Cdev();

        static int8_t readBit(uint8_t devAddr, uint8_t regAddr, uint8_t bitNum, uint8_t *data,
uint16_t timeout=I2Cdev::readTimeout);
        static int8_t readBitW(uint8_t devAddr, uint8_t regAddr, uint8_t bitNum, uint16_t
*data, uint16_t timeout=I2Cdev::readTimeout);
        static int8_t readBits(uint8_t devAddr, uint8_t regAddr, uint8_t bitStart, uint8_t
length, uint8_t *data, uint16_t timeout=I2Cdev::readTimeout);
        static int8_t readBitsW(uint8_t devAddr, uint8_t regAddr, uint8_t bitStart, uint8_t
length, uint16_t *data, uint16_t timeout=I2Cdev::readTimeout);
        static int8_t readByte(uint8_t devAddr, uint8_t regAddr, uint8_t *data, uint16_t
timeout=I2Cdev::readTimeout);
        static int8_t readWord(uint8_t devAddr, uint8_t regAddr, uint16_t *data, uint16_t
timeout=I2Cdev::readTimeout);
        static int8_t readBytes(uint8_t devAddr, uint8_t regAddr, uint8_t length, uint8_t
*data, uint16_t timeout=I2Cdev::readTimeout);
        static int8_t readWords(uint8_t devAddr, uint8_t regAddr, uint8_t length, uint16_t
*data, uint16_t timeout=I2Cdev::readTimeout);

        static bool writeBit(uint8_t devAddr, uint8_t regAddr, uint8_t bitNum, uint8_t data);
        static bool writeBitW(uint8_t devAddr, uint8_t regAddr, uint8_t bitNum, uint16_t
data);
        static bool writeBits(uint8_t devAddr, uint8_t regAddr, uint8_t bitStart, uint8_t
length, uint8_t data);
        static bool writeBitsW(uint8_t devAddr, uint8_t regAddr, uint8_t bitStart, uint8_t
length, uint16_t data);
        static bool writeByte(uint8_t devAddr, uint8_t regAddr, uint8_t data);
        static bool writeWord(uint8_t devAddr, uint8_t regAddr, uint16_t data);
        static bool writeBytes(uint8_t devAddr, uint8_t regAddr, uint8_t length, uint8_t
*data);
        static bool writeWords(uint8_t devAddr, uint8_t regAddr, uint8_t length, uint16_t
*data);

        static uint16_t readTimeout;
};

#if I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_FASTWIRE
    ///////////////////////
    // FastWire 0.2
    // This is a library to help faster programs to read I2C devices.
    // Copyright(C) 2011
    // Francesco Ferrara
    ///////////////////////
```

```
    /* Master */
    #define TW_START         0x08
    #define TW_REP_START     0x10

    /* Master Transmitter */
    #define TW_MT_SLA_ACK    0x18
    #define TW_MT_SLA_NACK   0x20
    #define TW_MT_DATA_ACK   0x28
    #define TW_MT_DATA_NACK  0x30
    #define TW_MT_ARB_LOST   0x38

    /* Master Receiver */
    #define TW_MR_ARB_LOST   0x38
    #define TW_MR_SLA_ACK    0x40
    #define TW_MR_SLA_NACK   0x48
    #define TW_MR_DATA_ACK   0x50
    #define TW_MR_DATA_NACK  0x58

    #define TW_OK            0
    #define TW_ERROR         1

    class Fastwire {
        private:
            static boolean waitInt();

        public:
            static void setup(int khz, boolean pullup);
            static byte write(byte device, byte address, byte value);
            static byte readBuf(byte device, byte address, byte *data, byte num);
    };
#endif

#if I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_NBWIRE
    // NBWire implementation based heavily on code by Gene Knight
<Gene@Telobot.com>
    // Originally posted on the Arduino forum at
http://arduino.cc/forum/index.php/topic,70705.0.html
    // Originally offered to the i2cdevlib project at
http://arduino.cc/forum/index.php/topic,68210.30.html

    #define NBWIRE_BUFFER_LENGTH 32

    class TwoWire {
        private:
            static uint8_t rxBuffer[];
            static uint8_t rxBufferIndex;
            static uint8_t rxBufferLength;

            static uint8_t txAddress;
            static uint8_t txBuffer[];
            static uint8_t txBufferIndex;
```

```
        static uint8_t txBufferLength;

        // static uint8_t transmitting;
        static void (*user_onRequest)(void);
        static void (*user_onReceive)(int);
        static void onRequestService(void);
        static void onReceiveService(uint8_t*, int);

      public:
        TwoWire();
        void begin();
        void begin(uint8_t);
        void begin(int);
        void beginTransmission(uint8_t);
        //void beginTransmission(int);
        uint8_t endTransmission(uint16_t timeout=0);
        void nbendTransmission(void (*function)(int)) ;
        uint8_t requestFrom(uint8_t, int, uint16_t timeout=0);
        //uint8_t requestFrom(int, int);
        void nbrequestFrom(uint8_t, int, void (*function)(int));
        void send(uint8_t);
        void send(uint8_t*, uint8_t);
        //void send(int);
        void send(char*);
        uint8_t available(void);
        uint8_t receive(void);
        void onReceive(void (*)(int));
        void onRequest(void (*)(void));
    };

    #define TWI_READY   0
    #define TWI_MRX     1
    #define TWI_MTX     2
    #define TWI_SRX     3
    #define TWI_STX     4

    #define TW_WRITE    0
    #define TW_READ     1

    #define TW_MT_SLA_NACK      0x20
    #define TW_MT_DATA_NACK     0x30

    #define CPU_FREQ           16000000L
    #define TWI_FREQ           100000L
    #define TWI_BUFFER_LENGTH   32

    /* TWI Status is in TWSR, in the top 5 bits: TWS7 - TWS3 */

    #define TW_STATUS_MASK
(_BV(TWS7)|_BV(TWS6)|_BV(TWS5)|_BV(TWS4)|_BV(TWS3))
    #define TW_STATUS                (TWSR & TW_STATUS_MASK)
```

```
    #define TW_START                0x08
    #define TW_REP_START            0x10
    #define TW_MT_SLA_ACK           0x18
    #define TW_MT_SLA_NACK          0x20
    #define TW_MT_DATA_ACK          0x28
    #define TW_MT_DATA_NACK         0x30
    #define TW_MT_ARB_LOST          0x38
    #define TW_MR_ARB_LOST          0x38
    #define TW_MR_SLA_ACK           0x40
    #define TW_MR_SLA_NACK          0x48
    #define TW_MR_DATA_ACK          0x50
    #define TW_MR_DATA_NACK         0x58
    #define TW_ST_SLA_ACK           0xA8
    #define TW_ST_ARB_LOST_SLA_ACK  0xB0
    #define TW_ST_DATA_ACK          0xB8
    #define TW_ST_DATA_NACK         0xC0
    #define TW_ST_LAST_DATA         0xC8
    #define TW_SR_SLA_ACK           0x60
    #define TW_SR_ARB_LOST_SLA_ACK  0x68
    #define TW_SR_GCALL_ACK         0x70
    #define TW_SR_ARB_LOST_GCALL_ACK 0x78
    #define TW_SR_DATA_ACK          0x80
    #define TW_SR_DATA_NACK         0x88
    #define TW_SR_GCALL_DATA_ACK    0x90
    #define TW_SR_GCALL_DATA_NACK   0x98
    #define TW_SR_STOP              0xA0
    #define TW_NO_INFO              0xF8
    #define TW_BUS_ERROR            0x00

    //#define _MMIO_BYTE(mem_addr) (*(volatile uint8_t *)(mem_addr))
    //#define _SFR_BYTE(sfr) _MMIO_BYTE(_SFR_ADDR(sfr))

    #ifndef sbi // set bit
        #define sbi(sfr, bit) (_SFR_BYTE(sfr) |= _BV(bit))
    #endif // sbi

    #ifndef cbi // clear bit
        #define cbi(sfr, bit) (_SFR_BYTE(sfr) &= ~_BV(bit))
    #endif // cbi

    extern TwoWire Wire;

#endif // I2CDEV_IMPLEMENTATION == I2CDEV_BUILTIN_NBWIRE

#endif /* _I2CDEV_H_ */

//end I2Cdev.h
////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////
//MPU6050.h
```

```
// I2Cdev library collection - MPU6050 I2C device class
// Based on InvenSense MPU-6050 register map document rev. 2.0, 5/19/2011 (RM-MPU-6000A-00)
// 10/3/2011 by Jeff Rowberg <jeff@rowberg.net>
// Updates should (hopefully) always be available at https://github.com/jrowberg/i2cdevlib
//
// Changelog:
//     ... - ongoing debug release

// NOTE: THIS IS ONLY A PARIAL RELEASE. THIS DEVICE CLASS IS CURRENTLY UNDERGOING ACTIVE
// DEVELOPMENT AND IS STILL MISSING SOME IMPORTANT FEATURES. PLEASE KEEP THIS IN MIND IF
// YOU DECIDE TO USE THIS PARTICULAR CODE FOR ANYTHING.

/* ============================================
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>
```

```
//Magnetometer Registers
#define MPU9150_RA_MAG_ADDRESS          0x0C
#define MPU9150_RA_MAG_XOUT_L           0x03
#define MPU9150_RA_MAG_XOUT_H           0x04
#define MPU9150_RA_MAG_YOUT_L           0x05
#define MPU9150_RA_MAG_YOUT_H           0x06
#define MPU9150_RA_MAG_ZOUT_L           0x07
#define MPU9150_RA_MAG_ZOUT_H           0x08

#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

#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
#define MPU6050_RA_I2C_SLV4_ADDR    0x31
#define MPU6050_RA_I2C_SLV4_REG     0x32
#define MPU6050_RA_I2C_SLV4_DO      0x33
#define MPU6050_RA_I2C_SLV4_CTRL    0x34
#define MPU6050_RA_I2C_SLV4_DI      0x35
#define MPU6050_RA_I2C_MST_STATUS   0x36
#define MPU6050_RA_INT_PIN_CFG      0x37
#define MPU6050_RA_INT_ENABLE       0x38
#define MPU6050_RA_DMP_INT_STATUS   0x39
#define MPU6050_RA_INT_STATUS       0x3A
#define MPU6050_RA_ACCEL_XOUT_H     0x3B
#define MPU6050_RA_ACCEL_XOUT_L     0x3C
#define MPU6050_RA_ACCEL_YOUT_H     0x3D
#define MPU6050_RA_ACCEL_YOUT_L     0x3E
#define MPU6050_RA_ACCEL_ZOUT_H     0x3F
#define MPU6050_RA_ACCEL_ZOUT_L     0x40
#define MPU6050_RA_TEMP_OUT_H       0x41
#define MPU6050_RA_TEMP_OUT_L       0x42
#define MPU6050_RA_GYRO_XOUT_H      0x43
#define MPU6050_RA_GYRO_XOUT_L      0x44
#define MPU6050_RA_GYRO_YOUT_H      0x45
#define MPU6050_RA_GYRO_YOUT_L      0x46
#define MPU6050_RA_GYRO_ZOUT_H      0x47
#define MPU6050_RA_GYRO_ZOUT_L      0x48
#define MPU6050_RA_EXT_SENS_DATA_00 0x49
#define MPU6050_RA_EXT_SENS_DATA_01 0x4A
#define MPU6050_RA_EXT_SENS_DATA_02 0x4B
#define MPU6050_RA_EXT_SENS_DATA_03 0x4C
#define MPU6050_RA_EXT_SENS_DATA_04 0x4D
#define MPU6050_RA_EXT_SENS_DATA_05 0x4E
#define MPU6050_RA_EXT_SENS_DATA_06 0x4F
#define MPU6050_RA_EXT_SENS_DATA_07 0x50
#define MPU6050_RA_EXT_SENS_DATA_08 0x51
#define MPU6050_RA_EXT_SENS_DATA_09 0x52
#define MPU6050_RA_EXT_SENS_DATA_10 0x53
#define MPU6050_RA_EXT_SENS_DATA_11 0x54
#define MPU6050_RA_EXT_SENS_DATA_12 0x55
#define MPU6050_RA_EXT_SENS_DATA_13 0x56
#define MPU6050_RA_EXT_SENS_DATA_14 0x57
#define MPU6050_RA_EXT_SENS_DATA_15 0x58
#define MPU6050_RA_EXT_SENS_DATA_16 0x59
#define MPU6050_RA_EXT_SENS_DATA_17 0x5A
#define MPU6050_RA_EXT_SENS_DATA_18 0x5B
```

```
#define MPU6050_RA_EXT_SENS_DATA_19 0x5C
#define MPU6050_RA_EXT_SENS_DATA_20 0x5D
#define MPU6050_RA_EXT_SENS_DATA_21 0x5E
#define MPU6050_RA_EXT_SENS_DATA_22 0x5F
#define MPU6050_RA_EXT_SENS_DATA_23 0x60
#define MPU6050_RA_MOT_DETECT_STATUS    0x61
#define MPU6050_RA_I2C_SLV0_DO      0x63
#define MPU6050_RA_I2C_SLV1_DO      0x64
#define MPU6050_RA_I2C_SLV2_DO      0x65
#define MPU6050_RA_I2C_SLV3_DO      0x66
#define MPU6050_RA_I2C_MST_DELAY_CTRL   0x67
#define MPU6050_RA_SIGNAL_PATH_RESET    0x68
#define MPU6050_RA_MOT_DETECT_CTRL      0x69
#define MPU6050_RA_USER_CTRL        0x6A
#define MPU6050_RA_PWR_MGMT_1       0x6B
#define MPU6050_RA_PWR_MGMT_2       0x6C
#define MPU6050_RA_BANK_SEL         0x6D
#define MPU6050_RA_MEM_START_ADDR   0x6E
#define MPU6050_RA_MEM_R_W          0x6F
#define MPU6050_RA_DMP_CFG_1        0x70
#define MPU6050_RA_DMP_CFG_2        0x71
#define MPU6050_RA_FIFO_COUNTH      0x72
#define MPU6050_RA_FIFO_COUNTL      0x73
#define MPU6050_RA_FIFO_R_W         0x74
#define MPU6050_RA_WHO_AM_I         0x75

#define MPU6050_TC_PWR_MODE_BIT     7
#define MPU6050_TC_OFFSET_BIT       6
#define MPU6050_TC_OFFSET_LENGTH    6
#define MPU6050_TC_OTP_BNK_VLD_BIT  0

#define MPU6050_VDDIO_LEVEL_VLOGIC  0
#define MPU6050_VDDIO_LEVEL_VDD     1

#define MPU6050_CFG_EXT_SYNC_SET_BIT    5
#define MPU6050_CFG_EXT_SYNC_SET_LENGTH 3
#define MPU6050_CFG_DLPF_CFG_BIT    2
#define MPU6050_CFG_DLPF_CFG_LENGTH 3

#define MPU6050_EXT_SYNC_DISABLED       0x0
#define MPU6050_EXT_SYNC_TEMP_OUT_L     0x1
#define MPU6050_EXT_SYNC_GYRO_XOUT_L    0x2
#define MPU6050_EXT_SYNC_GYRO_YOUT_L    0x3
#define MPU6050_EXT_SYNC_GYRO_ZOUT_L    0x4
#define MPU6050_EXT_SYNC_ACCEL_XOUT_L   0x5
#define MPU6050_EXT_SYNC_ACCEL_YOUT_L   0x6
#define MPU6050_EXT_SYNC_ACCEL_ZOUT_L   0x7

#define MPU6050_DLPF_BW_256         0x00
#define MPU6050_DLPF_BW_188         0x01
#define MPU6050_DLPF_BW_98          0x02
```

```
#define MPU6050_DLPF_BW_42          0x03
#define MPU6050_DLPF_BW_20          0x04
#define MPU6050_DLPF_BW_10          0x05
#define MPU6050_DLPF_BW_5           0x06

#define MPU6050_GCONFIG_FS_SEL_BIT      4
#define MPU6050_GCONFIG_FS_SEL_LENGTH   2

#define MPU6050_GYRO_FS_250         0x00
#define MPU6050_GYRO_FS_500         0x01
#define MPU6050_GYRO_FS_1000        0x02
#define MPU6050_GYRO_FS_2000        0x03

#define MPU6050_ACONFIG_XA_ST_BIT           7
#define MPU6050_ACONFIG_YA_ST_BIT           6
#define MPU6050_ACONFIG_ZA_ST_BIT           5
#define MPU6050_ACONFIG_AFS_SEL_BIT         4
#define MPU6050_ACONFIG_AFS_SEL_LENGTH      2
#define MPU6050_ACONFIG_ACCEL_HPF_BIT       2
#define MPU6050_ACONFIG_ACCEL_HPF_LENGTH    3

#define MPU6050_ACCEL_FS_2          0x00
#define MPU6050_ACCEL_FS_4          0x01
#define MPU6050_ACCEL_FS_8          0x02
#define MPU6050_ACCEL_FS_16         0x03

#define MPU6050_DHPF_RESET          0x00
#define MPU6050_DHPF_5              0x01
#define MPU6050_DHPF_2P5            0x02
#define MPU6050_DHPF_1P25           0x03
#define MPU6050_DHPF_0P63           0x04
#define MPU6050_DHPF_HOLD           0x07

#define MPU6050_TEMP_FIFO_EN_BIT    7
#define MPU6050_XG_FIFO_EN_BIT      6
#define MPU6050_YG_FIFO_EN_BIT      5
#define MPU6050_ZG_FIFO_EN_BIT      4
#define MPU6050_ACCEL_FIFO_EN_BIT   3
#define MPU6050_SLV2_FIFO_EN_BIT    2
#define MPU6050_SLV1_FIFO_EN_BIT    1
#define MPU6050_SLV0_FIFO_EN_BIT    0

#define MPU6050_MULT_MST_EN_BIT     7
#define MPU6050_WAIT_FOR_ES_BIT     6
#define MPU6050_SLV_3_FIFO_EN_BIT   5
#define MPU6050_I2C_MST_P_NSR_BIT   4
#define MPU6050_I2C_MST_CLK_BIT     3
#define MPU6050_I2C_MST_CLK_LENGTH  4

#define MPU6050_CLOCK_DIV_348       0x0
#define MPU6050_CLOCK_DIV_333       0x1
```

```
#define MPU6050_CLOCK_DIV_320       0x2
#define MPU6050_CLOCK_DIV_308       0x3
#define MPU6050_CLOCK_DIV_296       0x4
#define MPU6050_CLOCK_DIV_286       0x5
#define MPU6050_CLOCK_DIV_276       0x6
#define MPU6050_CLOCK_DIV_267       0x7
#define MPU6050_CLOCK_DIV_258       0x8
#define MPU6050_CLOCK_DIV_500       0x9
#define MPU6050_CLOCK_DIV_471       0xA
#define MPU6050_CLOCK_DIV_444       0xB
#define MPU6050_CLOCK_DIV_421       0xC
#define MPU6050_CLOCK_DIV_400       0xD
#define MPU6050_CLOCK_DIV_381       0xE
#define MPU6050_CLOCK_DIV_364       0xF

#define MPU6050_I2C_SLV_RW_BIT      7
#define MPU6050_I2C_SLV_ADDR_BIT    6
#define MPU6050_I2C_SLV_ADDR_LENGTH 7
#define MPU6050_I2C_SLV_EN_BIT      7
#define MPU6050_I2C_SLV_BYTE_SW_BIT 6
#define MPU6050_I2C_SLV_REG_DIS_BIT 5
#define MPU6050_I2C_SLV_GRP_BIT     4
#define MPU6050_I2C_SLV_LEN_BIT     3
#define MPU6050_I2C_SLV_LEN_LENGTH  4

#define MPU6050_I2C_SLV4_RW_BIT         7
#define MPU6050_I2C_SLV4_ADDR_BIT       6
#define MPU6050_I2C_SLV4_ADDR_LENGTH    7
#define MPU6050_I2C_SLV4_EN_BIT         7
#define MPU6050_I2C_SLV4_INT_EN_BIT     6
#define MPU6050_I2C_SLV4_REG_DIS_BIT    5
#define MPU6050_I2C_SLV4_MST_DLY_BIT    4
#define MPU6050_I2C_SLV4_MST_DLY_LENGTH 5

#define MPU6050_MST_PASS_THROUGH_BIT    7
#define MPU6050_MST_I2C_SLV4_DONE_BIT   6
#define MPU6050_MST_I2C_LOST_ARB_BIT    5
#define MPU6050_MST_I2C_SLV4_NACK_BIT   4
#define MPU6050_MST_I2C_SLV3_NACK_BIT   3
#define MPU6050_MST_I2C_SLV2_NACK_BIT   2
#define MPU6050_MST_I2C_SLV1_NACK_BIT   1
#define MPU6050_MST_I2C_SLV0_NACK_BIT   0

#define MPU6050_INTCFG_INT_LEVEL_BIT        7
#define MPU6050_INTCFG_INT_OPEN_BIT         6
#define MPU6050_INTCFG_LATCH_INT_EN_BIT     5
#define MPU6050_INTCFG_INT_RD_CLEAR_BIT     4
#define MPU6050_INTCFG_FSYNC_INT_LEVEL_BIT  3
#define MPU6050_INTCFG_FSYNC_INT_EN_BIT     2
#define MPU6050_INTCFG_I2C_BYPASS_EN_BIT    1
#define MPU6050_INTCFG_CLKOUT_EN_BIT        0
```

```
#define MPU6050_INTMODE_ACTIVEHIGH  0x00
#define MPU6050_INTMODE_ACTIVELOW   0x01

#define MPU6050_INTDRV_PUSHPULL     0x00
#define MPU6050_INTDRV_OPENDRAIN    0x01

#define MPU6050_INTLATCH_50USPULSE  0x00
#define MPU6050_INTLATCH_WAITCLEAR  0x01

#define MPU6050_INTCLEAR_STATUSREAD 0x00
#define MPU6050_INTCLEAR_ANYREAD    0x01

#define MPU6050_INTERRUPT_FF_BIT          7
#define MPU6050_INTERRUPT_MOT_BIT         6
#define MPU6050_INTERRUPT_ZMOT_BIT        5
#define MPU6050_INTERRUPT_FIFO_OFLOW_BIT  4
#define MPU6050_INTERRUPT_I2C_MST_INT_BIT 3
#define MPU6050_INTERRUPT_PLL_RDY_INT_BIT 2
#define MPU6050_INTERRUPT_DMP_INT_BIT     1
#define MPU6050_INTERRUPT_DATA_RDY_BIT    0

// TODO: figure out what these actually do
// UMPL source code is not very obivous
#define MPU6050_DMPINT_5_BIT          5
#define MPU6050_DMPINT_4_BIT          4
#define MPU6050_DMPINT_3_BIT          3
#define MPU6050_DMPINT_2_BIT          2
#define MPU6050_DMPINT_1_BIT          1
#define MPU6050_DMPINT_0_BIT          0

#define MPU6050_MOTION_MOT_XNEG_BIT   7
#define MPU6050_MOTION_MOT_XPOS_BIT   6
#define MPU6050_MOTION_MOT_YNEG_BIT   5
#define MPU6050_MOTION_MOT_YPOS_BIT   4
#define MPU6050_MOTION_MOT_ZNEG_BIT   3
#define MPU6050_MOTION_MOT_ZPOS_BIT   2
#define MPU6050_MOTION_MOT_ZRMOT_BIT  0

#define MPU6050_DELAYCTRL_DELAY_ES_SHADOW_BIT  7
#define MPU6050_DELAYCTRL_I2C_SLV4_DLY_EN_BIT  4
#define MPU6050_DELAYCTRL_I2C_SLV3_DLY_EN_BIT  3
#define MPU6050_DELAYCTRL_I2C_SLV2_DLY_EN_BIT  2
#define MPU6050_DELAYCTRL_I2C_SLV1_DLY_EN_BIT  1
#define MPU6050_DELAYCTRL_I2C_SLV0_DLY_EN_BIT  0

#define MPU6050_PATHRESET_GYRO_RESET_BIT   2
#define MPU6050_PATHRESET_ACCEL_RESET_BIT  1
#define MPU6050_PATHRESET_TEMP_RESET_BIT   0

#define MPU6050_DETECT_ACCEL_ON_DELAY_BIT      5
```

```
#define MPU6050_DETECT_ACCEL_ON_DELAY_LENGTH    2
#define MPU6050_DETECT_FF_COUNT_BIT             3
#define MPU6050_DETECT_FF_COUNT_LENGTH          2
#define MPU6050_DETECT_MOT_COUNT_BIT            1
#define MPU6050_DETECT_MOT_COUNT_LENGTH         2

#define MPU6050_DETECT_DECREMENT_RESET  0x0
#define MPU6050_DETECT_DECREMENT_1      0x1
#define MPU6050_DETECT_DECREMENT_2      0x2
#define MPU6050_DETECT_DECREMENT_4      0x3

#define MPU6050_USERCTRL_DMP_EN_BIT             7
#define MPU6050_USERCTRL_FIFO_EN_BIT            6
#define MPU6050_USERCTRL_I2C_MST_EN_BIT         5
#define MPU6050_USERCTRL_I2C_IF_DIS_BIT         4
#define MPU6050_USERCTRL_DMP_RESET_BIT          3
#define MPU6050_USERCTRL_FIFO_RESET_BIT         2
#define MPU6050_USERCTRL_I2C_MST_RESET_BIT      1
#define MPU6050_USERCTRL_SIG_COND_RESET_BIT     0

#define MPU6050_PWR1_DEVICE_RESET_BIT   7
#define MPU6050_PWR1_SLEEP_BIT          6
#define MPU6050_PWR1_CYCLE_BIT          5
#define MPU6050_PWR1_TEMP_DIS_BIT       3
#define MPU6050_PWR1_CLKSEL_BIT         2
#define MPU6050_PWR1_CLKSEL_LENGTH      3

#define MPU6050_CLOCK_INTERNAL          0x00
#define MPU6050_CLOCK_PLL_XGYRO         0x01
#define MPU6050_CLOCK_PLL_YGYRO         0x02
#define MPU6050_CLOCK_PLL_ZGYRO         0x03
#define MPU6050_CLOCK_PLL_EXT32K        0x04
#define MPU6050_CLOCK_PLL_EXT19M        0x05
#define MPU6050_CLOCK_KEEP_RESET        0x07

#define MPU6050_PWR2_LP_WAKE_CTRL_BIT       7
#define MPU6050_PWR2_LP_WAKE_CTRL_LENGTH    2
#define MPU6050_PWR2_STBY_XA_BIT            5
#define MPU6050_PWR2_STBY_YA_BIT            4
#define MPU6050_PWR2_STBY_ZA_BIT            3
#define MPU6050_PWR2_STBY_XG_BIT            2
#define MPU6050_PWR2_STBY_YG_BIT            1
#define MPU6050_PWR2_STBY_ZG_BIT            0

#define MPU6050_WAKE_FREQ_1P25      0x0
#define MPU6050_WAKE_FREQ_2P5       0x1
#define MPU6050_WAKE_FREQ_5         0x2
#define MPU6050_WAKE_FREQ_10        0x3

#define MPU6050_BANKSEL_PRFTCH_EN_BIT       6
#define MPU6050_BANKSEL_CFG_USER_BANK_BIT   5
```

```
#define MPU6050_BANKSEL_MEM_SEL_BIT       4
#define MPU6050_BANKSEL_MEM_SEL_LENGTH    5

#define MPU6050_WHO_AM_I_BIT       6
#define MPU6050_WHO_AM_I_LENGTH    6

#define MPU6050_DMP_MEMORY_BANKS       8
#define MPU6050_DMP_MEMORY_BANK_SIZE   256
#define MPU6050_DMP_MEMORY_CHUNK_SIZE  16

// note: DMP code memory blocks defined at end of header file

class MPU6050 {
    public:
        MPU6050();
        MPU6050(uint8_t address);

        void initialize();
        bool testConnection();

        // AUX_VDDIO register
        uint8_t getAuxVDDIOLevel();
        void setAuxVDDIOLevel(uint8_t level);

        // SMPLRT_DIV register
        uint8_t getRate();
        void setRate(uint8_t rate);

        // CONFIG register
        uint8_t getExternalFrameSync();
        void setExternalFrameSync(uint8_t sync);
        uint8_t getDLPFMode();
        void setDLPFMode(uint8_t bandwidth);

        // GYRO_CONFIG register
        uint8_t getFullScaleGyroRange();
        void setFullScaleGyroRange(uint8_t range);

        // ACCEL_CONFIG register
        bool getAccelXSelfTest();
        void setAccelXSelfTest(bool enabled);
        bool getAccelYSelfTest();
        void setAccelYSelfTest(bool enabled);
        bool getAccelZSelfTest();
        void setAccelZSelfTest(bool enabled);
        uint8_t getFullScaleAccelRange();
        void setFullScaleAccelRange(uint8_t range);
        uint8_t getDHPFMode();
        void setDHPFMode(uint8_t mode);

        // FF_THR register
```

```
uint8_t getFreefallDetectionThreshold();
void setFreefallDetectionThreshold(uint8_t threshold);

// FF_DUR register
uint8_t getFreefallDetectionDuration();
void setFreefallDetectionDuration(uint8_t duration);

// MOT_THR register
uint8_t getMotionDetectionThreshold();
void setMotionDetectionThreshold(uint8_t threshold);

// MOT_DUR register
uint8_t getMotionDetectionDuration();
void setMotionDetectionDuration(uint8_t duration);

// ZRMOT_THR register
uint8_t getZeroMotionDetectionThreshold();
void setZeroMotionDetectionThreshold(uint8_t threshold);

// ZRMOT_DUR register
uint8_t getZeroMotionDetectionDuration();
void setZeroMotionDetectionDuration(uint8_t duration);

// FIFO_EN register
bool getTempFIFOEnabled();
void setTempFIFOEnabled(bool enabled);
bool getXGyroFIFOEnabled();
void setXGyroFIFOEnabled(bool enabled);
bool getYGyroFIFOEnabled();
void setYGyroFIFOEnabled(bool enabled);
bool getZGyroFIFOEnabled();
void setZGyroFIFOEnabled(bool enabled);
bool getAccelFIFOEnabled();
void setAccelFIFOEnabled(bool enabled);
bool getSlave2FIFOEnabled();
void setSlave2FIFOEnabled(bool enabled);
bool getSlave1FIFOEnabled();
void setSlave1FIFOEnabled(bool enabled);
bool getSlave0FIFOEnabled();
void setSlave0FIFOEnabled(bool enabled);

// I2C_MST_CTRL register
bool getMultiMasterEnabled();
void setMultiMasterEnabled(bool enabled);
bool getWaitForExternalSensorEnabled();
void setWaitForExternalSensorEnabled(bool enabled);
bool getSlave3FIFOEnabled();
void setSlave3FIFOEnabled(bool enabled);
bool getSlaveReadWriteTransitionEnabled();
void setSlaveReadWriteTransitionEnabled(bool enabled);
uint8_t getMasterClockSpeed();
```

```
void setMasterClockSpeed(uint8_t speed);

// I2C_SLV* registers (Slave 0-3)
uint8_t getSlaveAddress(uint8_t num);
void setSlaveAddress(uint8_t num, uint8_t address);
uint8_t getSlaveRegister(uint8_t num);
void setSlaveRegister(uint8_t num, uint8_t reg);
bool getSlaveEnabled(uint8_t num);
void setSlaveEnabled(uint8_t num, bool enabled);
bool getSlaveWordByteSwap(uint8_t num);
void setSlaveWordByteSwap(uint8_t num, bool enabled);
bool getSlaveWriteMode(uint8_t num);
void setSlaveWriteMode(uint8_t num, bool mode);
bool getSlaveWordGroupOffset(uint8_t num);
void setSlaveWordGroupOffset(uint8_t num, bool enabled);
uint8_t getSlaveDataLength(uint8_t num);
void setSlaveDataLength(uint8_t num, uint8_t length);

// I2C_SLV* registers (Slave 4)
uint8_t getSlave4Address();
void setSlave4Address(uint8_t address);
uint8_t getSlave4Register();
void setSlave4Register(uint8_t reg);
void setSlave4OutputByte(uint8_t data);
bool getSlave4Enabled();
void setSlave4Enabled(bool enabled);
bool getSlave4InterruptEnabled();
void setSlave4InterruptEnabled(bool enabled);
bool getSlave4WriteMode();
void setSlave4WriteMode(bool mode);
uint8_t getSlave4MasterDelay();
void setSlave4MasterDelay(uint8_t delay);
uint8_t getSlate4InputByte();

// I2C_MST_STATUS register
bool getPassthroughStatus();
bool getSlave4IsDone();
bool getLostArbitration();
bool getSlave4Nack();
bool getSlave3Nack();
bool getSlave2Nack();
bool getSlave1Nack();
bool getSlave0Nack();

// INT_PIN_CFG register
bool getInterruptMode();
void setInterruptMode(bool mode);
bool getInterruptDrive();
void setInterruptDrive(bool drive);
bool getInterruptLatch();
void setInterruptLatch(bool latch);
```

```
        bool getInterruptLatchClear();
        void setInterruptLatchClear(bool clear);
        bool getFSyncInterruptLevel();
        void setFSyncInterruptLevel(bool level);
        bool getFSyncInterruptEnabled();
        void setFSyncInterruptEnabled(bool enabled);
        bool getI2CBypassEnabled();
        void setI2CBypassEnabled(bool enabled);
        bool getClockOutputEnabled();
        void setClockOutputEnabled(bool enabled);

        // INT_ENABLE register
        uint8_t getIntEnabled();
        void setIntEnabled(uint8_t enabled);
        bool getIntFreefallEnabled();
        void setIntFreefallEnabled(bool enabled);
        bool getIntMotionEnabled();
        void setIntMotionEnabled(bool enabled);
        bool getIntZeroMotionEnabled();
        void setIntZeroMotionEnabled(bool enabled);
        bool getIntFIFOBufferOverflowEnabled();
        void setIntFIFOBufferOverflowEnabled(bool enabled);
        bool getIntI2CMasterEnabled();
        void setIntI2CMasterEnabled(bool enabled);
        bool getIntDataReadyEnabled();
        void setIntDataReadyEnabled(bool enabled);

        // INT_STATUS register
        uint8_t getIntStatus();
        bool getIntFreefallStatus();
        bool getIntMotionStatus();
        bool getIntZeroMotionStatus();
        bool getIntFIFOBufferOverflowStatus();
        bool getIntI2CMasterStatus();
        bool getIntDataReadyStatus();

        // ACCEL_*OUT_* registers
        void getMotion9(int16_t* ax, int16_t* ay, int16_t* az, int16_t* gx, int16_t* gy,
int16_t* gz, int16_t* mx, int16_t* my, int16_t* mz);
        void getMotion6(int16_t* ax, int16_t* ay, int16_t* az, int16_t* gx, int16_t* gy,
int16_t* gz);
        void getAcceleration(int16_t* x, int16_t* y, int16_t* z);
        int16_t getAccelerationX();
        int16_t getAccelerationY();
        int16_t getAccelerationZ();

        // TEMP_OUT_* registers
        int16_t getTemperature();

        // GYRO_*OUT_* registers
        void getRotation(int16_t* x, int16_t* y, int16_t* z);
```

```
int16_t getRotationX();
int16_t getRotationY();
int16_t getRotationZ();

// EXT_SENS_DATA_* registers
uint8_t getExternalSensorByte(int position);
uint16_t getExternalSensorWord(int position);
uint32_t getExternalSensorDWord(int position);

// MOT_DETECT_STATUS register
bool getXNegMotionDetected();
bool getXPosMotionDetected();
bool getYNegMotionDetected();
bool getYPosMotionDetected();
bool getZNegMotionDetected();
bool getZPosMotionDetected();
bool getZeroMotionDetected();

// I2C_SLV*_DO register
void setSlaveOutputByte(uint8_t num, uint8_t data);

// I2C_MST_DELAY_CTRL register
bool getExternalShadowDelayEnabled();
void setExternalShadowDelayEnabled(bool enabled);
bool getSlaveDelayEnabled(uint8_t num);
void setSlaveDelayEnabled(uint8_t num, bool enabled);

// SIGNAL_PATH_RESET register
void resetGyroscopePath();
void resetAccelerometerPath();
void resetTemperaturePath();

// MOT_DETECT_CTRL register
uint8_t getAccelerometerPowerOnDelay();
void setAccelerometerPowerOnDelay(uint8_t delay);
uint8_t getFreefallDetectionCounterDecrement();
void setFreefallDetectionCounterDecrement(uint8_t decrement);
uint8_t getMotionDetectionCounterDecrement();
void setMotionDetectionCounterDecrement(uint8_t decrement);

// USER_CTRL register
bool getFIFOEnabled();
void setFIFOEnabled(bool enabled);
bool getI2CMasterModeEnabled();
void setI2CMasterModeEnabled(bool enabled);
void switchSPIEnabled(bool enabled);
void resetFIFO();
void resetI2CMaster();
void resetSensors();

// PWR_MGMT_1 register
```

```
void reset();
bool getSleepEnabled();
void setSleepEnabled(bool enabled);
bool getWakeCycleEnabled();
void setWakeCycleEnabled(bool enabled);
bool getTempSensorEnabled();
void setTempSensorEnabled(bool enabled);
uint8_t getClockSource();
void setClockSource(uint8_t source);

// PWR_MGMT_2 register
uint8_t getWakeFrequency();
void setWakeFrequency(uint8_t frequency);
bool getStandbyXAccelEnabled();
void setStandbyXAccelEnabled(bool enabled);
bool getStandbyYAccelEnabled();
void setStandbyYAccelEnabled(bool enabled);
bool getStandbyZAccelEnabled();
void setStandbyZAccelEnabled(bool enabled);
bool getStandbyXGyroEnabled();
void setStandbyXGyroEnabled(bool enabled);
bool getStandbyYGyroEnabled();
void setStandbyYGyroEnabled(bool enabled);
bool getStandbyZGyroEnabled();
void setStandbyZGyroEnabled(bool enabled);

// FIFO_COUNT_* registers
uint16_t getFIFOCount();

// FIFO_R_W register
uint8_t getFIFOByte();
void setFIFOByte(uint8_t data);
void getFIFOBytes(uint8_t *data, uint8_t length);

// WHO_AM_I register
uint8_t getDeviceID();
void setDeviceID(uint8_t id);

// ======== UNDOCUMENTED/DMP REGISTERS/METHODS ========

// XG_OFFS_TC register
uint8_t getOTPBankValid();
void setOTPBankValid(bool enabled);
int8_t getXGyroOffset();
void setXGyroOffset(int8_t offset);

// YG_OFFS_TC register
int8_t getYGyroOffset();
void setYGyroOffset(int8_t offset);

// ZG_OFFS_TC register
```

```
int8_t getZGyroOffset();
void setZGyroOffset(int8_t offset);

// X_FINE_GAIN register
int8_t getXFineGain();
void setXFineGain(int8_t gain);

// Y_FINE_GAIN register
int8_t getYFineGain();
void setYFineGain(int8_t gain);

// Z_FINE_GAIN register
int8_t getZFineGain();
void setZFineGain(int8_t gain);

// XA_OFFS_* registers
int16_t getXAccelOffset();
void setXAccelOffset(int16_t offset);

// YA_OFFS_* register
int16_t getYAccelOffset();
void setYAccelOffset(int16_t offset);

// ZA_OFFS_* register
int16_t getZAccelOffset();
void setZAccelOffset(int16_t offset);

// XG_OFFS_USR* registers
int16_t getXGyroOffsetUser();
void setXGyroOffsetUser(int16_t offset);

// YG_OFFS_USR* register
int16_t getYGyroOffsetUser();
void setYGyroOffsetUser(int16_t offset);

// ZG_OFFS_USR* register
int16_t getZGyroOffsetUser();
void setZGyroOffsetUser(int16_t offset);

// INT_ENABLE register (DMP functions)
bool getIntPLLReadyEnabled();
void setIntPLLReadyEnabled(bool enabled);
bool getIntDMPEnabled();
void setIntDMPEnabled(bool enabled);

// DMP_INT_STATUS
bool getDMPInt5Status();
bool getDMPInt4Status();
bool getDMPInt3Status();
bool getDMPInt2Status();
bool getDMPInt1Status();
```

```
        bool getDMPInt0Status();

        // INT_STATUS register (DMP functions)
        bool getIntPLLReadyStatus();
        bool getIntDMPStatus();

        // USER_CTRL register (DMP functions)
        bool getDMPEnabled();
        void setDMPEnabled(bool enabled);
        void resetDMP();

        // BANK_SEL register
        void setMemoryBank(uint8_t bank, bool prefetchEnabled=false, bool
userBank=false);

        // MEM_START_ADDR register
        void setMemoryStartAddress(uint8_t address);

        // MEM_R_W register
        uint8_t readMemoryByte();
        void writeMemoryByte(uint8_t data);
        void readMemoryBlock(uint8_t *data, uint16_t dataSize, uint8_t bank=0, uint8_t
address=0);
        bool writeMemoryBlock(const uint8_t *data, uint16_t dataSize, uint8_t bank=0,
uint8_t address=0, bool verify=true, bool useProgMem=false);
        bool writeProgMemoryBlock(const uint8_t *data, uint16_t dataSize, uint8_t bank=0,
uint8_t address=0, bool verify=true);

        bool writeDMPConfigurationSet(const uint8_t *data, uint16_t dataSize, bool
useProgMem=false);
        bool writeProgDMPConfigurationSet(const uint8_t *data, uint16_t dataSize);

        // DMP_CFG_1 register
        uint8_t getDMPConfig1();
        void setDMPConfig1(uint8_t config);

        // DMP_CFG_2 register
        uint8_t getDMPConfig2();
        void setDMPConfig2(uint8_t config);

        // special methods for MotionApps 2.0 implementation
        #ifdef MPU6050_INCLUDE_DMP_MOTIONAPPS20
            uint8_t *dmpPacketBuffer;
            uint16_t dmpPacketSize;

            uint8_t dmpInitialize();
            bool dmpPacketAvailable();

            uint8_t dmpSetFIFORate(uint8_t fifoRate);
            uint8_t dmpGetFIFORate();
            uint8_t dmpGetSampleStepSizeMS();
```

```
        uint8_t dmpGetSampleFrequency();
        int32_t dmpDecodeTemperature(int8_t tempReg);

        // Register callbacks after a packet of FIFO data is processed
        //uint8_t dmpRegisterFIFORateProcess(inv_obj_func func, int16_t priority);
        //uint8_t dmpUnregisterFIFORateProcess(inv_obj_func func);
        uint8_t dmpRunFIFORateProcesses();

        // Setup FIFO for various output
        uint8_t dmpSendQuaternion(uint_fast16_t accuracy);
        uint8_t dmpSendGyro(uint_fast16_t elements, uint_fast16_t accuracy);
        uint8_t dmpSendAccel(uint_fast16_t elements, uint_fast16_t accuracy);
        uint8_t dmpSendLinearAccel(uint_fast16_t elements, uint_fast16_t accuracy);
        uint8_t dmpSendLinearAccelInWorld(uint_fast16_t elements, uint_fast16_t
accuracy);
        uint8_t dmpSendControlData(uint_fast16_t elements, uint_fast16_t accuracy);
        uint8_t dmpSendSensorData(uint_fast16_t elements, uint_fast16_t accuracy);
        uint8_t dmpSendExternalSensorData(uint_fast16_t elements, uint_fast16_t
accuracy);
        uint8_t dmpSendGravity(uint_fast16_t elements, uint_fast16_t accuracy);
        uint8_t dmpSendPacketNumber(uint_fast16_t accuracy);
        uint8_t dmpSendQuantizedAccel(uint_fast16_t elements, uint_fast16_t
accuracy);
        uint8_t dmpSendEIS(uint_fast16_t elements, uint_fast16_t accuracy);

        // Get Fixed Point data from FIFO
        uint8_t dmpGetAccel(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGetAccel(int16_t *data, const uint8_t* packet=0);
        uint8_t dmpGetAccel(VectorInt16 *v, const uint8_t* packet=0);
        uint8_t dmpGetQuaternion(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGetQuaternion(int16_t *data, const uint8_t* packet=0);
        uint8_t dmpGetQuaternion(Quaternion *q, const uint8_t* packet=0);
        uint8_t dmpGet6AxisQuaternion(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGet6AxisQuaternion(int16_t *data, const uint8_t* packet=0);
        uint8_t dmpGet6AxisQuaternion(Quaternion *q, const uint8_t* packet=0);
        uint8_t dmpGetRelativeQuaternion(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGetRelativeQuaternion(int16_t *data, const uint8_t* packet=0);
        uint8_t dmpGetRelativeQuaternion(Quaternion *data, const uint8_t* packet=0);
        uint8_t dmpGetGyro(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGetGyro(int16_t *data, const uint8_t* packet=0);
        uint8_t dmpGetGyro(VectorInt16 *v, const uint8_t* packet=0);
        uint8_t dmpSetLinearAccelFilterCoefficient(float coef);
        uint8_t dmpGetLinearAccel(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGetLinearAccel(int16_t *data, const uint8_t* packet=0);
        uint8_t dmpGetLinearAccel(VectorInt16 *v, const uint8_t* packet=0);
        uint8_t dmpGetLinearAccel(VectorInt16 *v, VectorInt16 *vRaw, VectorFloat
*gravity);
        uint8_t dmpGetLinearAccelInWorld(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGetLinearAccelInWorld(int16_t *data, const uint8_t* packet=0);
        uint8_t dmpGetLinearAccelInWorld(VectorInt16 *v, const uint8_t* packet=0);
        uint8_t dmpGetLinearAccelInWorld(VectorInt16 *v, VectorInt16 *vReal,
```

```
Quaternion *q);
        uint8_t dmpGetGyroAndAccelSensor(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGetGyroAndAccelSensor(int16_t *data, const uint8_t* packet=0);
        uint8_t dmpGetGyroAndAccelSensor(VectorInt16 *g, VectorInt16 *a, const
uint8_t* packet=0);
        uint8_t dmpGetGyroSensor(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGetGyroSensor(int16_t *data, const uint8_t* packet=0);
        uint8_t dmpGetGyroSensor(VectorInt16 *v, const uint8_t* packet=0);
        uint8_t dmpGetControlData(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGetTemperature(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGetGravity(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGetGravity(int16_t *data, const uint8_t* packet=0);
        uint8_t dmpGetGravity(VectorInt16 *v, const uint8_t* packet=0);
        uint8_t dmpGetGravity(VectorFloat *v, Quaternion *q);
        uint8_t dmpGetUnquantizedAccel(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGetUnquantizedAccel(int16_t *data, const uint8_t* packet=0);
        uint8_t dmpGetUnquantizedAccel(VectorInt16 *v, const uint8_t* packet=0);
        uint8_t dmpGetQuantizedAccel(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGetQuantizedAccel(int16_t *data, const uint8_t* packet=0);
        uint8_t dmpGetQuantizedAccel(VectorInt16 *v, const uint8_t* packet=0);
        uint8_t dmpGetExternalSensorData(int32_t *data, uint16_t size, const uint8_t*
packet=0);
        uint8_t dmpGetEIS(int32_t *data, const uint8_t* packet=0);

        uint8_t dmpGetEuler(float *data, Quaternion *q);
        uint8_t dmpGetYawPitchRoll(float *data, Quaternion *q, VectorFloat *gravity);

        // Get Floating Point data from FIFO
        uint8_t dmpGetAccelFloat(float *data, const uint8_t* packet=0);
        uint8_t dmpGetQuaternionFloat(float *data, const uint8_t* packet=0);

        uint8_t dmpProcessFIFOPacket(const unsigned char *dmpData);
        uint8_t dmpReadAndProcessFIFOPacket(uint8_t numPackets, uint8_t
*processed=NULL);

        uint8_t dmpSetFIFOProcessedCallback(void (*func) (void));

        uint8_t dmpInitFIFOParam();
        uint8_t dmpCloseFIFO();
        uint8_t dmpSetGyroDataSource(uint8_t source);
        uint8_t dmpDecodeQuantizedAccel();
        uint32_t dmpGetGyroSumOfSquare();
        uint32_t dmpGetAccelSumOfSquare();
        void dmpOverrideQuaternion(long *q);
        uint16_t dmpGetFIFOPacketSize();
    #endif

    // special methods for MotionApps 4.1 implementation
    #ifdef MPU6050_INCLUDE_DMP_MOTIONAPPS41
        uint8_t *dmpPacketBuffer;
        uint16_t dmpPacketSize;
```

```
        uint8_t dmpInitialize();
        bool dmpPacketAvailable();

        uint8_t dmpSetFIFORate(uint8_t fifoRate);
        uint8_t dmpGetFIFORate();
        uint8_t dmpGetSampleStepSizeMS();
        uint8_t dmpGetSampleFrequency();
        int32_t dmpDecodeTemperature(int8_t tempReg);

        // Register callbacks after a packet of FIFO data is processed
        //uint8_t dmpRegisterFIFORateProcess(inv_obj_func func, int16_t priority);
        //uint8_t dmpUnregisterFIFORateProcess(inv_obj_func func);
        uint8_t dmpRunFIFORateProcesses();

        // Setup FIFO for various output
        uint8_t dmpSendQuaternion(uint_fast16_t accuracy);
        uint8_t dmpSendGyro(uint_fast16_t elements, uint_fast16_t accuracy);
        uint8_t dmpSendAccel(uint_fast16_t elements, uint_fast16_t accuracy);
        uint8_t dmpSendLinearAccel(uint_fast16_t elements, uint_fast16_t accuracy);
        uint8_t dmpSendLinearAccelInWorld(uint_fast16_t elements, uint_fast16_t
accuracy);
        uint8_t dmpSendControlData(uint_fast16_t elements, uint_fast16_t accuracy);
        uint8_t dmpSendSensorData(uint_fast16_t elements, uint_fast16_t accuracy);
        uint8_t dmpSendExternalSensorData(uint_fast16_t elements, uint_fast16_t
accuracy);
        uint8_t dmpSendGravity(uint_fast16_t elements, uint_fast16_t accuracy);
        uint8_t dmpSendPacketNumber(uint_fast16_t accuracy);
        uint8_t dmpSendQuantizedAccel(uint_fast16_t elements, uint_fast16_t
accuracy);
        uint8_t dmpSendEIS(uint_fast16_t elements, uint_fast16_t accuracy);

        // Get Fixed Point data from FIFO
        uint8_t dmpGetAccel(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGetAccel(int16_t *data, const uint8_t* packet=0);
        uint8_t dmpGetAccel(VectorInt16 *v, const uint8_t* packet=0);
        uint8_t dmpGetQuaternion(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGetQuaternion(int16_t *data, const uint8_t* packet=0);
        uint8_t dmpGetQuaternion(Quaternion *q, const uint8_t* packet=0);
        uint8_t dmpGet6AxisQuaternion(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGet6AxisQuaternion(int16_t *data, const uint8_t* packet=0);
        uint8_t dmpGet6AxisQuaternion(Quaternion *q, const uint8_t* packet=0);
        uint8_t dmpGetRelativeQuaternion(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGetRelativeQuaternion(int16_t *data, const uint8_t* packet=0);
        uint8_t dmpGetRelativeQuaternion(Quaternion *data, const uint8_t* packet=0);
        uint8_t dmpGetGyro(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGetGyro(int16_t *data, const uint8_t* packet=0);
        uint8_t dmpGetGyro(VectorInt16 *v, const uint8_t* packet=0);
        uint8_t dmpGetMag(int16_t *data, const uint8_t* packet=0);
        uint8_t dmpSetLinearAccelFilterCoefficient(float coef);
        uint8_t dmpGetLinearAccel(int32_t *data, const uint8_t* packet=0);
```

```
        uint8_t dmpGetLinearAccel(int16_t *data, const uint8_t* packet=0);
        uint8_t dmpGetLinearAccel(VectorInt16 *v, const uint8_t* packet=0);
        uint8_t dmpGetLinearAccel(VectorInt16 *v, VectorInt16 *vRaw, VectorFloat
*gravity);
        uint8_t dmpGetLinearAccelInWorld(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGetLinearAccelInWorld(int16_t *data, const uint8_t* packet=0);
        uint8_t dmpGetLinearAccelInWorld(VectorInt16 *v, const uint8_t* packet=0);
        uint8_t dmpGetLinearAccelInWorld(VectorInt16 *v, VectorInt16 *vReal,
Quaternion *q);
        uint8_t dmpGetGyroAndAccelSensor(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGetGyroAndAccelSensor(int16_t *data, const uint8_t* packet=0);
        uint8_t dmpGetGyroAndAccelSensor(VectorInt16 *g, VectorInt16 *a, const
uint8_t* packet=0);
        uint8_t dmpGetGyroSensor(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGetGyroSensor(int16_t *data, const uint8_t* packet=0);
        uint8_t dmpGetGyroSensor(VectorInt16 *v, const uint8_t* packet=0);
        uint8_t dmpGetControlData(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGetTemperature(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGetGravity(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGetGravity(int16_t *data, const uint8_t* packet=0);
        uint8_t dmpGetGravity(VectorInt16 *v, const uint8_t* packet=0);
        uint8_t dmpGetGravity(VectorFloat *v, Quaternion *q);
        uint8_t dmpGetUnquantizedAccel(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGetUnquantizedAccel(int16_t *data, const uint8_t* packet=0);
        uint8_t dmpGetUnquantizedAccel(VectorInt16 *v, const uint8_t* packet=0);
        uint8_t dmpGetQuantizedAccel(int32_t *data, const uint8_t* packet=0);
        uint8_t dmpGetQuantizedAccel(int16_t *data, const uint8_t* packet=0);
        uint8_t dmpGetQuantizedAccel(VectorInt16 *v, const uint8_t* packet=0);
        uint8_t dmpGetExternalSensorData(int32_t *data, uint16_t size, const uint8_t*
packet=0);
        uint8_t dmpGetEIS(int32_t *data, const uint8_t* packet=0);

        uint8_t dmpGetEuler(float *data, Quaternion *q);
        uint8_t dmpGetYawPitchRoll(float *data, Quaternion *q, VectorFloat *gravity);

        // Get Floating Point data from FIFO
        uint8_t dmpGetAccelFloat(float *data, const uint8_t* packet=0);
        uint8_t dmpGetQuaternionFloat(float *data, const uint8_t* packet=0);

        uint8_t dmpProcessFIFOPacket(const unsigned char *dmpData);
        uint8_t dmpReadAndProcessFIFOPacket(uint8_t numPackets, uint8_t
*processed=NULL);

        uint8_t dmpSetFIFOProcessedCallback(void (*func) (void));

        uint8_t dmpInitFIFOParam();
        uint8_t dmpCloseFIFO();
        uint8_t dmpSetGyroDataSource(uint8_t source);
        uint8_t dmpDecodeQuantizedAccel();
        uint32_t dmpGetGyroSumOfSquare();
        uint32_t dmpGetAccelSumOfSquare();
```

```
        void dmpOverrideQuaternion(long *q);
        uint16_t dmpGetFIFOPacketSize();
    #endif

  private:
    uint8_t devAddr;
    uint8_t buffer[14];
};

#endif /* _MPU6050_H_ */

//end MPU6050.h
////////////////////////////////////////////////////////////////////////////////

////////////////////////////////////////////////////////////////////////////////
//MPU6050.cpp
// I2Cdev library collection - MPU6050 I2C device class
// Based on InvenSense MPU-6050 register map document rev. 2.0, 5/19/2011 (RM-
MPU-6000A-00)
// 8/24/2011 by Jeff Rowberg <jeff@rowberg.net>
// Updates should (hopefully) always be available at https://github.com/jrowberg/i2cdevlib
//
// Changelog:
//     ... - ongoing debug release

// NOTE: THIS IS ONLY A PARIAL RELEASE. THIS DEVICE CLASS IS CURRENTLY
UNDERGOING ACTIVE
// DEVELOPMENT AND IS STILL MISSING SOME IMPORTANT FEATURES. PLEASE
KEEP THIS IN MIND IF
// YOU DECIDE TO USE THIS PARTICULAR CODE FOR ANYTHING.

/* ============================================
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.
===============================================
*/

#include "MPU6050.h"

/** Default constructor, uses default I2C address.
 * @see MPU6050_DEFAULT_ADDRESS
 */
MPU6050::MPU6050() {
    devAddr = MPU6050_DEFAULT_ADDRESS;
}

/** Specific address constructor.
 * @param address I2C address
 * @see MPU6050_DEFAULT_ADDRESS
 * @see MPU6050_ADDRESS_AD0_LOW
 * @see MPU6050_ADDRESS_AD0_HIGH
 */
MPU6050::MPU6050(uint8_t address) {
    devAddr = address;
}

/** Power on and prepare for general usage.
 * This will activate the device and take it out of sleep mode (which must be done
 * after start-up). This function also sets both the accelerometer and the gyroscope
 * to their most sensitive settings, namely +/- 2g and +/- 250 degrees/sec, and sets
 * the clock source to use the X Gyro for reference, which is slightly better than
 * the default internal clock source.
 */
void MPU6050::initialize() {
    setClockSource(MPU6050_CLOCK_PLL_XGYRO);
    setFullScaleGyroRange(MPU6050_GYRO_FS_250);
    setFullScaleAccelRange(MPU6050_ACCEL_FS_2);
    setSleepEnabled(false); // thanks to Jack Elston for pointing this one out!
}

/** Verify the I2C connection.
 * Make sure the device is connected and responds as expected.
 * @return True if connection is valid, false otherwise
 */
bool MPU6050::testConnection() {
    return getDeviceID() == 0x34;
}
```

```
// AUX_VDDIO register (InvenSense demo code calls this RA_*G_OFFS_TC)

/** Get the auxiliary I2C supply voltage level.
 * When set to 1, the auxiliary I2C bus high logic level is VDD. When cleared to
 * 0, the auxiliary I2C bus high logic level is VLOGIC. This does not apply to
 * the MPU-6000, which does not have a VLOGIC pin.
 * @return I2C supply voltage level (0=VLOGIC, 1=VDD)
 */
uint8_t MPU6050::getAuxVDDIOLevel() {
    I2Cdev::readBit(devAddr, MPU6050_RA_YG_OFFS_TC,
MPU6050_TC_PWR_MODE_BIT, buffer);
    return buffer[0];
}
/** Set the auxiliary I2C supply voltage level.
 * When set to 1, the auxiliary I2C bus high logic level is VDD. When cleared to
 * 0, the auxiliary I2C bus high logic level is VLOGIC. This does not apply to
 * the MPU-6000, which does not have a VLOGIC pin.
 * @param level I2C supply voltage level (0=VLOGIC, 1=VDD)
 */
void MPU6050::setAuxVDDIOLevel(uint8_t level) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_YG_OFFS_TC,
MPU6050_TC_PWR_MODE_BIT, level);
}

// SMPLRT_DIV register

/** Get gyroscope output rate divider.
 * The sensor register output, FIFO output, DMP sampling, Motion detection, Zero
 * Motion detection, and Free Fall detection are all based on the Sample Rate.
 * The Sample Rate is generated by dividing the gyroscope output rate by
 * SMPLRT_DIV:
 *
 * Sample Rate = Gyroscope Output Rate / (1 + SMPLRT_DIV)
 *
 * where Gyroscope Output Rate = 8kHz when the DLPF is disabled (DLPF_CFG = 0 or
 * 7), and 1kHz when the DLPF is enabled (see Register 26).
 *
 * Note: The accelerometer output rate is 1kHz. This means that for a Sample
 * Rate greater than 1kHz, the same accelerometer sample may be output to the
 * FIFO, DMP, and sensor registers more than once.
 *
 * For a diagram of the gyroscope and accelerometer signal paths, see Section 8
 * of the MPU-6000/MPU-6050 Product Specification document.
 *
 * @return Current sample rate
 * @see MPU6050_RA_SMPLRT_DIV
 */
uint8_t MPU6050::getRate() {
    I2Cdev::readByte(devAddr, MPU6050_RA_SMPLRT_DIV, buffer);
    return buffer[0];
```

```
}
/** Set gyroscope sample rate divider.
 * @param rate New sample rate divider
 * @see getRate()
 * @see MPU6050_RA_SMPLRT_DIV
 */
void MPU6050::setRate(uint8_t rate) {
    I2Cdev::writeByte(devAddr, MPU6050_RA_SMPLRT_DIV, rate);
}

// CONFIG register

/** Get external FSYNC configuration.
 * Configures the external Frame Synchronization (FSYNC) pin sampling. An
 * external signal connected to the FSYNC pin can be sampled by configuring
 * EXT_SYNC_SET. Signal changes to the FSYNC pin are latched so that short
 * strobes may be captured. The latched FSYNC signal will be sampled at the
 * Sampling Rate, as defined in register 25. After sampling, the latch will
 * reset to the current FSYNC signal state.
 *
 * The sampled value will be reported in place of the least significant bit in
 * a sensor data register determined by the value of EXT_SYNC_SET according to
 * the following table.
 *
 * <pre>
 * EXT_SYNC_SET | FSYNC Bit Location
 * -------------+-------------------
 * 0            | Input disabled
 * 1            | TEMP_OUT_L[0]
 * 2            | GYRO_XOUT_L[0]
 * 3            | GYRO_YOUT_L[0]
 * 4            | GYRO_ZOUT_L[0]
 * 5            | ACCEL_XOUT_L[0]
 * 6            | ACCEL_YOUT_L[0]
 * 7            | ACCEL_ZOUT_L[0]
 * </pre>
 *
 * @return FSYNC configuration value
 */
uint8_t MPU6050::getExternalFrameSync() {
    I2Cdev::readBits(devAddr, MPU6050_RA_CONFIG,
MPU6050_CFG_EXT_SYNC_SET_BIT, MPU6050_CFG_EXT_SYNC_SET_LENGTH,
buffer);
    return buffer[0];
}
/** Set external FSYNC configuration.
 * @see getExternalFrameSync()
 * @see MPU6050_RA_CONFIG
 * @param sync New FSYNC configuration value
 */
void MPU6050::setExternalFrameSync(uint8_t sync) {
```

```
    I2Cdev::writeBits(devAddr, MPU6050_RA_CONFIG,
MPU6050_CFG_EXT_SYNC_SET_BIT, MPU6050_CFG_EXT_SYNC_SET_LENGTH,
sync);
}
/** Get digital low-pass filter configuration.
 * The DLPF_CFG parameter sets the digital low pass filter configuration. It
 * also determines the internal sampling rate used by the device as shown in
 * the table below.
 *
 * Note: The accelerometer output rate is 1kHz. This means that for a Sample
 * Rate greater than 1kHz, the same accelerometer sample may be output to the
 * FIFO, DMP, and sensor registers more than once.
 *
 * <pre>
 *          |   ACCELEROMETER    |           GYROSCOPE
 * DLPF_CFG | Bandwidth | Delay  | Bandwidth | Delay  | Sample Rate
 * ---------+-----------+--------+-----------+--------+-------------
 * 0        | 260Hz     | 0ms    | 256Hz     | 0.98ms | 8kHz
 * 1        | 184Hz     | 2.0ms  | 188Hz     | 1.9ms  | 1kHz
 * 2        | 94Hz      | 3.0ms  | 98Hz      | 2.8ms  | 1kHz
 * 3        | 44Hz      | 4.9ms  | 42Hz      | 4.8ms  | 1kHz
 * 4        | 21Hz      | 8.5ms  | 20Hz      | 8.3ms  | 1kHz
 * 5        | 10Hz      | 13.8ms | 10Hz      | 13.4ms | 1kHz
 * 6        | 5Hz       | 19.0ms | 5Hz       | 18.6ms | 1kHz
 * 7        |   -- Reserved --   |   -- Reserved --   | Reserved
 * </pre>
 *
 * @return DLFP configuration
 * @see MPU6050_RA_CONFIG
 * @see MPU6050_CFG_DLPF_CFG_BIT
 * @see MPU6050_CFG_DLPF_CFG_LENGTH
 */
uint8_t MPU6050::getDLPFMode() {
    I2Cdev::readBits(devAddr, MPU6050_RA_CONFIG,
MPU6050_CFG_DLPF_CFG_BIT, MPU6050_CFG_DLPF_CFG_LENGTH, buffer);
    return buffer[0];
}
/** Set digital low-pass filter configuration.
 * @param mode New DLFP configuration setting
 * @see getDLPFBandwidth()
 * @see MPU6050_DLPF_BW_256
 * @see MPU6050_RA_CONFIG
 * @see MPU6050_CFG_DLPF_CFG_BIT
 * @see MPU6050_CFG_DLPF_CFG_LENGTH
 */
void MPU6050::setDLPFMode(uint8_t mode) {
    I2Cdev::writeBits(devAddr, MPU6050_RA_CONFIG,
MPU6050_CFG_DLPF_CFG_BIT, MPU6050_CFG_DLPF_CFG_LENGTH, mode);
}

// GYRO_CONFIG register
```

```
/** Get full-scale gyroscope range.
 * The FS_SEL parameter allows setting the full-scale range of the gyro sensors,
 * as described in the table below.
 *
 * <pre>
 * 0 = +/- 250 degrees/sec
 * 1 = +/- 500 degrees/sec
 * 2 = +/- 1000 degrees/sec
 * 3 = +/- 2000 degrees/sec
 * </pre>
 *
 * @return Current full-scale gyroscope range setting
 * @see MPU6050_GYRO_FS_250
 * @see MPU6050_RA_GYRO_CONFIG
 * @see MPU6050_GCONFIG_FS_SEL_BIT
 * @see MPU6050_GCONFIG_FS_SEL_LENGTH
 */
uint8_t MPU6050::getFullScaleGyroRange() {
    I2Cdev::readBits(devAddr, MPU6050_RA_GYRO_CONFIG,
MPU6050_GCONFIG_FS_SEL_BIT, MPU6050_GCONFIG_FS_SEL_LENGTH, buffer);
    return buffer[0];
}
/** Set full-scale gyroscope range.
 * @param range New full-scale gyroscope range value
 * @see getFullScaleRange()
 * @see MPU6050_GYRO_FS_250
 * @see MPU6050_RA_GYRO_CONFIG
 * @see MPU6050_GCONFIG_FS_SEL_BIT
 * @see MPU6050_GCONFIG_FS_SEL_LENGTH
 */
void MPU6050::setFullScaleGyroRange(uint8_t range) {
    I2Cdev::writeBits(devAddr, MPU6050_RA_GYRO_CONFIG,
MPU6050_GCONFIG_FS_SEL_BIT, MPU6050_GCONFIG_FS_SEL_LENGTH, range);
}

// ACCEL_CONFIG register

/** Get self-test enabled setting for accelerometer X axis.
 * @return Self-test enabled value
 * @see MPU6050_RA_ACCEL_CONFIG
 */
bool MPU6050::getAccelXSelfTest() {
    I2Cdev::readBit(devAddr, MPU6050_RA_ACCEL_CONFIG,
MPU6050_ACONFIG_XA_ST_BIT, buffer);
    return buffer[0];
}
/** Get self-test enabled setting for accelerometer X axis.
 * @param enabled Self-test enabled value
 * @see MPU6050_RA_ACCEL_CONFIG
 */
```

```
void MPU6050::setAccelXSelfTest(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_ACCEL_CONFIG,
MPU6050_ACONFIG_XA_ST_BIT, enabled);
}
/** Get self-test enabled value for accelerometer Y axis.
 * @return Self-test enabled value
 * @see MPU6050_RA_ACCEL_CONFIG
 */
bool MPU6050::getAccelYSelfTest() {
    I2Cdev::readBit(devAddr, MPU6050_RA_ACCEL_CONFIG,
MPU6050_ACONFIG_YA_ST_BIT, buffer);
    return buffer[0];
}
/** Get self-test enabled value for accelerometer Y axis.
 * @param enabled Self-test enabled value
 * @see MPU6050_RA_ACCEL_CONFIG
 */
void MPU6050::setAccelYSelfTest(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_ACCEL_CONFIG,
MPU6050_ACONFIG_YA_ST_BIT, enabled);
}
/** Get self-test enabled value for accelerometer Z axis.
 * @return Self-test enabled value
 * @see MPU6050_RA_ACCEL_CONFIG
 */
bool MPU6050::getAccelZSelfTest() {
    I2Cdev::readBit(devAddr, MPU6050_RA_ACCEL_CONFIG,
MPU6050_ACONFIG_ZA_ST_BIT, buffer);
    return buffer[0];
}
/** Set self-test enabled value for accelerometer Z axis.
 * @param enabled Self-test enabled value
 * @see MPU6050_RA_ACCEL_CONFIG
 */
void MPU6050::setAccelZSelfTest(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_ACCEL_CONFIG,
MPU6050_ACONFIG_ZA_ST_BIT, enabled);
}
/** Get full-scale accelerometer range.
 * The FS_SEL parameter allows setting the full-scale range of the accelerometer
 * sensors, as described in the table below.
 *
 * <pre>
 * 0 = +/- 2g
 * 1 = +/- 4g
 * 2 = +/- 8g
 * 3 = +/- 16g
 * </pre>
 *
 * @return Current full-scale accelerometer range setting
 * @see MPU6050_ACCEL_FS_2
```

```
 * @see MPU6050_RA_ACCEL_CONFIG
 * @see MPU6050_ACONFIG_AFS_SEL_BIT
 * @see MPU6050_ACONFIG_AFS_SEL_LENGTH
 */
uint8_t MPU6050::getFullScaleAccelRange() {
    I2Cdev::readBits(devAddr, MPU6050_RA_ACCEL_CONFIG,
MPU6050_ACONFIG_AFS_SEL_BIT, MPU6050_ACONFIG_AFS_SEL_LENGTH,
buffer);
    return buffer[0];
}
/** Set full-scale accelerometer range.
 * @param range New full-scale accelerometer range setting
 * @see getFullScaleAccelRange()
 */
void MPU6050::setFullScaleAccelRange(uint8_t range) {
    I2Cdev::writeBits(devAddr, MPU6050_RA_ACCEL_CONFIG,
MPU6050_ACONFIG_AFS_SEL_BIT, MPU6050_ACONFIG_AFS_SEL_LENGTH,
range);
}
/** Get the high-pass filter configuration.
 * The DHPF is a filter module in the path leading to motion detectors (Free
 * Fall, Motion threshold, and Zero Motion). The high pass filter output is not
 * available to the data registers (see Figure in Section 8 of the MPU-6000/
 * MPU-6050 Product Specification document).
 *
 * The high pass filter has three modes:
 *
 * <pre>
 *    Reset: The filter output settles to zero within one sample. This
 *           effectively disables the high pass filter. This mode may be toggled
 *           to quickly settle the filter.
 *
 *    On:    The high pass filter will pass signals above the cut off frequency.
 *
 *    Hold:  When triggered, the filter holds the present sample. The filter
 *           output will be the difference between the input sample and the held
 *           sample.
 * </pre>
 *
 * <pre>
 * ACCEL_HPF | Filter Mode | Cut-off Frequency
 * ----------+-------------+------------------
 * 0         | Reset       | None
 * 1         | On          | 5Hz
 * 2         | On          | 2.5Hz
 * 3         | On          | 1.25Hz
 * 4         | On          | 0.63Hz
 * 7         | Hold        | None
 * </pre>
 *
 * @return Current high-pass filter configuration
```

```
 * @see MPU6050_DHPF_RESET
 * @see MPU6050_RA_ACCEL_CONFIG
 */
uint8_t MPU6050::getDHPFMode() {
    I2Cdev::readBits(devAddr, MPU6050_RA_ACCEL_CONFIG,
MPU6050_ACONFIG_ACCEL_HPF_BIT,
MPU6050_ACONFIG_ACCEL_HPF_LENGTH, buffer);
    return buffer[0];
}
/** Set the high-pass filter configuration.
 * @param bandwidth New high-pass filter configuration
 * @see setDHPFMode()
 * @see MPU6050_DHPF_RESET
 * @see MPU6050_RA_ACCEL_CONFIG
 */
void MPU6050::setDHPFMode(uint8_t bandwidth) {
    I2Cdev::writeBits(devAddr, MPU6050_RA_ACCEL_CONFIG,
MPU6050_ACONFIG_ACCEL_HPF_BIT,
MPU6050_ACONFIG_ACCEL_HPF_LENGTH, bandwidth);
}

// FF_THR register

/** Get free-fall event acceleration threshold.
 * This register configures the detection threshold for Free Fall event
 * detection. The unit of FF_THR is 1LSB = 2mg. Free Fall is detected when the
 * absolute value of the accelerometer measurements for the three axes are each
 * less than the detection threshold. This condition increments the Free Fall
 * duration counter (Register 30). The Free Fall interrupt is triggered when the
 * Free Fall duration counter reaches the time specified in FF_DUR.
 *
 * For more details on the Free Fall detection interrupt, see Section 8.2 of the
 * MPU-6000/MPU-6050 Product Specification document as well as Registers 56 and
 * 58 of this document.
 *
 * @return Current free-fall acceleration threshold value (LSB = 2mg)
 * @see MPU6050_RA_FF_THR
 */
uint8_t MPU6050::getFreefallDetectionThreshold() {
    I2Cdev::readByte(devAddr, MPU6050_RA_FF_THR, buffer);
    return buffer[0];
}
/** Get free-fall event acceleration threshold.
 * @param threshold New free-fall acceleration threshold value (LSB = 2mg)
 * @see getFreefallDetectionThreshold()
 * @see MPU6050_RA_FF_THR
 */
void MPU6050::setFreefallDetectionThreshold(uint8_t threshold) {
    I2Cdev::writeByte(devAddr, MPU6050_RA_FF_THR, threshold);
}
```

```
// FF_DUR register

/** Get free-fall event duration threshold.
 * This register configures the duration counter threshold for Free Fall event
 * detection. The duration counter ticks at 1kHz, therefore FF_DUR has a unit
 * of 1 LSB = 1 ms.
 *
 * The Free Fall duration counter increments while the absolute value of the
 * accelerometer measurements are each less than the detection threshold
 * (Register 29). The Free Fall interrupt is triggered when the Free Fall
 * duration counter reaches the time specified in this register.
 *
 * For more details on the Free Fall detection interrupt, see Section 8.2 of
 * the MPU-6000/MPU-6050 Product Specification document as well as Registers 56
 * and 58 of this document.
 *
 * @return Current free-fall duration threshold value (LSB = 1ms)
 * @see MPU6050_RA_FF_DUR
 */
uint8_t MPU6050::getFreefallDetectionDuration() {
    I2Cdev::readByte(devAddr, MPU6050_RA_FF_DUR, buffer);
    return buffer[0];
}
/** Get free-fall event duration threshold.
 * @param duration New free-fall duration threshold value (LSB = 1ms)
 * @see getFreefallDetectionDuration()
 * @see MPU6050_RA_FF_DUR
 */
void MPU6050::setFreefallDetectionDuration(uint8_t duration) {
    I2Cdev::writeByte(devAddr, MPU6050_RA_FF_DUR, duration);
}

// MOT_THR register

/** Get motion detection event acceleration threshold.
 * This register configures the detection threshold for Motion interrupt
 * generation. The unit of MOT_THR is 1LSB = 2mg. Motion is detected when the
 * absolute value of any of the accelerometer measurements exceeds this Motion
 * detection threshold. This condition increments the Motion detection duration
 * counter (Register 32). The Motion detection interrupt is triggered when the
 * Motion Detection counter reaches the time count specified in MOT_DUR
 * (Register 32).
 *
 * The Motion interrupt will indicate the axis and polarity of detected motion
 * in MOT_DETECT_STATUS (Register 97).
 *
 * For more details on the Motion detection interrupt, see Section 8.3 of the
 * MPU-6000/MPU-6050 Product Specification document as well as Registers 56 and
 * 58 of this document.
 *
 * @return Current motion detection acceleration threshold value (LSB = 2mg)
```

```
 * @see MPU6050_RA_MOT_THR
 */
uint8_t MPU6050::getMotionDetectionThreshold() {
    I2Cdev::readByte(devAddr, MPU6050_RA_MOT_THR, buffer);
    return buffer[0];
}
/** Set free-fall event acceleration threshold.
 * @param threshold New motion detection acceleration threshold value (LSB = 2mg)
 * @see getMotionDetectionThreshold()
 * @see MPU6050_RA_MOT_THR
 */
void MPU6050::setMotionDetectionThreshold(uint8_t threshold) {
    I2Cdev::writeByte(devAddr, MPU6050_RA_MOT_THR, threshold);
}

// MOT_DUR register

/** Get motion detection event duration threshold.
 * This register configures the duration counter threshold for Motion interrupt
 * generation. The duration counter ticks at 1 kHz, therefore MOT_DUR has a unit
 * of 1LSB = 1ms. The Motion detection duration counter increments when the
 * absolute value of any of the accelerometer measurements exceeds the Motion
 * detection threshold (Register 31). The Motion detection interrupt is
 * triggered when the Motion detection counter reaches the time count specified
 * in this register.
 *
 * For more details on the Motion detection interrupt, see Section 8.3 of the
 * MPU-6000/MPU-6050 Product Specification document.
 *
 * @return Current motion detection duration threshold value (LSB = 1ms)
 * @see MPU6050_RA_MOT_DUR
 */
uint8_t MPU6050::getMotionDetectionDuration() {
    I2Cdev::readByte(devAddr, MPU6050_RA_MOT_DUR, buffer);
    return buffer[0];
}
/** Set motion detection event duration threshold.
 * @param duration New motion detection duration threshold value (LSB = 1ms)
 * @see getMotionDetectionDuration()
 * @see MPU6050_RA_MOT_DUR
 */
void MPU6050::setMotionDetectionDuration(uint8_t duration) {
    I2Cdev::writeByte(devAddr, MPU6050_RA_MOT_DUR, duration);
}

// ZRMOT_THR register

/** Get zero motion detection event acceleration threshold.
 * This register configures the detection threshold for Zero Motion interrupt
 * generation. The unit of ZRMOT_THR is 1LSB = 2mg. Zero Motion is detected when
 * the absolute value of the accelerometer measurements for the 3 axes are each
```

```
 * less than the detection threshold. This condition increments the Zero Motion
 * duration counter (Register 34). The Zero Motion interrupt is triggered when
 * the Zero Motion duration counter reaches the time count specified in
 * ZRMOT_DUR (Register 34).
 *
 * Unlike Free Fall or Motion detection, Zero Motion detection triggers an
 * interrupt both when Zero Motion is first detected and when Zero Motion is no
 * longer detected.
 *
 * When a zero motion event is detected, a Zero Motion Status will be indicated
 * in the MOT_DETECT_STATUS register (Register 97). When a motion-to-zero-motion
 * condition is detected, the status bit is set to 1. When a zero-motion-to-
 * motion condition is detected, the status bit is set to 0.
 *
 * For more details on the Zero Motion detection interrupt, see Section 8.4 of
 * the MPU-6000/MPU-6050 Product Specification document as well as Registers 56
 * and 58 of this document.
 *
 * @return Current zero motion detection acceleration threshold value (LSB = 2mg)
 * @see MPU6050_RA_ZRMOT_THR
 */
uint8_t MPU6050::getZeroMotionDetectionThreshold() {
    I2Cdev::readByte(devAddr, MPU6050_RA_ZRMOT_THR, buffer);
    return buffer[0];
}
/** Set zero motion detection event acceleration threshold.
 * @param threshold New zero motion detection acceleration threshold value (LSB =
2mg)
 * @see getZeroMotionDetectionThreshold()
 * @see MPU6050_RA_ZRMOT_THR
 */
void MPU6050::setZeroMotionDetectionThreshold(uint8_t threshold) {
    I2Cdev::writeByte(devAddr, MPU6050_RA_ZRMOT_THR, threshold);
}

// ZRMOT_DUR register

/** Get zero motion detection event duration threshold.
 * This register configures the duration counter threshold for Zero Motion
 * interrupt generation. The duration counter ticks at 16 Hz, therefore
 * ZRMOT_DUR has a unit of 1 LSB = 64 ms. The Zero Motion duration counter
 * increments while the absolute value of the accelerometer measurements are
 * each less than the detection threshold (Register 33). The Zero Motion
 * interrupt is triggered when the Zero Motion duration counter reaches the time
 * count specified in this register.
 *
 * For more details on the Zero Motion detection interrupt, see Section 8.4 of
 * the MPU-6000/MPU-6050 Product Specification document, as well as Registers 56
 * and 58 of this document.
 *
 * @return Current zero motion detection duration threshold value (LSB = 64ms)
```

```
 * @see MPU6050_RA_ZRMOT_DUR
 */
uint8_t MPU6050::getZeroMotionDetectionDuration() {
    I2Cdev::readByte(devAddr, MPU6050_RA_ZRMOT_DUR, buffer);
    return buffer[0];
}
/** Set zero motion detection event duration threshold.
 * @param duration New zero motion detection duration threshold value (LSB = 1ms)
 * @see getZeroMotionDetectionDuration()
 * @see MPU6050_RA_ZRMOT_DUR
 */
void MPU6050::setZeroMotionDetectionDuration(uint8_t duration) {
    I2Cdev::writeByte(devAddr, MPU6050_RA_ZRMOT_DUR, duration);
}

// FIFO_EN register

/** Get temperature FIFO enabled value.
 * When set to 1, this bit enables TEMP_OUT_H and TEMP_OUT_L (Registers 65 and
 * 66) to be written into the FIFO buffer.
 * @return Current temperature FIFO enabled value
 * @see MPU6050_RA_FIFO_EN
 */
bool MPU6050::getTempFIFOEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN,
MPU6050_TEMP_FIFO_EN_BIT, buffer);
    return buffer[0];
}
/** Set temperature FIFO enabled value.
 * @param enabled New temperature FIFO enabled value
 * @see getTempFIFOEnabled()
 * @see MPU6050_RA_FIFO_EN
 */
void MPU6050::setTempFIFOEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN,
MPU6050_TEMP_FIFO_EN_BIT, enabled);
}
/** Get gyroscope X-axis FIFO enabled value.
 * When set to 1, this bit enables GYRO_XOUT_H and GYRO_XOUT_L (Registers 67
and
 * 68) to be written into the FIFO buffer.
 * @return Current gyroscope X-axis FIFO enabled value
 * @see MPU6050_RA_FIFO_EN
 */
bool MPU6050::getXGyroFIFOEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_XG_FIFO_EN_BIT,
buffer);
    return buffer[0];
}
/** Set gyroscope X-axis FIFO enabled value.
 * @param enabled New gyroscope X-axis FIFO enabled value
```

```
 * @see getXGyroFIFOEnabled()
 * @see MPU6050_RA_FIFO_EN
 */
void MPU6050::setXGyroFIFOEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_XG_FIFO_EN_BIT,
enabled);
}
/** Get gyroscope Y-axis FIFO enabled value.
 * When set to 1, this bit enables GYRO_YOUT_H and GYRO_YOUT_L (Registers 69
and
 * 70) to be written into the FIFO buffer.
 * @return Current gyroscope Y-axis FIFO enabled value
 * @see MPU6050_RA_FIFO_EN
 */
bool MPU6050::getYGyroFIFOEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_YG_FIFO_EN_BIT,
buffer);
    return buffer[0];
}
/** Set gyroscope Y-axis FIFO enabled value.
 * @param enabled New gyroscope Y-axis FIFO enabled value
 * @see getYGyroFIFOEnabled()
 * @see MPU6050_RA_FIFO_EN
 */
void MPU6050::setYGyroFIFOEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_YG_FIFO_EN_BIT,
enabled);
}
/** Get gyroscope Z-axis FIFO enabled value.
 * When set to 1, this bit enables GYRO_ZOUT_H and GYRO_ZOUT_L (Registers 71
and
 * 72) to be written into the FIFO buffer.
 * @return Current gyroscope Z-axis FIFO enabled value
 * @see MPU6050_RA_FIFO_EN
 */
bool MPU6050::getZGyroFIFOEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_ZG_FIFO_EN_BIT,
buffer);
    return buffer[0];
}
/** Set gyroscope Z-axis FIFO enabled value.
 * @param enabled New gyroscope Z-axis FIFO enabled value
 * @see getZGyroFIFOEnabled()
 * @see MPU6050_RA_FIFO_EN
 */
void MPU6050::setZGyroFIFOEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_ZG_FIFO_EN_BIT,
enabled);
}
/** Get accelerometer FIFO enabled value.
 * When set to 1, this bit enables ACCEL_XOUT_H, ACCEL_XOUT_L,
```

```
ACCEL_YOUT_H,
 * ACCEL_YOUT_L, ACCEL_ZOUT_H, and ACCEL_ZOUT_L (Registers 59 to 64) to be
 * written into the FIFO buffer.
 * @return Current accelerometer FIFO enabled value
 * @see MPU6050_RA_FIFO_EN
 */
bool MPU6050::getAccelFIFOEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN,
MPU6050_ACCEL_FIFO_EN_BIT, buffer);
    return buffer[0];
}
/** Set accelerometer FIFO enabled value.
 * @param enabled New accelerometer FIFO enabled value
 * @see getAccelFIFOEnabled()
 * @see MPU6050_RA_FIFO_EN
 */
void MPU6050::setAccelFIFOEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN,
MPU6050_ACCEL_FIFO_EN_BIT, enabled);
}
/** Get Slave 2 FIFO enabled value.
 * When set to 1, this bit enables EXT_SENS_DATA registers (Registers 73 to 96)
 * associated with Slave 2 to be written into the FIFO buffer.
 * @return Current Slave 2 FIFO enabled value
 * @see MPU6050_RA_FIFO_EN
 */
bool MPU6050::getSlave2FIFOEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_SLV2_FIFO_EN_BIT,
buffer);
    return buffer[0];
}
/** Set Slave 2 FIFO enabled value.
 * @param enabled New Slave 2 FIFO enabled value
 * @see getSlave2FIFOEnabled()
 * @see MPU6050_RA_FIFO_EN
 */
void MPU6050::setSlave2FIFOEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_SLV2_FIFO_EN_BIT,
enabled);
}
/** Get Slave 1 FIFO enabled value.
 * When set to 1, this bit enables EXT_SENS_DATA registers (Registers 73 to 96)
 * associated with Slave 1 to be written into the FIFO buffer.
 * @return Current Slave 1 FIFO enabled value
 * @see MPU6050_RA_FIFO_EN
 */
bool MPU6050::getSlave1FIFOEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_SLV1_FIFO_EN_BIT,
buffer);
    return buffer[0];
}
```

```
/** Set Slave 1 FIFO enabled value.
 * @param enabled New Slave 1 FIFO enabled value
 * @see getSlave1FIFOEnabled()
 * @see MPU6050_RA_FIFO_EN
 */
void MPU6050::setSlave1FIFOEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_SLV1_FIFO_EN_BIT,
enabled);
}
/** Get Slave 0 FIFO enabled value.
 * When set to 1, this bit enables EXT_SENS_DATA registers (Registers 73 to 96)
 * associated with Slave 0 to be written into the FIFO buffer.
 * @return Current Slave 0 FIFO enabled value
 * @see MPU6050_RA_FIFO_EN
 */
bool MPU6050::getSlave0FIFOEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_SLV0_FIFO_EN_BIT,
buffer);
    return buffer[0];
}
/** Set Slave 0 FIFO enabled value.
 * @param enabled New Slave 0 FIFO enabled value
 * @see getSlave0FIFOEnabled()
 * @see MPU6050_RA_FIFO_EN
 */
void MPU6050::setSlave0FIFOEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_FIFO_EN, MPU6050_SLV0_FIFO_EN_BIT,
enabled);
}

// I2C_MST_CTRL register

/** Get multi-master enabled value.
 * Multi-master capability allows multiple I2C masters to operate on the same
 * bus. In circuits where multi-master capability is required, set MULT_MST_EN
 * to 1. This will increase current drawn by approximately 30uA.
 *
 * In circuits where multi-master capability is required, the state of the I2C
 * bus must always be monitored by each separate I2C Master. Before an I2C
 * Master can assume arbitration of the bus, it must first confirm that no other
 * I2C Master has arbitration of the bus. When MULT_MST_EN is set to 1, the
 * MPU-60X0's bus arbitration detection logic is turned on, enabling it to
 * detect when the bus is available.
 *
 * @return Current multi-master enabled value
 * @see MPU6050_RA_I2C_MST_CTRL
 */
bool MPU6050::getMultiMasterEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_CTRL,
MPU6050_MULT_MST_EN_BIT, buffer);
    return buffer[0];
```

```
}
/** Set multi-master enabled value.
 * @param enabled New multi-master enabled value
 * @see getMultiMasterEnabled()
 * @see MPU6050_RA_I2C_MST_CTRL
 */
void MPU6050::setMultiMasterEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_MST_CTRL,
MPU6050_MULT_MST_EN_BIT, enabled);
}
/** Get wait-for-external-sensor-data enabled value.
 * When the WAIT_FOR_ES bit is set to 1, the Data Ready interrupt will be
 * delayed until External Sensor data from the Slave Devices are loaded into the
 * EXT_SENS_DATA registers. This is used to ensure that both the internal sensor
 * data (i.e. from gyro and accel) and external sensor data have been loaded to
 * their respective data registers (i.e. the data is synced) when the Data Ready
 * interrupt is triggered.
 *
 * @return Current wait-for-external-sensor-data enabled value
 * @see MPU6050_RA_I2C_MST_CTRL
 */
bool MPU6050::getWaitForExternalSensorEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_CTRL,
MPU6050_WAIT_FOR_ES_BIT, buffer);
    return buffer[0];
}
/** Set wait-for-external-sensor-data enabled value.
 * @param enabled New wait-for-external-sensor-data enabled value
 * @see getWaitForExternalSensorEnabled()
 * @see MPU6050_RA_I2C_MST_CTRL
 */
void MPU6050::setWaitForExternalSensorEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_MST_CTRL,
MPU6050_WAIT_FOR_ES_BIT, enabled);
}
/** Get Slave 3 FIFO enabled value.
 * When set to 1, this bit enables EXT_SENS_DATA registers (Registers 73 to 96)
 * associated with Slave 3 to be written into the FIFO buffer.
 * @return Current Slave 3 FIFO enabled value
 * @see MPU6050_RA_MST_CTRL
 */
bool MPU6050::getSlave3FIFOEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_CTRL,
MPU6050_SLV_3_FIFO_EN_BIT, buffer);
    return buffer[0];
}
/** Set Slave 3 FIFO enabled value.
 * @param enabled New Slave 3 FIFO enabled value
 * @see getSlave3FIFOEnabled()
 * @see MPU6050_RA_MST_CTRL
 */
```

```
void MPU6050::setSlave3FIFOEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_MST_CTRL,
MPU6050_SLV_3_FIFO_EN_BIT, enabled);
}
/** Get slave read/write transition enabled value.
 * The I2C_MST_P_NSR bit configures the I2C Master's transition from one slave
 * read to the next slave read. If the bit equals 0, there will be a restart
 * between reads. If the bit equals 1, there will be a stop followed by a start
 * of the following read. When a write transaction follows a read transaction,
 * the stop followed by a start of the successive write will be always used.
 *
 * @return Current slave read/write transition enabled value
 * @see MPU6050_RA_I2C_MST_CTRL
 */
bool MPU6050::getSlaveReadWriteTransitionEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_CTRL,
MPU6050_I2C_MST_P_NSR_BIT, buffer);
    return buffer[0];
}
/** Set slave read/write transition enabled value.
 * @param enabled New slave read/write transition enabled value
 * @see getSlaveReadWriteTransitionEnabled()
 * @see MPU6050_RA_I2C_MST_CTRL
 */
void MPU6050::setSlaveReadWriteTransitionEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_MST_CTRL,
MPU6050_I2C_MST_P_NSR_BIT, enabled);
}
/** Get I2C master clock speed.
 * I2C_MST_CLK is a 4 bit unsigned value which configures a divider on the
 * MPU-60X0 internal 8MHz clock. It sets the I2C master clock speed according to
 * the following table:
 *
 * <pre>
 * I2C_MST_CLK | I2C Master Clock Speed | 8MHz Clock Divider
 * ------------+------------------------+-------------------
 * 0           | 348kHz                 | 23
 * 1           | 333kHz                 | 24
 * 2           | 320kHz                 | 25
 * 3           | 308kHz                 | 26
 * 4           | 296kHz                 | 27
 * 5           | 286kHz                 | 28
 * 6           | 276kHz                 | 29
 * 7           | 267kHz                 | 30
 * 8           | 258kHz                 | 31
 * 9           | 500kHz                 | 16
 * 10          | 471kHz                 | 17
 * 11          | 444kHz                 | 18
 * 12          | 421kHz                 | 19
 * 13          | 400kHz                 | 20
 * 14          | 381kHz                 | 21
```

```
 * 15         | 364kHz              | 22
 * </pre>
 *
 * @return Current I2C master clock speed
 * @see MPU6050_RA_I2C_MST_CTRL
 */
uint8_t MPU6050::getMasterClockSpeed() {
    I2Cdev::readBits(devAddr, MPU6050_RA_I2C_MST_CTRL,
MPU6050_I2C_MST_CLK_BIT, MPU6050_I2C_MST_CLK_LENGTH, buffer);
    return buffer[0];
}
/** Set I2C master clock speed.
 * @reparam speed Current I2C master clock speed
 * @see MPU6050_RA_I2C_MST_CTRL
 */
void MPU6050::setMasterClockSpeed(uint8_t speed) {
    I2Cdev::writeBits(devAddr, MPU6050_RA_I2C_MST_CTRL,
MPU6050_I2C_MST_CLK_BIT, MPU6050_I2C_MST_CLK_LENGTH, speed);
}

// I2C_SLV* registers (Slave 0-3)

/** Get the I2C address of the specified slave (0-3).
 * Note that Bit 7 (MSB) controls read/write mode. If Bit 7 is set, it's a read
 * operation, and if it is cleared, then it's a write operation. The remaining
 * bits (6-0) are the 7-bit device address of the slave device.
 *
 * In read mode, the result of the read is placed in the lowest available
 * EXT_SENS_DATA register. For further information regarding the allocation of
 * read results, please refer to the EXT_SENS_DATA register description
 * (Registers 73 - 96).
 *
 * The MPU-6050 supports a total of five slaves, but Slave 4 has unique
 * characteristics, and so it has its own functions (getSlave4* and setSlave4*).
 *
 * I2C data transactions are performed at the Sample Rate, as defined in
 * Register 25. The user is responsible for ensuring that I2C data transactions
 * to and from each enabled Slave can be completed within a single period of the
 * Sample Rate.
 *
 * The I2C slave access rate can be reduced relative to the Sample Rate. This
 * reduced access rate is determined by I2C_MST_DLY (Register 52). Whether a
 * slave's access rate is reduced relative to the Sample Rate is determined by
 * I2C_MST_DELAY_CTRL (Register 103).
 *
 * The processing order for the slaves is fixed. The sequence followed for
 * processing the slaves is Slave 0, Slave 1, Slave 2, Slave 3 and Slave 4. If a
 * particular Slave is disabled it will be skipped.
 *
 * Each slave can either be accessed at the sample rate or at a reduced sample
 * rate. In a case where some slaves are accessed at the Sample Rate and some
```

```
 * slaves are accessed at the reduced rate, the sequence of accessing the slaves
 * (Slave 0 to Slave 4) is still followed. However, the reduced rate slaves will
 * be skipped if their access rate dictates that they should not be accessed
 * during that particular cycle. For further information regarding the reduced
 * access rate, please refer to Register 52. Whether a slave is accessed at the
 * Sample Rate or at the reduced rate is determined by the Delay Enable bits in
 * Register 103.
 *
 * @param num Slave number (0-3)
 * @return Current address for specified slave
 * @see MPU6050_RA_I2C_SLV0_ADDR
 */
uint8_t MPU6050::getSlaveAddress(uint8_t num) {
    if (num > 3) return 0;
    I2Cdev::readByte(devAddr, MPU6050_RA_I2C_SLV0_ADDR + num*3, buffer);
    return buffer[0];
}
/** Set the I2C address of the specified slave (0-3).
 * @param num Slave number (0-3)
 * @param address New address for specified slave
 * @see getSlaveAddress()
 * @see MPU6050_RA_I2C_SLV0_ADDR
 */
void MPU6050::setSlaveAddress(uint8_t num, uint8_t address) {
    if (num > 3) return;
    I2Cdev::writeByte(devAddr, MPU6050_RA_I2C_SLV0_ADDR + num*3, address);
}
/** Get the active internal register for the specified slave (0-3).
 * Read/write operations for this slave will be done to whatever internal
 * register address is stored in this MPU register.
 *
 * The MPU-6050 supports a total of five slaves, but Slave 4 has unique
 * characteristics, and so it has its own functions.
 *
 * @param num Slave number (0-3)
 * @return Current active register for specified slave
 * @see MPU6050_RA_I2C_SLV0_REG
 */
uint8_t MPU6050::getSlaveRegister(uint8_t num) {
    if (num > 3) return 0;
    I2Cdev::readByte(devAddr, MPU6050_RA_I2C_SLV0_REG + num*3, buffer);
    return buffer[0];
}
/** Set the active internal register for the specified slave (0-3).
 * @param num Slave number (0-3)
 * @param reg New active register for specified slave
 * @see getSlaveRegister()
 * @see MPU6050_RA_I2C_SLV0_REG
 */
void MPU6050::setSlaveRegister(uint8_t num, uint8_t reg) {
    if (num > 3) return;
```

```
    I2Cdev::writeByte(devAddr, MPU6050_RA_I2C_SLV0_REG + num*3, reg);
}
/** Get the enabled value for the specified slave (0-3).
 * When set to 1, this bit enables Slave 0 for data transfer operations. When
 * cleared to 0, this bit disables Slave 0 from data transfer operations.
 * @param num Slave number (0-3)
 * @return Current enabled value for specified slave
 * @see MPU6050_RA_I2C_SLV0_CTRL
 */
bool MPU6050::getSlaveEnabled(uint8_t num) {
    if (num > 3) return 0;
    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3,
MPU6050_I2C_SLV_EN_BIT, buffer);
    return buffer[0];
}
/** Set the enabled value for the specified slave (0-3).
 * @param num Slave number (0-3)
 * @param enabled New enabled value for specified slave
 * @see getSlaveEnabled()
 * @see MPU6050_RA_I2C_SLV0_CTRL
 */
void MPU6050::setSlaveEnabled(uint8_t num, bool enabled) {
    if (num > 3) return;
    I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3,
MPU6050_I2C_SLV_EN_BIT, enabled);
}
/** Get word pair byte-swapping enabled for the specified slave (0-3).
 * When set to 1, this bit enables byte swapping. When byte swapping is enabled,
 * the high and low bytes of a word pair are swapped. Please refer to
 * I2C_SLV0_GRP for the pairing convention of the word pairs. When cleared to 0,
 * bytes transferred to and from Slave 0 will be written to EXT_SENS_DATA
 * registers in the order they were transferred.
 *
 * @param num Slave number (0-3)
 * @return Current word pair byte-swapping enabled value for specified slave
 * @see MPU6050_RA_I2C_SLV0_CTRL
 */
bool MPU6050::getSlaveWordByteSwap(uint8_t num) {
    if (num > 3) return 0;
    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3,
MPU6050_I2C_SLV_BYTE_SW_BIT, buffer);
    return buffer[0];
}
/** Set word pair byte-swapping enabled for the specified slave (0-3).
 * @param num Slave number (0-3)
 * @param enabled New word pair byte-swapping enabled value for specified slave
 * @see getSlaveWordByteSwap()
 * @see MPU6050_RA_I2C_SLV0_CTRL
 */
void MPU6050::setSlaveWordByteSwap(uint8_t num, bool enabled) {
    if (num > 3) return;
```

```
    I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3,
MPU6050_I2C_SLV_BYTE_SW_BIT, enabled);
}
/** Get write mode for the specified slave (0-3).
 * When set to 1, the transaction will read or write data only. When cleared to
 * 0, the transaction will write a register address prior to reading or writing
 * data. This should equal 0 when specifying the register address within the
 * Slave device to/from which the ensuing data transaction will take place.
 *
 * @param num Slave number (0-3)
 * @return Current write mode for specified slave (0 = register address + data, 1 = data
only)
 * @see MPU6050_RA_I2C_SLV0_CTRL
 */
bool MPU6050::getSlaveWriteMode(uint8_t num) {
    if (num > 3) return 0;
    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3,
MPU6050_I2C_SLV_REG_DIS_BIT, buffer);
    return buffer[0];
}
/** Set write mode for the specified slave (0-3).
 * @param num Slave number (0-3)
 * @param mode New write mode for specified slave (0 = register address + data, 1 =
data only)
 * @see getSlaveWriteMode()
 * @see MPU6050_RA_I2C_SLV0_CTRL
 */
void MPU6050::setSlaveWriteMode(uint8_t num, bool mode) {
    if (num > 3) return;
    I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3,
MPU6050_I2C_SLV_REG_DIS_BIT, mode);
}
/** Get word pair grouping order offset for the specified slave (0-3).
 * This sets specifies the grouping order of word pairs received from registers.
 * When cleared to 0, bytes from register addresses 0 and 1, 2 and 3, etc (even,
 * then odd register addresses) are paired to form a word. When set to 1, bytes
 * from register addresses are paired 1 and 2, 3 and 4, etc. (odd, then even
 * register addresses) are paired to form a word.
 *
 * @param num Slave number (0-3)
 * @return Current word pair grouping order offset for specified slave
 * @see MPU6050_RA_I2C_SLV0_CTRL
 */
bool MPU6050::getSlaveWordGroupOffset(uint8_t num) {
    if (num > 3) return 0;
    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3,
MPU6050_I2C_SLV_GRP_BIT, buffer);
    return buffer[0];
}
/** Set word pair grouping order offset for the specified slave (0-3).
 * @param num Slave number (0-3)
```

```
 * @param enabled New word pair grouping order offset for specified slave
 * @see getSlaveWordGroupOffset()
 * @see MPU6050_RA_I2C_SLV0_CTRL
 */
void MPU6050::setSlaveWordGroupOffset(uint8_t num, bool enabled) {
    if (num > 3) return;
    I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3,
MPU6050_I2C_SLV_GRP_BIT, enabled);
}
/** Get number of bytes to read for the specified slave (0-3).
 * Specifies the number of bytes transferred to and from Slave 0. Clearing this
 * bit to 0 is equivalent to disabling the register by writing 0 to I2C_SLV0_EN.
 * @param num Slave number (0-3)
 * @return Number of bytes to read for specified slave
 * @see MPU6050_RA_I2C_SLV0_CTRL
 */
uint8_t MPU6050::getSlaveDataLength(uint8_t num) {
    if (num > 3) return 0;
    I2Cdev::readBits(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3,
MPU6050_I2C_SLV_LEN_BIT, MPU6050_I2C_SLV_LEN_LENGTH, buffer);
    return buffer[0];
}
/** Set number of bytes to read for the specified slave (0-3).
 * @param num Slave number (0-3)
 * @param length Number of bytes to read for specified slave
 * @see getSlaveDataLength()
 * @see MPU6050_RA_I2C_SLV0_CTRL
 */
void MPU6050::setSlaveDataLength(uint8_t num, uint8_t length) {
    if (num > 3) return;
    I2Cdev::writeBits(devAddr, MPU6050_RA_I2C_SLV0_CTRL + num*3,
MPU6050_I2C_SLV_LEN_BIT, MPU6050_I2C_SLV_LEN_LENGTH, length);
}

// I2C_SLV* registers (Slave 4)

/** Get the I2C address of Slave 4.
 * Note that Bit 7 (MSB) controls read/write mode. If Bit 7 is set, it's a read
 * operation, and if it is cleared, then it's a write operation. The remaining
 * bits (6-0) are the 7-bit device address of the slave device.
 *
 * @return Current address for Slave 4
 * @see getSlaveAddress()
 * @see MPU6050_RA_I2C_SLV4_ADDR
 */
uint8_t MPU6050::getSlave4Address() {
    I2Cdev::readByte(devAddr, MPU6050_RA_I2C_SLV4_ADDR, buffer);
    return buffer[0];
}
/** Set the I2C address of Slave 4.
 * @param address New address for Slave 4
```

```
 * @see getSlave4Address()
 * @see MPU6050_RA_I2C_SLV4_ADDR
 */
void MPU6050::setSlave4Address(uint8_t address) {
    I2Cdev::writeByte(devAddr, MPU6050_RA_I2C_SLV4_ADDR, address);
}
/** Get the active internal register for the Slave 4.
 * Read/write operations for this slave will be done to whatever internal
 * register address is stored in this MPU register.
 *
 * @return Current active register for Slave 4
 * @see MPU6050_RA_I2C_SLV4_REG
 */
uint8_t MPU6050::getSlave4Register() {
    I2Cdev::readByte(devAddr, MPU6050_RA_I2C_SLV4_REG, buffer);
    return buffer[0];
}
/** Set the active internal register for Slave 4.
 * @param reg New active register for Slave 4
 * @see getSlave4Register()
 * @see MPU6050_RA_I2C_SLV4_REG
 */
void MPU6050::setSlave4Register(uint8_t reg) {
    I2Cdev::writeByte(devAddr, MPU6050_RA_I2C_SLV4_REG, reg);
}
/** Set new byte to write to Slave 4.
 * This register stores the data to be written into the Slave 4. If I2C_SLV4_RW
 * is set 1 (set to read), this register has no effect.
 * @param data New byte to write to Slave 4
 * @see MPU6050_RA_I2C_SLV4_DO
 */
void MPU6050::setSlave4OutputByte(uint8_t data) {
    I2Cdev::writeByte(devAddr, MPU6050_RA_I2C_SLV4_DO, data);
}
/** Get the enabled value for the Slave 4.
 * When set to 1, this bit enables Slave 4 for data transfer operations. When
 * cleared to 0, this bit disables Slave 4 from data transfer operations.
 * @return Current enabled value for Slave 4
 * @see MPU6050_RA_I2C_SLV4_CTRL
 */
bool MPU6050::getSlave4Enabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_SLV4_CTRL,
MPU6050_I2C_SLV4_EN_BIT, buffer);
    return buffer[0];
}
/** Set the enabled value for Slave 4.
 * @param enabled New enabled value for Slave 4
 * @see getSlave4Enabled()
 * @see MPU6050_RA_I2C_SLV4_CTRL
 */
void MPU6050::setSlave4Enabled(bool enabled) {
```

```
    I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_SLV4_CTRL,
MPU6050_I2C_SLV4_EN_BIT, enabled);
}
/** Get the enabled value for Slave 4 transaction interrupts.
 * When set to 1, this bit enables the generation of an interrupt signal upon
 * completion of a Slave 4 transaction. When cleared to 0, this bit disables the
 * generation of an interrupt signal upon completion of a Slave 4 transaction.
 * The interrupt status can be observed in Register 54.
 *
 * @return Current enabled value for Slave 4 transaction interrupts.
 * @see MPU6050_RA_I2C_SLV4_CTRL
 */
bool MPU6050::getSlave4InterruptEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_SLV4_CTRL,
MPU6050_I2C_SLV4_INT_EN_BIT, buffer);
    return buffer[0];
}
/** Set the enabled value for Slave 4 transaction interrupts.
 * @param enabled New enabled value for Slave 4 transaction interrupts.
 * @see getSlave4InterruptEnabled()
 * @see MPU6050_RA_I2C_SLV4_CTRL
 */
void MPU6050::setSlave4InterruptEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_SLV4_CTRL,
MPU6050_I2C_SLV4_INT_EN_BIT, enabled);
}
/** Get write mode for Slave 4.
 * When set to 1, the transaction will read or write data only. When cleared to
 * 0, the transaction will write a register address prior to reading or writing
 * data. This should equal 0 when specifying the register address within the
 * Slave device to/from which the ensuing data transaction will take place.
 *
 * @return Current write mode for Slave 4 (0 = register address + data, 1 = data only)
 * @see MPU6050_RA_I2C_SLV4_CTRL
 */
bool MPU6050::getSlave4WriteMode() {
    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_SLV4_CTRL,
MPU6050_I2C_SLV4_REG_DIS_BIT, buffer);
    return buffer[0];
}
/** Set write mode for the Slave 4.
 * @param mode New write mode for Slave 4 (0 = register address + data, 1 = data
only)
 * @see getSlave4WriteMode()
 * @see MPU6050_RA_I2C_SLV4_CTRL
 */
void MPU6050::setSlave4WriteMode(bool mode) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_SLV4_CTRL,
MPU6050_I2C_SLV4_REG_DIS_BIT, mode);
}
/** Get Slave 4 master delay value.
```

```
 * This configures the reduced access rate of I2C slaves relative to the Sample
 * Rate. When a slave's access rate is decreased relative to the Sample Rate,
 * the slave is accessed every:
 *
 *     1 / (1 + I2C_MST_DLY) samples
 *
 * This base Sample Rate in turn is determined by SMPLRT_DIV (register 25) and
 * DLPF_CFG (register 26). Whether a slave's access rate is reduced relative to
 * the Sample Rate is determined by I2C_MST_DELAY_CTRL (register 103). For
 * further information regarding the Sample Rate, please refer to register 25.
 *
 * @return Current Slave 4 master delay value
 * @see MPU6050_RA_I2C_SLV4_CTRL
 */
uint8_t MPU6050::getSlave4MasterDelay() {
    I2Cdev::readBits(devAddr, MPU6050_RA_I2C_SLV4_CTRL,
MPU6050_I2C_SLV4_MST_DLY_BIT, MPU6050_I2C_SLV4_MST_DLY_LENGTH,
buffer);
    return buffer[0];
}
/** Set Slave 4 master delay value.
 * @param delay New Slave 4 master delay value
 * @see getSlave4MasterDelay()
 * @see MPU6050_RA_I2C_SLV4_CTRL
 */
void MPU6050::setSlave4MasterDelay(uint8_t delay) {
    I2Cdev::writeBits(devAddr, MPU6050_RA_I2C_SLV4_CTRL,
MPU6050_I2C_SLV4_MST_DLY_BIT, MPU6050_I2C_SLV4_MST_DLY_LENGTH,
delay);
}
/** Get last available byte read from Slave 4.
 * This register stores the data read from Slave 4. This field is populated
 * after a read transaction.
 * @return Last available byte read from to Slave 4
 * @see MPU6050_RA_I2C_SLV4_DI
 */
uint8_t MPU6050::getSlate4InputByte() {
    I2Cdev::readByte(devAddr, MPU6050_RA_I2C_SLV4_DI, buffer);
    return buffer[0];
}

// I2C_MST_STATUS register

/** Get FSYNC interrupt status.
 * This bit reflects the status of the FSYNC interrupt from an external device
 * into the MPU-60X0. This is used as a way to pass an external interrupt
 * through the MPU-60X0 to the host application processor. When set to 1, this
 * bit will cause an interrupt if FSYNC_INT_EN is asserted in INT_PIN_CFG
 * (Register 55).
 * @return FSYNC interrupt status
 * @see MPU6050_RA_I2C_MST_STATUS
```

```
 */
bool MPU6050::getPassthroughStatus() {
    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_STATUS,
MPU6050_MST_PASS_THROUGH_BIT, buffer);
    return buffer[0];
}
/** Get Slave 4 transaction done status.
 * Automatically sets to 1 when a Slave 4 transaction has completed. This
 * triggers an interrupt if the I2C_MST_INT_EN bit in the INT_ENABLE register
 * (Register 56) is asserted and if the SLV_4_DONE_INT bit is asserted in the
 * I2C_SLV4_CTRL register (Register 52).
 * @return Slave 4 transaction done status
 * @see MPU6050_RA_I2C_MST_STATUS
 */
bool MPU6050::getSlave4IsDone() {
    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_STATUS,
MPU6050_MST_I2C_SLV4_DONE_BIT, buffer);
    return buffer[0];
}
/** Get master arbitration lost status.
 * This bit automatically sets to 1 when the I2C Master has lost arbitration of
 * the auxiliary I2C bus (an error condition). This triggers an interrupt if the
 * I2C_MST_INT_EN bit in the INT_ENABLE register (Register 56) is asserted.
 * @return Master arbitration lost status
 * @see MPU6050_RA_I2C_MST_STATUS
 */
bool MPU6050::getLostArbitration() {
    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_STATUS,
MPU6050_MST_I2C_LOST_ARB_BIT, buffer);
    return buffer[0];
}
/** Get Slave 4 NACK status.
 * This bit automatically sets to 1 when the I2C Master receives a NACK in a
 * transaction with Slave 4. This triggers an interrupt if the I2C_MST_INT_EN
 * bit in the INT_ENABLE register (Register 56) is asserted.
 * @return Slave 4 NACK interrupt status
 * @see MPU6050_RA_I2C_MST_STATUS
 */
bool MPU6050::getSlave4Nack() {
    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_STATUS,
MPU6050_MST_I2C_SLV4_NACK_BIT, buffer);
    return buffer[0];
}
/** Get Slave 3 NACK status.
 * This bit automatically sets to 1 when the I2C Master receives a NACK in a
 * transaction with Slave 3. This triggers an interrupt if the I2C_MST_INT_EN
 * bit in the INT_ENABLE register (Register 56) is asserted.
 * @return Slave 3 NACK interrupt status
 * @see MPU6050_RA_I2C_MST_STATUS
 */
bool MPU6050::getSlave3Nack() {
```

```
    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_STATUS,
MPU6050_MST_I2C_SLV3_NACK_BIT, buffer);
    return buffer[0];
}
/** Get Slave 2 NACK status.
 * This bit automatically sets to 1 when the I2C Master receives a NACK in a
 * transaction with Slave 2. This triggers an interrupt if the I2C_MST_INT_EN
 * bit in the INT_ENABLE register (Register 56) is asserted.
 * @return Slave 2 NACK interrupt status
 * @see MPU6050_RA_I2C_MST_STATUS
 */
bool MPU6050::getSlave2Nack() {
    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_STATUS,
MPU6050_MST_I2C_SLV2_NACK_BIT, buffer);
    return buffer[0];
}
/** Get Slave 1 NACK status.
 * This bit automatically sets to 1 when the I2C Master receives a NACK in a
 * transaction with Slave 1. This triggers an interrupt if the I2C_MST_INT_EN
 * bit in the INT_ENABLE register (Register 56) is asserted.
 * @return Slave 1 NACK interrupt status
 * @see MPU6050_RA_I2C_MST_STATUS
 */
bool MPU6050::getSlave1Nack() {
    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_STATUS,
MPU6050_MST_I2C_SLV1_NACK_BIT, buffer);
    return buffer[0];
}
/** Get Slave 0 NACK status.
 * This bit automatically sets to 1 when the I2C Master receives a NACK in a
 * transaction with Slave 0. This triggers an interrupt if the I2C_MST_INT_EN
 * bit in the INT_ENABLE register (Register 56) is asserted.
 * @return Slave 0 NACK interrupt status
 * @see MPU6050_RA_I2C_MST_STATUS
 */
bool MPU6050::getSlave0Nack() {
    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_STATUS,
MPU6050_MST_I2C_SLV0_NACK_BIT, buffer);
    return buffer[0];
}

// INT_PIN_CFG register

/** Get interrupt logic level mode.
 * Will be set 0 for active-high, 1 for active-low.
 * @return Current interrupt mode (0=active-high, 1=active-low)
 * @see MPU6050_RA_INT_PIN_CFG
 * @see MPU6050_INTCFG_INT_LEVEL_BIT
 */
bool MPU6050::getInterruptMode() {
    I2Cdev::readBit(devAddr, MPU6050_RA_INT_PIN_CFG,
```

```
MPU6050_INTCFG_INT_LEVEL_BIT, buffer);
    return buffer[0];
}
/** Set interrupt logic level mode.
 * @param mode New interrupt mode (0=active-high, 1=active-low)
 * @see getInterruptMode()
 * @see MPU6050_RA_INT_PIN_CFG
 * @see MPU6050_INTCFG_INT_LEVEL_BIT
 */
void MPU6050::setInterruptMode(bool mode) {
   I2Cdev::writeBit(devAddr, MPU6050_RA_INT_PIN_CFG,
MPU6050_INTCFG_INT_LEVEL_BIT, mode);
}
/** Get interrupt drive mode.
 * Will be set 0 for push-pull, 1 for open-drain.
 * @return Current interrupt drive mode (0=push-pull, 1=open-drain)
 * @see MPU6050_RA_INT_PIN_CFG
 * @see MPU6050_INTCFG_INT_OPEN_BIT
 */
bool MPU6050::getInterruptDrive() {
    I2Cdev::readBit(devAddr, MPU6050_RA_INT_PIN_CFG,
MPU6050_INTCFG_INT_OPEN_BIT, buffer);
    return buffer[0];
}
/** Set interrupt drive mode.
 * @param drive New interrupt drive mode (0=push-pull, 1=open-drain)
 * @see getInterruptDrive()
 * @see MPU6050_RA_INT_PIN_CFG
 * @see MPU6050_INTCFG_INT_OPEN_BIT
 */
void MPU6050::setInterruptDrive(bool drive) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_PIN_CFG,
MPU6050_INTCFG_INT_OPEN_BIT, drive);
}
/** Get interrupt latch mode.
 * Will be set 0 for 50us-pulse, 1 for latch-until-int-cleared.
 * @return Current latch mode (0=50us-pulse, 1=latch-until-int-cleared)
 * @see MPU6050_RA_INT_PIN_CFG
 * @see MPU6050_INTCFG_LATCH_INT_EN_BIT
 */
bool MPU6050::getInterruptLatch() {
    I2Cdev::readBit(devAddr, MPU6050_RA_INT_PIN_CFG,
MPU6050_INTCFG_LATCH_INT_EN_BIT, buffer);
    return buffer[0];
}
/** Set interrupt latch mode.
 * @param latch New latch mode (0=50us-pulse, 1=latch-until-int-cleared)
 * @see getInterruptLatch()
 * @see MPU6050_RA_INT_PIN_CFG
 * @see MPU6050_INTCFG_LATCH_INT_EN_BIT
 */
```

```
void MPU6050::setInterruptLatch(bool latch) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_PIN_CFG,
MPU6050_INTCFG_LATCH_INT_EN_BIT, latch);
}
/** Get interrupt latch clear mode.
 * Will be set 0 for status-read-only, 1 for any-register-read.
 * @return Current latch clear mode (0=status-read-only, 1=any-register-read)
 * @see MPU6050_RA_INT_PIN_CFG
 * @see MPU6050_INTCFG_INT_RD_CLEAR_BIT
 */
bool MPU6050::getInterruptLatchClear() {
    I2Cdev::readBit(devAddr, MPU6050_RA_INT_PIN_CFG,
MPU6050_INTCFG_INT_RD_CLEAR_BIT, buffer);
    return buffer[0];
}
/** Set interrupt latch clear mode.
 * @param clear New latch clear mode (0=status-read-only, 1=any-register-read)
 * @see getInterruptLatchClear()
 * @see MPU6050_RA_INT_PIN_CFG
 * @see MPU6050_INTCFG_INT_RD_CLEAR_BIT
 */
void MPU6050::setInterruptLatchClear(bool clear) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_PIN_CFG,
MPU6050_INTCFG_INT_RD_CLEAR_BIT, clear);
}
/** Get FSYNC interrupt logic level mode.
 * @return Current FSYNC interrupt mode (0=active-high, 1=active-low)
 * @see getFSyncInterruptMode()
 * @see MPU6050_RA_INT_PIN_CFG
 * @see MPU6050_INTCFG_FSYNC_INT_LEVEL_BIT
 */
bool MPU6050::getFSyncInterruptLevel() {
    I2Cdev::readBit(devAddr, MPU6050_RA_INT_PIN_CFG,
MPU6050_INTCFG_FSYNC_INT_LEVEL_BIT, buffer);
    return buffer[0];
}
/** Set FSYNC interrupt logic level mode.
 * @param mode New FSYNC interrupt mode (0=active-high, 1=active-low)
 * @see getFSyncInterruptMode()
 * @see MPU6050_RA_INT_PIN_CFG
 * @see MPU6050_INTCFG_FSYNC_INT_LEVEL_BIT
 */
void MPU6050::setFSyncInterruptLevel(bool level) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_PIN_CFG,
MPU6050_INTCFG_FSYNC_INT_LEVEL_BIT, level);
}
/** Get FSYNC pin interrupt enabled setting.
 * Will be set 0 for disabled, 1 for enabled.
 * @return Current interrupt enabled setting
 * @see MPU6050_RA_INT_PIN_CFG
 * @see MPU6050_INTCFG_FSYNC_INT_EN_BIT
```

```
 */
bool MPU6050::getFSyncInterruptEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_INT_PIN_CFG,
MPU6050_INTCFG_FSYNC_INT_EN_BIT, buffer);
    return buffer[0];
}
/** Set FSYNC pin interrupt enabled setting.
 * @param enabled New FSYNC pin interrupt enabled setting
 * @see getFSyncInterruptEnabled()
 * @see MPU6050_RA_INT_PIN_CFG
 * @see MPU6050_INTCFG_FSYNC_INT_EN_BIT
 */
void MPU6050::setFSyncInterruptEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_PIN_CFG,
MPU6050_INTCFG_FSYNC_INT_EN_BIT, enabled);
}
/** Get I2C bypass enabled status.
 * When this bit is equal to 1 and I2C_MST_EN (Register 106 bit[5]) is equal to
 * 0, the host application processor will be able to directly access the
 * auxiliary I2C bus of the MPU-60X0. When this bit is equal to 0, the host
 * application processor will not be able to directly access the auxiliary I2C
 * bus of the MPU-60X0 regardless of the state of I2C_MST_EN (Register 106
 * bit[5]).
 * @return Current I2C bypass enabled status
 * @see MPU6050_RA_INT_PIN_CFG
 * @see MPU6050_INTCFG_I2C_BYPASS_EN_BIT
 */
bool MPU6050::getI2CBypassEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_INT_PIN_CFG,
MPU6050_INTCFG_I2C_BYPASS_EN_BIT, buffer);
    return buffer[0];
}
/** Set I2C bypass enabled status.
 * When this bit is equal to 1 and I2C_MST_EN (Register 106 bit[5]) is equal to
 * 0, the host application processor will be able to directly access the
 * auxiliary I2C bus of the MPU-60X0. When this bit is equal to 0, the host
 * application processor will not be able to directly access the auxiliary I2C
 * bus of the MPU-60X0 regardless of the state of I2C_MST_EN (Register 106
 * bit[5]).
 * @param enabled New I2C bypass enabled status
 * @see MPU6050_RA_INT_PIN_CFG
 * @see MPU6050_INTCFG_I2C_BYPASS_EN_BIT
 */
void MPU6050::setI2CBypassEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_PIN_CFG,
MPU6050_INTCFG_I2C_BYPASS_EN_BIT, enabled);
}
/** Get reference clock output enabled status.
 * When this bit is equal to 1, a reference clock output is provided at the
 * CLKOUT pin. When this bit is equal to 0, the clock output is disabled. For
 * further information regarding CLKOUT, please refer to the MPU-60X0 Product
```

```
 * Specification document.
 * @return Current reference clock output enabled status
 * @see MPU6050_RA_INT_PIN_CFG
 * @see MPU6050_INTCFG_CLKOUT_EN_BIT
 */
bool MPU6050::getClockOutputEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_INT_PIN_CFG,
MPU6050_INTCFG_CLKOUT_EN_BIT, buffer);
    return buffer[0];
}
/** Set reference clock output enabled status.
 * When this bit is equal to 1, a reference clock output is provided at the
 * CLKOUT pin. When this bit is equal to 0, the clock output is disabled. For
 * further information regarding CLKOUT, please refer to the MPU-60X0 Product
 * Specification document.
 * @param enabled New reference clock output enabled status
 * @see MPU6050_RA_INT_PIN_CFG
 * @see MPU6050_INTCFG_CLKOUT_EN_BIT
 */
void MPU6050::setClockOutputEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_PIN_CFG,
MPU6050_INTCFG_CLKOUT_EN_BIT, enabled);
}

// INT_ENABLE register

/** Get full interrupt enabled status.
 * Full register byte for all interrupts, for quick reading. Each bit will be
 * set 0 for disabled, 1 for enabled.
 * @return Current interrupt enabled status
 * @see MPU6050_RA_INT_ENABLE
 * @see MPU6050_INTERRUPT_FF_BIT
 **/
uint8_t MPU6050::getIntEnabled() {
    I2Cdev::readByte(devAddr, MPU6050_RA_INT_ENABLE, buffer);
    return buffer[0];
}
/** Set full interrupt enabled status.
 * Full register byte for all interrupts, for quick reading. Each bit should be
 * set 0 for disabled, 1 for enabled.
 * @param enabled New interrupt enabled status
 * @see getIntFreefallEnabled()
 * @see MPU6050_RA_INT_ENABLE
 * @see MPU6050_INTERRUPT_FF_BIT
 **/
void MPU6050::setIntEnabled(uint8_t enabled) {
    I2Cdev::writeByte(devAddr, MPU6050_RA_INT_ENABLE, enabled);
}
/** Get Free Fall interrupt enabled status.
 * Will be set 0 for disabled, 1 for enabled.
 * @return Current interrupt enabled status
```

```
 * @see MPU6050_RA_INT_ENABLE
 * @see MPU6050_INTERRUPT_FF_BIT
 **/
bool MPU6050::getIntFreefallEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_INT_ENABLE,
MPU6050_INTERRUPT_FF_BIT, buffer);
    return buffer[0];
}
/** Set Free Fall interrupt enabled status.
 * @param enabled New interrupt enabled status
 * @see getIntFreefallEnabled()
 * @see MPU6050_RA_INT_ENABLE
 * @see MPU6050_INTERRUPT_FF_BIT
 **/
void MPU6050::setIntFreefallEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_ENABLE,
MPU6050_INTERRUPT_FF_BIT, enabled);
}
/** Get Motion Detection interrupt enabled status.
 * Will be set 0 for disabled, 1 for enabled.
 * @return Current interrupt enabled status
 * @see MPU6050_RA_INT_ENABLE
 * @see MPU6050_INTERRUPT_MOT_BIT
 **/
bool MPU6050::getIntMotionEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_INT_ENABLE,
MPU6050_INTERRUPT_MOT_BIT, buffer);
    return buffer[0];
}
/** Set Motion Detection interrupt enabled status.
 * @param enabled New interrupt enabled status
 * @see getIntMotionEnabled()
 * @see MPU6050_RA_INT_ENABLE
 * @see MPU6050_INTERRUPT_MOT_BIT
 **/
void MPU6050::setIntMotionEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_ENABLE,
MPU6050_INTERRUPT_MOT_BIT, enabled);
}
/** Get Zero Motion Detection interrupt enabled status.
 * Will be set 0 for disabled, 1 for enabled.
 * @return Current interrupt enabled status
 * @see MPU6050_RA_INT_ENABLE
 * @see MPU6050_INTERRUPT_ZMOT_BIT
 **/
bool MPU6050::getIntZeroMotionEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_INT_ENABLE,
MPU6050_INTERRUPT_ZMOT_BIT, buffer);
    return buffer[0];
}
/** Set Zero Motion Detection interrupt enabled status.
```

```
 * @param enabled New interrupt enabled status
 * @see getIntZeroMotionEnabled()
 * @see MPU6050_RA_INT_ENABLE
 * @see MPU6050_INTERRUPT_ZMOT_BIT
 **/
void MPU6050::setIntZeroMotionEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_ENABLE,
MPU6050_INTERRUPT_ZMOT_BIT, enabled);
}
/** Get FIFO Buffer Overflow interrupt enabled status.
 * Will be set 0 for disabled, 1 for enabled.
 * @return Current interrupt enabled status
 * @see MPU6050_RA_INT_ENABLE
 * @see MPU6050_INTERRUPT_FIFO_OFLOW_BIT
 **/
bool MPU6050::getIntFIFOBufferOverflowEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_INT_ENABLE,
MPU6050_INTERRUPT_FIFO_OFLOW_BIT, buffer);
    return buffer[0];
}
/** Set FIFO Buffer Overflow interrupt enabled status.
 * @param enabled New interrupt enabled status
 * @see getIntFIFOBufferOverflowEnabled()
 * @see MPU6050_RA_INT_ENABLE
 * @see MPU6050_INTERRUPT_FIFO_OFLOW_BIT
 **/
void MPU6050::setIntFIFOBufferOverflowEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_ENABLE,
MPU6050_INTERRUPT_FIFO_OFLOW_BIT, enabled);
}
/** Get I2C Master interrupt enabled status.
 * This enables any of the I2C Master interrupt sources to generate an
 * interrupt. Will be set 0 for disabled, 1 for enabled.
 * @return Current interrupt enabled status
 * @see MPU6050_RA_INT_ENABLE
 * @see MPU6050_INTERRUPT_I2C_MST_INT_BIT
 **/
bool MPU6050::getIntI2CMasterEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_INT_ENABLE,
MPU6050_INTERRUPT_I2C_MST_INT_BIT, buffer);
    return buffer[0];
}
/** Set I2C Master interrupt enabled status.
 * @param enabled New interrupt enabled status
 * @see getIntI2CMasterEnabled()
 * @see MPU6050_RA_INT_ENABLE
 * @see MPU6050_INTERRUPT_I2C_MST_INT_BIT
 **/
void MPU6050::setIntI2CMasterEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_ENABLE,
MPU6050_INTERRUPT_I2C_MST_INT_BIT, enabled);
```

```
}
/** Get Data Ready interrupt enabled setting.
 * This event occurs each time a write operation to all of the sensor registers
 * has been completed. Will be set 0 for disabled, 1 for enabled.
 * @return Current interrupt enabled status
 * @see MPU6050_RA_INT_ENABLE
 * @see MPU6050_INTERRUPT_DATA_RDY_BIT
 */
bool MPU6050::getIntDataReadyEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_INT_ENABLE,
MPU6050_INTERRUPT_DATA_RDY_BIT, buffer);
    return buffer[0];
}
/** Set Data Ready interrupt enabled status.
 * @param enabled New interrupt enabled status
 * @see getIntDataReadyEnabled()
 * @see MPU6050_RA_INT_CFG
 * @see MPU6050_INTERRUPT_DATA_RDY_BIT
 */
void MPU6050::setIntDataReadyEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_ENABLE,
MPU6050_INTERRUPT_DATA_RDY_BIT, enabled);
}

// INT_STATUS register

/** Get full set of interrupt status bits.
 * These bits clear to 0 after the register has been read. Very useful
 * for getting multiple INT statuses, since each single bit read clears
 * all of them because it has to read the whole byte.
 * @return Current interrupt status
 * @see MPU6050_RA_INT_STATUS
 */
uint8_t MPU6050::getIntStatus() {
    I2Cdev::readByte(devAddr, MPU6050_RA_INT_STATUS, buffer);
    return buffer[0];
}
/** Get Free Fall interrupt status.
 * This bit automatically sets to 1 when a Free Fall interrupt has been
 * generated. The bit clears to 0 after the register has been read.
 * @return Current interrupt status
 * @see MPU6050_RA_INT_STATUS
 * @see MPU6050_INTERRUPT_FF_BIT
 */
bool MPU6050::getIntFreefallStatus() {
    I2Cdev::readBit(devAddr, MPU6050_RA_INT_STATUS,
MPU6050_INTERRUPT_FF_BIT, buffer);
    return buffer[0];
}
/** Get Motion Detection interrupt status.
 * This bit automatically sets to 1 when a Motion Detection interrupt has been
```

```
 * generated. The bit clears to 0 after the register has been read.
 * @return Current interrupt status
 * @see MPU6050_RA_INT_STATUS
 * @see MPU6050_INTERRUPT_MOT_BIT
 */
bool MPU6050::getIntMotionStatus() {
    I2Cdev::readBit(devAddr, MPU6050_RA_INT_STATUS,
MPU6050_INTERRUPT_MOT_BIT, buffer);
    return buffer[0];
}
/** Get Zero Motion Detection interrupt status.
 * This bit automatically sets to 1 when a Zero Motion Detection interrupt has
 * been generated. The bit clears to 0 after the register has been read.
 * @return Current interrupt status
 * @see MPU6050_RA_INT_STATUS
 * @see MPU6050_INTERRUPT_ZMOT_BIT
 */
bool MPU6050::getIntZeroMotionStatus() {
    I2Cdev::readBit(devAddr, MPU6050_RA_INT_STATUS,
MPU6050_INTERRUPT_ZMOT_BIT, buffer);
    return buffer[0];
}
/** Get FIFO Buffer Overflow interrupt status.
 * This bit automatically sets to 1 when a Free Fall interrupt has been
 * generated. The bit clears to 0 after the register has been read.
 * @return Current interrupt status
 * @see MPU6050_RA_INT_STATUS
 * @see MPU6050_INTERRUPT_FIFO_OFLOW_BIT
 */
bool MPU6050::getIntFIFOBufferOverflowStatus() {
    I2Cdev::readBit(devAddr, MPU6050_RA_INT_STATUS,
MPU6050_INTERRUPT_FIFO_OFLOW_BIT, buffer);
    return buffer[0];
}
/** Get I2C Master interrupt status.
 * This bit automatically sets to 1 when an I2C Master interrupt has been
 * generated. For a list of I2C Master interrupts, please refer to Register 54.
 * The bit clears to 0 after the register has been read.
 * @return Current interrupt status
 * @see MPU6050_RA_INT_STATUS
 * @see MPU6050_INTERRUPT_I2C_MST_INT_BIT
 */
bool MPU6050::getIntI2CMasterStatus() {
    I2Cdev::readBit(devAddr, MPU6050_RA_INT_STATUS,
MPU6050_INTERRUPT_I2C_MST_INT_BIT, buffer);
    return buffer[0];
}
/** Get Data Ready interrupt status.
 * This bit automatically sets to 1 when a Data Ready interrupt has been
 * generated. The bit clears to 0 after the register has been read.
 * @return Current interrupt status
```

```
 * @see MPU6050_RA_INT_STATUS
 * @see MPU6050_INTERRUPT_DATA_RDY_BIT
 */
bool MPU6050::getIntDataReadyStatus() {
    I2Cdev::readBit(devAddr, MPU6050_RA_INT_STATUS,
MPU6050_INTERRUPT_DATA_RDY_BIT, buffer);
    return buffer[0];
}

// ACCEL_*OUT_* registers

/** Get raw 9-axis motion sensor readings (accel/gyro/compass).
 * FUNCTION NOT FULLY IMPLEMENTED YET.
 * @param ax 16-bit signed integer container for accelerometer X-axis value
 * @param ay 16-bit signed integer container for accelerometer Y-axis value
 * @param az 16-bit signed integer container for accelerometer Z-axis value
 * @param gx 16-bit signed integer container for gyroscope X-axis value
 * @param gy 16-bit signed integer container for gyroscope Y-axis value
 * @param gz 16-bit signed integer container for gyroscope Z-axis value
 * @param mx 16-bit signed integer container for magnetometer X-axis value
 * @param my 16-bit signed integer container for magnetometer Y-axis value
 * @param mz 16-bit signed integer container for magnetometer Z-axis value
 * @see getMotion6()
 * @see getAcceleration()
 * @see getRotation()
 * @see MPU6050_RA_ACCEL_XOUT_H
 */
void MPU6050::getMotion9(int16_t* ax, int16_t* ay, int16_t* az, int16_t* gx, int16_t* gy,
int16_t* gz, int16_t* mx, int16_t* my, int16_t* mz) {

        //get accel and gyro
        getMotion6(ax, ay, az, gx, gy, gz);

        //read mag
        I2Cdev::writeByte(devAddr, MPU6050_RA_INT_PIN_CFG, 0x02); //set i2c bypass
enable pin to true to access magnetometer
        delay(10);
        I2Cdev::writeByte(MPU9150_RA_MAG_ADDRESS, 0x0A, 0x01); //enable the
magnetometer
        delay(10);
        I2Cdev::readBytes(MPU9150_RA_MAG_ADDRESS,
MPU9150_RA_MAG_XOUT_L, 6, buffer);
        *mx = (((int16_t)buffer[0]) << 8) | buffer[1];
    *my = (((int16_t)buffer[2]) << 8) | buffer[3];
    *mz = (((int16_t)buffer[4]) << 8) | buffer[5];
}
/** Get raw 6-axis motion sensor readings (accel/gyro).
 * Retrieves all currently available motion sensor values.
 * @param ax 16-bit signed integer container for accelerometer X-axis value
 * @param ay 16-bit signed integer container for accelerometer Y-axis value
 * @param az 16-bit signed integer container for accelerometer Z-axis value
```

```
 * @param gx 16-bit signed integer container for gyroscope X-axis value
 * @param gy 16-bit signed integer container for gyroscope Y-axis value
 * @param gz 16-bit signed integer container for gyroscope Z-axis value
 * @see getAcceleration()
 * @see getRotation()
 * @see MPU6050_RA_ACCEL_XOUT_H
 */
void MPU6050::getMotion6(int16_t* ax, int16_t* ay, int16_t* az, int16_t* gx, int16_t* gy,
int16_t* gz) {
    I2Cdev::readBytes(devAddr, MPU6050_RA_ACCEL_XOUT_H, 14, buffer);
    *ax = (((int16_t)buffer[0]) << 8) | buffer[1];
    *ay = (((int16_t)buffer[2]) << 8) | buffer[3];
    *az = (((int16_t)buffer[4]) << 8) | buffer[5];
    *gx = (((int16_t)buffer[8]) << 8) | buffer[9];
    *gy = (((int16_t)buffer[10]) << 8) | buffer[11];
    *gz = (((int16_t)buffer[12]) << 8) | buffer[13];
}
/** Get 3-axis accelerometer readings.
 * These registers store the most recent accelerometer measurements.
 * Accelerometer measurements are written to these registers at the Sample Rate
 * as defined in Register 25.
 *
 * The accelerometer measurement registers, along with the temperature
 * measurement registers, gyroscope measurement registers, and external sensor
 * data registers, are composed of two sets of registers: an internal register
 * set and a user-facing read register set.
 *
 * The data within the accelerometer sensors' internal register set is always
 * updated at the Sample Rate. Meanwhile, the user-facing read register set
 * duplicates the internal register set's data values whenever the serial
 * interface is idle. This guarantees that a burst read of sensor registers will
 * read measurements from the same sampling instant. Note that if burst reads
 * are not used, the user is responsible for ensuring a set of single byte reads
 * correspond to a single sampling instant by checking the Data Ready interrupt.
 *
 * Each 16-bit accelerometer measurement has a full scale defined in ACCEL_FS
 * (Register 28). For each full scale setting, the accelerometers' sensitivity
 * per LSB in ACCEL_xOUT is shown in the table below:
 *
 * <pre>
 * AFS_SEL | Full Scale Range | LSB Sensitivity
 * --------+------------------+----------------
 * 0       | +/- 2g           | 8192 LSB/mg
 * 1       | +/- 4g           | 4096 LSB/mg
 * 2       | +/- 8g           | 2048 LSB/mg
 * 3       | +/- 16g          | 1024 LSB/mg
 * </pre>
 *
 * @param x 16-bit signed integer container for X-axis acceleration
 * @param y 16-bit signed integer container for Y-axis acceleration
 * @param z 16-bit signed integer container for Z-axis acceleration
```

```
 * @see MPU6050_RA_GYRO_XOUT_H
 */
void MPU6050::getAcceleration(int16_t* x, int16_t* y, int16_t* z) {
    I2Cdev::readBytes(devAddr, MPU6050_RA_ACCEL_XOUT_H, 6, buffer);
    *x = (((int16_t)buffer[0]) << 8) | buffer[1];
    *y = (((int16_t)buffer[2]) << 8) | buffer[3];
    *z = (((int16_t)buffer[4]) << 8) | buffer[5];
}
/** Get X-axis accelerometer reading.
 * @return X-axis acceleration measurement in 16-bit 2's complement format
 * @see getMotion6()
 * @see MPU6050_RA_ACCEL_XOUT_H
 */
int16_t MPU6050::getAccelerationX() {
    I2Cdev::readBytes(devAddr, MPU6050_RA_ACCEL_XOUT_H, 2, buffer);
    return (((int16_t)buffer[0]) << 8) | buffer[1];
}
/** Get Y-axis accelerometer reading.
 * @return Y-axis acceleration measurement in 16-bit 2's complement format
 * @see getMotion6()
 * @see MPU6050_RA_ACCEL_YOUT_H
 */
int16_t MPU6050::getAccelerationY() {
    I2Cdev::readBytes(devAddr, MPU6050_RA_ACCEL_YOUT_H, 2, buffer);
    return (((int16_t)buffer[0]) << 8) | buffer[1];
}
/** Get Z-axis accelerometer reading.
 * @return Z-axis acceleration measurement in 16-bit 2's complement format
 * @see getMotion6()
 * @see MPU6050_RA_ACCEL_ZOUT_H
 */
int16_t MPU6050::getAccelerationZ() {
    I2Cdev::readBytes(devAddr, MPU6050_RA_ACCEL_ZOUT_H, 2, buffer);
    return (((int16_t)buffer[0]) << 8) | buffer[1];
}

// TEMP_OUT_* registers

/** Get current internal temperature.
 * @return Temperature reading in 16-bit 2's complement format
 * @see MPU6050_RA_TEMP_OUT_H
 */
int16_t MPU6050::getTemperature() {
    I2Cdev::readBytes(devAddr, MPU6050_RA_TEMP_OUT_H, 2, buffer);
    return (((int16_t)buffer[0]) << 8) | buffer[1];
}

// GYRO_*OUT_* registers

/** Get 3-axis gyroscope readings.
 * These gyroscope measurement registers, along with the accelerometer
```

```
 * measurement registers, temperature measurement registers, and external sensor
 * data registers, are composed of two sets of registers: an internal register
 * set and a user-facing read register set.
 * The data within the gyroscope sensors' internal register set is always
 * updated at the Sample Rate. Meanwhile, the user-facing read register set
 * duplicates the internal register set's data values whenever the serial
 * interface is idle. This guarantees that a burst read of sensor registers will
 * read measurements from the same sampling instant. Note that if burst reads
 * are not used, the user is responsible for ensuring a set of single byte reads
 * correspond to a single sampling instant by checking the Data Ready interrupt.
 *
 * Each 16-bit gyroscope measurement has a full scale defined in FS_SEL
 * (Register 27). For each full scale setting, the gyroscopes' sensitivity per
 * LSB in GYRO_xOUT is shown in the table below:
 *
 * <pre>
 * FS_SEL | Full Scale Range   | LSB Sensitivity
 * -------+--------------------+----------------
 * 0      | +/- 250 degrees/s  | 131 LSB/deg/s
 * 1      | +/- 500 degrees/s  | 65.5 LSB/deg/s
 * 2      | +/- 1000 degrees/s | 32.8 LSB/deg/s
 * 3      | +/- 2000 degrees/s | 16.4 LSB/deg/s
 * </pre>
 *
 * @param x 16-bit signed integer container for X-axis rotation
 * @param y 16-bit signed integer container for Y-axis rotation
 * @param z 16-bit signed integer container for Z-axis rotation
 * @see getMotion6()
 * @see MPU6050_RA_GYRO_XOUT_H
 */
void MPU6050::getRotation(int16_t* x, int16_t* y, int16_t* z) {
    I2Cdev::readBytes(devAddr, MPU6050_RA_GYRO_XOUT_H, 6, buffer);
    *x = (((int16_t)buffer[0]) << 8) | buffer[1];
    *y = (((int16_t)buffer[2]) << 8) | buffer[3];
    *z = (((int16_t)buffer[4]) << 8) | buffer[5];
}
/** Get X-axis gyroscope reading.
 * @return X-axis rotation measurement in 16-bit 2's complement format
 * @see getMotion6()
 * @see MPU6050_RA_GYRO_XOUT_H
 */
int16_t MPU6050::getRotationX() {
    I2Cdev::readBytes(devAddr, MPU6050_RA_GYRO_XOUT_H, 2, buffer);
    return (((int16_t)buffer[0]) << 8) | buffer[1];
}
/** Get Y-axis gyroscope reading.
 * @return Y-axis rotation measurement in 16-bit 2's complement format
 * @see getMotion6()
 * @see MPU6050_RA_GYRO_YOUT_H
 */
int16_t MPU6050::getRotationY() {
```

```
    I2Cdev::readBytes(devAddr, MPU6050_RA_GYRO_YOUT_H, 2, buffer);
    return (((int16_t)buffer[0]) << 8) | buffer[1];
}
/** Get Z-axis gyroscope reading.
 * @return Z-axis rotation measurement in 16-bit 2's complement format
 * @see getMotion6()
 * @see MPU6050_RA_GYRO_ZOUT_H
 */
int16_t MPU6050::getRotationZ() {
    I2Cdev::readBytes(devAddr, MPU6050_RA_GYRO_ZOUT_H, 2, buffer);
    return (((int16_t)buffer[0]) << 8) | buffer[1];
}

// EXT_SENS_DATA_* registers

/** Read single byte from external sensor data register.
 * These registers store data read from external sensors by the Slave 0, 1, 2,
 * and 3 on the auxiliary I2C interface. Data read by Slave 4 is stored in
 * I2C_SLV4_DI (Register 53).
 *
 * External sensor data is written to these registers at the Sample Rate as
 * defined in Register 25. This access rate can be reduced by using the Slave
 * Delay Enable registers (Register 103).
 *
 * External sensor data registers, along with the gyroscope measurement
 * registers, accelerometer measurement registers, and temperature measurement
 * registers, are composed of two sets of registers: an internal register set
 * and a user-facing read register set.
 *
 * The data within the external sensors' internal register set is always updated
 * at the Sample Rate (or the reduced access rate) whenever the serial interface
 * is idle. This guarantees that a burst read of sensor registers will read
 * measurements from the same sampling instant. Note that if burst reads are not
 * used, the user is responsible for ensuring a set of single byte reads
 * correspond to a single sampling instant by checking the Data Ready interrupt.
 *
 * Data is placed in these external sensor data registers according to
 * I2C_SLV0_CTRL, I2C_SLV1_CTRL, I2C_SLV2_CTRL, and I2C_SLV3_CTRL
(Registers 39,
 * 42, 45, and 48). When more than zero bytes are read (I2C_SLVx_LEN > 0) from
 * an enabled slave (I2C_SLVx_EN = 1), the slave is read at the Sample Rate (as
 * defined in Register 25) or delayed rate (if specified in Register 52 and
 * 103). During each Sample cycle, slave reads are performed in order of Slave
 * number. If all slaves are enabled with more than zero bytes to be read, the
 * order will be Slave 0, followed by Slave 1, Slave 2, and Slave 3.
 *
 * Each enabled slave will have EXT_SENS_DATA registers associated with it by
 * number of bytes read (I2C_SLVx_LEN) in order of slave number, starting from
 * EXT_SENS_DATA_00. Note that this means enabling or disabling a slave may
 * change the higher numbered slaves' associated registers. Furthermore, if
 * fewer total bytes are being read from the external sensors as a result of
```

```
 * such a change, then the data remaining in the registers which no longer have
 * an associated slave device (i.e. high numbered registers) will remain in
 * these previously allocated registers unless reset.
 *
 * If the sum of the read lengths of all SLVx transactions exceed the number of
 * available EXT_SENS_DATA registers, the excess bytes will be dropped. There
 * are 24 EXT_SENS_DATA registers and hence the total read lengths between all
 * the slaves cannot be greater than 24 or some bytes will be lost.
 *
 * Note: Slave 4's behavior is distinct from that of Slaves 0-3. For further
 * information regarding the characteristics of Slave 4, please refer to
 * Registers 49 to 53.
 *
 * EXAMPLE:
 * Suppose that Slave 0 is enabled with 4 bytes to be read (I2C_SLV0_EN = 1 and
 * I2C_SLV0_LEN = 4) while Slave 1 is enabled with 2 bytes to be read so that
 * I2C_SLV1_EN = 1 and I2C_SLV1_LEN = 2. In such a situation, EXT_SENS_DATA
_00
 * through _03 will be associated with Slave 0, while EXT_SENS_DATA _04 and 05
 * will be associated with Slave 1. If Slave 2 is enabled as well, registers
 * starting from EXT_SENS_DATA_06 will be allocated to Slave 2.
 *
 * If Slave 2 is disabled while Slave 3 is enabled in this same situation, then
 * registers starting from EXT_SENS_DATA_06 will be allocated to Slave 3
 * instead.
 *
 * REGISTER ALLOCATION FOR DYNAMIC DISABLE VS. NORMAL DISABLE:
 * If a slave is disabled at any time, the space initially allocated to the
 * slave in the EXT_SENS_DATA register, will remain associated with that slave.
 * This is to avoid dynamic adjustment of the register allocation.
 *
 * The allocation of the EXT_SENS_DATA registers is recomputed only when (1) all
 * slaves are disabled, or (2) the I2C_MST_RST bit is set (Register 106).
 *
 * This above is also true if one of the slaves gets NACKed and stops
 * functioning.
 *
 * @param position Starting position (0-23)
 * @return Byte read from register
 */
uint8_t MPU6050::getExternalSensorByte(int position) {
    I2Cdev::readByte(devAddr, MPU6050_RA_EXT_SENS_DATA_00 + position, buffer);
    return buffer[0];
}
/** Read word (2 bytes) from external sensor data registers.
 * @param position Starting position (0-21)
 * @return Word read from register
 * @see getExternalSensorByte()
 */
uint16_t MPU6050::getExternalSensorWord(int position) {
    I2Cdev::readBytes(devAddr, MPU6050_RA_EXT_SENS_DATA_00 + position, 2,
```

```
buffer);
    return (((uint16_t)buffer[0]) << 8) | buffer[1];
}
/** Read double word (4 bytes) from external sensor data registers.
 * @param position Starting position (0-20)
 * @return Double word read from registers
 * @see getExternalSensorByte()
 */
uint32_t MPU6050::getExternalSensorDWord(int position) {
    I2Cdev::readBytes(devAddr, MPU6050_RA_EXT_SENS_DATA_00 + position, 4,
buffer);
    return (((uint32_t)buffer[0]) << 24) | (((uint32_t)buffer[1]) << 16) | (((uint16_t)buffer[2])
<< 8) | buffer[3];
}

// MOT_DETECT_STATUS register

/** Get X-axis negative motion detection interrupt status.
 * @return Motion detection status
 * @see MPU6050_RA_MOT_DETECT_STATUS
 * @see MPU6050_MOTION_MOT_XNEG_BIT
 */
bool MPU6050::getXNegMotionDetected() {
    I2Cdev::readBit(devAddr, MPU6050_RA_MOT_DETECT_STATUS,
MPU6050_MOTION_MOT_XNEG_BIT, buffer);
    return buffer[0];
}
/** Get X-axis positive motion detection interrupt status.
 * @return Motion detection status
 * @see MPU6050_RA_MOT_DETECT_STATUS
 * @see MPU6050_MOTION_MOT_XPOS_BIT
 */
bool MPU6050::getXPosMotionDetected() {
    I2Cdev::readBit(devAddr, MPU6050_RA_MOT_DETECT_STATUS,
MPU6050_MOTION_MOT_XPOS_BIT, buffer);
    return buffer[0];
}
/** Get Y-axis negative motion detection interrupt status.
 * @return Motion detection status
 * @see MPU6050_RA_MOT_DETECT_STATUS
 * @see MPU6050_MOTION_MOT_YNEG_BIT
 */
bool MPU6050::getYNegMotionDetected() {
    I2Cdev::readBit(devAddr, MPU6050_RA_MOT_DETECT_STATUS,
MPU6050_MOTION_MOT_YNEG_BIT, buffer);
    return buffer[0];
}
/** Get Y-axis positive motion detection interrupt status.
 * @return Motion detection status
 * @see MPU6050_RA_MOT_DETECT_STATUS
 * @see MPU6050_MOTION_MOT_YPOS_BIT
```

```
 */
bool MPU6050::getYPosMotionDetected() {
    I2Cdev::readBit(devAddr, MPU6050_RA_MOT_DETECT_STATUS,
MPU6050_MOTION_MOT_YPOS_BIT, buffer);
    return buffer[0];
}
/** Get Z-axis negative motion detection interrupt status.
 * @return Motion detection status
 * @see MPU6050_RA_MOT_DETECT_STATUS
 * @see MPU6050_MOTION_MOT_ZNEG_BIT
 */
bool MPU6050::getZNegMotionDetected() {
    I2Cdev::readBit(devAddr, MPU6050_RA_MOT_DETECT_STATUS,
MPU6050_MOTION_MOT_ZNEG_BIT, buffer);
    return buffer[0];
}
/** Get Z-axis positive motion detection interrupt status.
 * @return Motion detection status
 * @see MPU6050_RA_MOT_DETECT_STATUS
 * @see MPU6050_MOTION_MOT_ZPOS_BIT
 */
bool MPU6050::getZPosMotionDetected() {
    I2Cdev::readBit(devAddr, MPU6050_RA_MOT_DETECT_STATUS,
MPU6050_MOTION_MOT_ZPOS_BIT, buffer);
    return buffer[0];
}
/** Get zero motion detection interrupt status.
 * @return Motion detection status
 * @see MPU6050_RA_MOT_DETECT_STATUS
 * @see MPU6050_MOTION_MOT_ZRMOT_BIT
 */
bool MPU6050::getZeroMotionDetected() {
    I2Cdev::readBit(devAddr, MPU6050_RA_MOT_DETECT_STATUS,
MPU6050_MOTION_MOT_ZRMOT_BIT, buffer);
    return buffer[0];
}

// I2C_SLV*_DO register

/** Write byte to Data Output container for specified slave.
 * This register holds the output data written into Slave when Slave is set to
 * write mode. For further information regarding Slave control, please
 * refer to Registers 37 to 39 and immediately following.
 * @param num Slave number (0-3)
 * @param data Byte to write
 * @see MPU6050_RA_I2C_SLV0_DO
 */
void MPU6050::setSlaveOutputByte(uint8_t num, uint8_t data) {
    if (num > 3) return;
    I2Cdev::writeByte(devAddr, MPU6050_RA_I2C_SLV0_DO + num, data);
}
```

```
// I2C_MST_DELAY_CTRL register

/** Get external data shadow delay enabled status.
 * This register is used to specify the timing of external sensor data
 * shadowing. When DELAY_ES_SHADOW is set to 1, shadowing of external
 * sensor data is delayed until all data has been received.
 * @return Current external data shadow delay enabled status.
 * @see MPU6050_RA_I2C_MST_DELAY_CTRL
 * @see MPU6050_DELAYCTRL_DELAY_ES_SHADOW_BIT
 */
bool MPU6050::getExternalShadowDelayEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_DELAY_CTRL,
MPU6050_DELAYCTRL_DELAY_ES_SHADOW_BIT, buffer);
    return buffer[0];
}
/** Set external data shadow delay enabled status.
 * @param enabled New external data shadow delay enabled status.
 * @see getExternalShadowDelayEnabled()
 * @see MPU6050_RA_I2C_MST_DELAY_CTRL
 * @see MPU6050_DELAYCTRL_DELAY_ES_SHADOW_BIT
 */
void MPU6050::setExternalShadowDelayEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_MST_DELAY_CTRL,
MPU6050_DELAYCTRL_DELAY_ES_SHADOW_BIT, enabled);
}
/** Get slave delay enabled status.
 * When a particular slave delay is enabled, the rate of access for the that
 * slave device is reduced. When a slave's access rate is decreased relative to
 * the Sample Rate, the slave is accessed every:
 *
 *     1 / (1 + I2C_MST_DLY) Samples
 *
 * This base Sample Rate in turn is determined by SMPLRT_DIV (register  * 25)
 * and DLPF_CFG (register 26).
 *
 * For further information regarding I2C_MST_DLY, please refer to register 52.
 * For further information regarding the Sample Rate, please refer to register 25.
 *
 * @param num Slave number (0-4)
 * @return Current slave delay enabled status.
 * @see MPU6050_RA_I2C_MST_DELAY_CTRL
 * @see MPU6050_DELAYCTRL_I2C_SLV0_DLY_EN_BIT
 */
bool MPU6050::getSlaveDelayEnabled(uint8_t num) {
    // MPU6050_DELAYCTRL_I2C_SLV4_DLY_EN_BIT is 4, SLV3 is 3, etc.
    if (num > 4) return 0;
    I2Cdev::readBit(devAddr, MPU6050_RA_I2C_MST_DELAY_CTRL, num, buffer);
    return buffer[0];
}
/** Set slave delay enabled status.
```

```
 * @param num Slave number (0-4)
 * @param enabled New slave delay enabled status.
 * @see MPU6050_RA_I2C_MST_DELAY_CTRL
 * @see MPU6050_DELAYCTRL_I2C_SLV0_DLY_EN_BIT
 */
void MPU6050::setSlaveDelayEnabled(uint8_t num, bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_I2C_MST_DELAY_CTRL, num, enabled);
}

// SIGNAL_PATH_RESET register

/** Reset gyroscope signal path.
 * The reset will revert the signal path analog to digital converters and
 * filters to their power up configurations.
 * @see MPU6050_RA_SIGNAL_PATH_RESET
 * @see MPU6050_PATHRESET_GYRO_RESET_BIT
 */
void MPU6050::resetGyroscopePath() {
    I2Cdev::writeBit(devAddr, MPU6050_RA_SIGNAL_PATH_RESET,
MPU6050_PATHRESET_GYRO_RESET_BIT, true);
}
/** Reset accelerometer signal path.
 * The reset will revert the signal path analog to digital converters and
 * filters to their power up configurations.
 * @see MPU6050_RA_SIGNAL_PATH_RESET
 * @see MPU6050_PATHRESET_ACCEL_RESET_BIT
 */
void MPU6050::resetAccelerometerPath() {
    I2Cdev::writeBit(devAddr, MPU6050_RA_SIGNAL_PATH_RESET,
MPU6050_PATHRESET_ACCEL_RESET_BIT, true);
}
/** Reset temperature sensor signal path.
 * The reset will revert the signal path analog to digital converters and
 * filters to their power up configurations.
 * @see MPU6050_RA_SIGNAL_PATH_RESET
 * @see MPU6050_PATHRESET_TEMP_RESET_BIT
 */
void MPU6050::resetTemperaturePath() {
    I2Cdev::writeBit(devAddr, MPU6050_RA_SIGNAL_PATH_RESET,
MPU6050_PATHRESET_TEMP_RESET_BIT, true);
}

// MOT_DETECT_CTRL register

/** Get accelerometer power-on delay.
 * The accelerometer data path provides samples to the sensor registers, Motion
 * detection, Zero Motion detection, and Free Fall detection modules. The
 * signal path contains filters which must be flushed on wake-up with new
 * samples before the detection modules begin operations. The default wake-up
 * delay, of 4ms can be lengthened by up to 3ms. This additional delay is
 * specified in ACCEL_ON_DELAY in units of 1 LSB = 1 ms. The user may select
```

```
 * any value above zero unless instructed otherwise by InvenSense. Please refer
 * to Section 8 of the MPU-6000/MPU-6050 Product Specification document for
 * further information regarding the detection modules.
 * @return Current accelerometer power-on delay
 * @see MPU6050_RA_MOT_DETECT_CTRL
 * @see MPU6050_DETECT_ACCEL_ON_DELAY_BIT
 */
uint8_t MPU6050::getAccelerometerPowerOnDelay() {
    I2Cdev::readBits(devAddr, MPU6050_RA_MOT_DETECT_CTRL,
MPU6050_DETECT_ACCEL_ON_DELAY_BIT,
MPU6050_DETECT_ACCEL_ON_DELAY_LENGTH, buffer);
    return buffer[0];
}
/** Set accelerometer power-on delay.
 * @param delay New accelerometer power-on delay (0-3)
 * @see getAccelerometerPowerOnDelay()
 * @see MPU6050_RA_MOT_DETECT_CTRL
 * @see MPU6050_DETECT_ACCEL_ON_DELAY_BIT
 */
void MPU6050::setAccelerometerPowerOnDelay(uint8_t delay) {
    I2Cdev::writeBits(devAddr, MPU6050_RA_MOT_DETECT_CTRL,
MPU6050_DETECT_ACCEL_ON_DELAY_BIT,
MPU6050_DETECT_ACCEL_ON_DELAY_LENGTH, delay);
}
/** Get Free Fall detection counter decrement configuration.
 * Detection is registered by the Free Fall detection module after accelerometer
 * measurements meet their respective threshold conditions over a specified
 * number of samples. When the threshold conditions are met, the corresponding
 * detection counter increments by 1. The user may control the rate at which the
 * detection counter decrements when the threshold condition is not met by
 * configuring FF_COUNT. The decrement rate can be set according to the
 * following table:
 *
 * <pre>
 * FF_COUNT | Counter Decrement
 * ---------+------------------
 * 0        | Reset
 * 1        | 1
 * 2        | 2
 * 3        | 4
 * </pre>
 *
 * When FF_COUNT is configured to 0 (reset), any non-qualifying sample will
 * reset the counter to 0. For further information on Free Fall detection,
 * please refer to Registers 29 to 32.
 *
 * @return Current decrement configuration
 * @see MPU6050_RA_MOT_DETECT_CTRL
 * @see MPU6050_DETECT_FF_COUNT_BIT
 */
uint8_t MPU6050::getFreefallDetectionCounterDecrement() {
```

```
    I2Cdev::readBits(devAddr, MPU6050_RA_MOT_DETECT_CTRL,
MPU6050_DETECT_FF_COUNT_BIT, MPU6050_DETECT_FF_COUNT_LENGTH,
buffer);
    return buffer[0];
}
/** Set Free Fall detection counter decrement configuration.
 * @param decrement New decrement configuration value
 * @see getFreefallDetectionCounterDecrement()
 * @see MPU6050_RA_MOT_DETECT_CTRL
 * @see MPU6050_DETECT_FF_COUNT_BIT
 */
void MPU6050::setFreefallDetectionCounterDecrement(uint8_t decrement) {
    I2Cdev::writeBits(devAddr, MPU6050_RA_MOT_DETECT_CTRL,
MPU6050_DETECT_FF_COUNT_BIT, MPU6050_DETECT_FF_COUNT_LENGTH,
decrement);
}
/** Get Motion detection counter decrement configuration.
 * Detection is registered by the Motion detection module after accelerometer
 * measurements meet their respective threshold conditions over a specified
 * number of samples. When the threshold conditions are met, the corresponding
 * detection counter increments by 1. The user may control the rate at which the
 * detection counter decrements when the threshold condition is not met by
 * configuring MOT_COUNT. The decrement rate can be set according to the
 * following table:
 *
 * <pre>
 * MOT_COUNT | Counter Decrement
 * ----------+------------------
 * 0         | Reset
 * 1         | 1
 * 2         | 2
 * 3         | 4
 * </pre>
 *
 * When MOT_COUNT is configured to 0 (reset), any non-qualifying sample will
 * reset the counter to 0. For further information on Motion detection,
 * please refer to Registers 29 to 32.
 *
 */
uint8_t MPU6050::getMotionDetectionCounterDecrement() {
    I2Cdev::readBits(devAddr, MPU6050_RA_MOT_DETECT_CTRL,
MPU6050_DETECT_MOT_COUNT_BIT,
MPU6050_DETECT_MOT_COUNT_LENGTH, buffer);
    return buffer[0];
}
/** Set Motion detection counter decrement configuration.
 * @param decrement New decrement configuration value
 * @see getMotionDetectionCounterDecrement()
 * @see MPU6050_RA_MOT_DETECT_CTRL
 * @see MPU6050_DETECT_MOT_COUNT_BIT
 */
```

```
void MPU6050::setMotionDetectionCounterDecrement(uint8_t decrement) {
    I2Cdev::writeBits(devAddr, MPU6050_RA_MOT_DETECT_CTRL,
MPU6050_DETECT_MOT_COUNT_BIT,
MPU6050_DETECT_MOT_COUNT_LENGTH, decrement);
}

// USER_CTRL register

/** Get FIFO enabled status.
 * When this bit is set to 0, the FIFO buffer is disabled. The FIFO buffer
 * cannot be written to or read from while disabled. The FIFO buffer's state
 * does not change unless the MPU-60X0 is power cycled.
 * @return Current FIFO enabled status
 * @see MPU6050_RA_USER_CTRL
 * @see MPU6050_USERCTRL_FIFO_EN_BIT
 */
bool MPU6050::getFIFOEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_USER_CTRL,
MPU6050_USERCTRL_FIFO_EN_BIT, buffer);
    return buffer[0];
}
/** Set FIFO enabled status.
 * @param enabled New FIFO enabled status
 * @see getFIFOEnabled()
 * @see MPU6050_RA_USER_CTRL
 * @see MPU6050_USERCTRL_FIFO_EN_BIT
 */
void MPU6050::setFIFOEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL,
MPU6050_USERCTRL_FIFO_EN_BIT, enabled);
}
/** Get I2C Master Mode enabled status.
 * When this mode is enabled, the MPU-60X0 acts as the I2C Master to the
 * external sensor slave devices on the auxiliary I2C bus. When this bit is
 * cleared to 0, the auxiliary I2C bus lines (AUX_DA and AUX_CL) are logically
 * driven by the primary I2C bus (SDA and SCL). This is a precondition to
 * enabling Bypass Mode. For further information regarding Bypass Mode, please
 * refer to Register 55.
 * @return Current I2C Master Mode enabled status
 * @see MPU6050_RA_USER_CTRL
 * @see MPU6050_USERCTRL_I2C_MST_EN_BIT
 */
bool MPU6050::getI2CMasterModeEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_USER_CTRL,
MPU6050_USERCTRL_I2C_MST_EN_BIT, buffer);
    return buffer[0];
}
/** Set I2C Master Mode enabled status.
 * @param enabled New I2C Master Mode enabled status
 * @see getI2CMasterModeEnabled()
 * @see MPU6050_RA_USER_CTRL
```

```
 * @see MPU6050_USERCTRL_I2C_MST_EN_BIT
 */
void MPU6050::setI2CMasterModeEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL,
MPU6050_USERCTRL_I2C_MST_EN_BIT, enabled);
}
/** Switch from I2C to SPI mode (MPU-6000 only)
 * If this is set, the primary SPI interface will be enabled in place of the
 * disabled primary I2C interface.
 */
void MPU6050::switchSPIEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL,
MPU6050_USERCTRL_I2C_IF_DIS_BIT, enabled);
}
/** Reset the FIFO.
 * This bit resets the FIFO buffer when set to 1 while FIFO_EN equals 0. This
 * bit automatically clears to 0 after the reset has been triggered.
 * @see MPU6050_RA_USER_CTRL
 * @see MPU6050_USERCTRL_FIFO_RESET_BIT
 */
void MPU6050::resetFIFO() {
    I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL,
MPU6050_USERCTRL_FIFO_RESET_BIT, true);
}
/** Reset the I2C Master.
 * This bit resets the I2C Master when set to 1 while I2C_MST_EN equals 0.
 * This bit automatically clears to 0 after the reset has been triggered.
 * @see MPU6050_RA_USER_CTRL
 * @see MPU6050_USERCTRL_I2C_MST_RESET_BIT
 */
void MPU6050::resetI2CMaster() {
    I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL,
MPU6050_USERCTRL_I2C_MST_RESET_BIT, true);
}
/** Reset all sensor registers and signal paths.
 * When set to 1, this bit resets the signal paths for all sensors (gyroscopes,
 * accelerometers, and temperature sensor). This operation will also clear the
 * sensor registers. This bit automatically clears to 0 after the reset has been
 * triggered.
 *
 * When resetting only the signal path (and not the sensor registers), please
 * use Register 104, SIGNAL_PATH_RESET.
 *
 * @see MPU6050_RA_USER_CTRL
 * @see MPU6050_USERCTRL_SIG_COND_RESET_BIT
 */
void MPU6050::resetSensors() {
    I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL,
MPU6050_USERCTRL_SIG_COND_RESET_BIT, true);
}
```

```
// PWR_MGMT_1 register

/** Trigger a full device reset.
 * A small delay of ~50ms may be desirable after triggering a reset.
 * @see MPU6050_RA_PWR_MGMT_1
 * @see MPU6050_PWR1_DEVICE_RESET_BIT
 */
void MPU6050::reset() {
    I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_1,
MPU6050_PWR1_DEVICE_RESET_BIT, true);
}
/** Get sleep mode status.
 * Setting the SLEEP bit in the register puts the device into very low power
 * sleep mode. In this mode, only the serial interface and internal registers
 * remain active, allowing for a very low standby current. Clearing this bit
 * puts the device back into normal mode. To save power, the individual standby
 * selections for each of the gyros should be used if any gyro axis is not used
 * by the application.
 * @return Current sleep mode enabled status
 * @see MPU6050_RA_PWR_MGMT_1
 * @see MPU6050_PWR1_SLEEP_BIT
 */
bool MPU6050::getSleepEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_1,
MPU6050_PWR1_SLEEP_BIT, buffer);
    return buffer[0];
}
/** Set sleep mode status.
 * @param enabled New sleep mode enabled status
 * @see getSleepEnabled()
 * @see MPU6050_RA_PWR_MGMT_1
 * @see MPU6050_PWR1_SLEEP_BIT
 */
void MPU6050::setSleepEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_1,
MPU6050_PWR1_SLEEP_BIT, enabled);
}
/** Get wake cycle enabled status.
 * When this bit is set to 1 and SLEEP is disabled, the MPU-60X0 will cycle
 * between sleep mode and waking up to take a single sample of data from active
 * sensors at a rate determined by LP_WAKE_CTRL (register 108).
 * @return Current sleep mode enabled status
 * @see MPU6050_RA_PWR_MGMT_1
 * @see MPU6050_PWR1_CYCLE_BIT
 */
bool MPU6050::getWakeCycleEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_1,
MPU6050_PWR1_CYCLE_BIT, buffer);
    return buffer[0];
}
/** Set wake cycle enabled status.
```

```
 * @param enabled New sleep mode enabled status
 * @see getWakeCycleEnabled()
 * @see MPU6050_RA_PWR_MGMT_1
 * @see MPU6050_PWR1_CYCLE_BIT
 */
void MPU6050::setWakeCycleEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_1,
MPU6050_PWR1_CYCLE_BIT, enabled);
}
/** Get temperature sensor enabled status.
 * Control the usage of the internal temperature sensor.
 *
 * Note: this register stores the *disabled* value, but for consistency with the
 * rest of the code, the function is named and used with standard true/false
 * values to indicate whether the sensor is enabled or disabled, respectively.
 *
 * @return Current temperature sensor enabled status
 * @see MPU6050_RA_PWR_MGMT_1
 * @see MPU6050_PWR1_TEMP_DIS_BIT
 */
bool MPU6050::getTempSensorEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_1,
MPU6050_PWR1_TEMP_DIS_BIT, buffer);
    return buffer[0] == 0; // 1 is actually disabled here
}
/** Set temperature sensor enabled status.
 * Note: this register stores the *disabled* value, but for consistency with the
 * rest of the code, the function is named and used with standard true/false
 * values to indicate whether the sensor is enabled or disabled, respectively.
 *
 * @param enabled New temperature sensor enabled status
 * @see getTempSensorEnabled()
 * @see MPU6050_RA_PWR_MGMT_1
 * @see MPU6050_PWR1_TEMP_DIS_BIT
 */
void MPU6050::setTempSensorEnabled(bool enabled) {
    // 1 is actually disabled here
    I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_1,
MPU6050_PWR1_TEMP_DIS_BIT, !enabled);
}
/** Get clock source setting.
 * @return Current clock source setting
 * @see MPU6050_RA_PWR_MGMT_1
 * @see MPU6050_PWR1_CLKSEL_BIT
 * @see MPU6050_PWR1_CLKSEL_LENGTH
 */
uint8_t MPU6050::getClockSource() {
    I2Cdev::readBits(devAddr, MPU6050_RA_PWR_MGMT_1,
MPU6050_PWR1_CLKSEL_BIT, MPU6050_PWR1_CLKSEL_LENGTH, buffer);
    return buffer[0];
}
```

```
/** Set clock source setting.
 * An internal 8MHz oscillator, gyroscope based clock, or external sources can
 * be selected as the MPU-60X0 clock source. When the internal 8 MHz oscillator
 * or an external source is chosen as the clock source, the MPU-60X0 can operate
 * in low power modes with the gyroscopes disabled.
 *
 * Upon power up, the MPU-60X0 clock source defaults to the internal oscillator.
 * However, it is highly recommended that the device be configured to use one of
 * the gyroscopes (or an external clock source) as the clock reference for
 * improved stability. The clock source can be selected according to the following table:
 *
 * <pre>
 * CLK_SEL | Clock Source
 * --------+-------------------------------------
 * 0       | Internal oscillator
 * 1       | PLL with X Gyro reference
 * 2       | PLL with Y Gyro reference
 * 3       | PLL with Z Gyro reference
 * 4       | PLL with external 32.768kHz reference
 * 5       | PLL with external 19.2MHz reference
 * 6       | Reserved
 * 7       | Stops the clock and keeps the timing generator in reset
 * </pre>
 *
 * @param source New clock source setting
 * @see getClockSource()
 * @see MPU6050_RA_PWR_MGMT_1
 * @see MPU6050_PWR1_CLKSEL_BIT
 * @see MPU6050_PWR1_CLKSEL_LENGTH
 */
void MPU6050::setClockSource(uint8_t source) {
    I2Cdev::writeBits(devAddr, MPU6050_RA_PWR_MGMT_1,
MPU6050_PWR1_CLKSEL_BIT, MPU6050_PWR1_CLKSEL_LENGTH, source);
}

// PWR_MGMT_2 register

/** Get wake frequency in Accel-Only Low Power Mode.
 * The MPU-60X0 can be put into Accerlerometer Only Low Power Mode by setting
 * PWRSEL to 1 in the Power Management 1 register (Register 107). In this mode,
 * the device will power off all devices except for the primary I2C interface,
 * waking only the accelerometer at fixed intervals to take a single
 * measurement. The frequency of wake-ups can be configured with LP_WAKE_CTRL
 * as shown below:
 *
 * <pre>
 * LP_WAKE_CTRL | Wake-up Frequency
 * -------------+------------------
 * 0            | 1.25 Hz
 * 1            | 2.5 Hz
 * 2            | 5 Hz
```

```
 * 3           | 10 Hz
 * <pre>
 *
 * For further information regarding the MPU-60X0's power modes, please refer to
 * Register 107.
 *
 * @return Current wake frequency
 * @see MPU6050_RA_PWR_MGMT_2
 */
uint8_t MPU6050::getWakeFrequency() {
    I2Cdev::readBits(devAddr, MPU6050_RA_PWR_MGMT_2,
MPU6050_PWR2_LP_WAKE_CTRL_BIT,
MPU6050_PWR2_LP_WAKE_CTRL_LENGTH, buffer);
    return buffer[0];
}
/** Set wake frequency in Accel-Only Low Power Mode.
 * @param frequency New wake frequency
 * @see MPU6050_RA_PWR_MGMT_2
 */
void MPU6050::setWakeFrequency(uint8_t frequency) {
    I2Cdev::writeBits(devAddr, MPU6050_RA_PWR_MGMT_2,
MPU6050_PWR2_LP_WAKE_CTRL_BIT,
MPU6050_PWR2_LP_WAKE_CTRL_LENGTH, frequency);
}

/** Get X-axis accelerometer standby enabled status.
 * If enabled, the X-axis will not gather or report data (or use power).
 * @return Current X-axis standby enabled status
 * @see MPU6050_RA_PWR_MGMT_2
 * @see MPU6050_PWR2_STBY_XA_BIT
 */
bool MPU6050::getStandbyXAccelEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_2,
MPU6050_PWR2_STBY_XA_BIT, buffer);
    return buffer[0];
}
/** Set X-axis accelerometer standby enabled status.
 * @param New X-axis standby enabled status
 * @see getStandbyXAccelEnabled()
 * @see MPU6050_RA_PWR_MGMT_2
 * @see MPU6050_PWR2_STBY_XA_BIT
 */
void MPU6050::setStandbyXAccelEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_2,
MPU6050_PWR2_STBY_XA_BIT, enabled);
}
/** Get Y-axis accelerometer standby enabled status.
 * If enabled, the Y-axis will not gather or report data (or use power).
 * @return Current Y-axis standby enabled status
 * @see MPU6050_RA_PWR_MGMT_2
 * @see MPU6050_PWR2_STBY_YA_BIT
```

```
 */
bool MPU6050::getStandbyYAccelEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_2,
MPU6050_PWR2_STBY_YA_BIT, buffer);
    return buffer[0];
}
/** Set Y-axis accelerometer standby enabled status.
 * @param New Y-axis standby enabled status
 * @see getStandbyYAccelEnabled()
 * @see MPU6050_RA_PWR_MGMT_2
 * @see MPU6050_PWR2_STBY_YA_BIT
 */
void MPU6050::setStandbyYAccelEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_2,
MPU6050_PWR2_STBY_YA_BIT, enabled);
}
/** Get Z-axis accelerometer standby enabled status.
 * If enabled, the Z-axis will not gather or report data (or use power).
 * @return Current Z-axis standby enabled status
 * @see MPU6050_RA_PWR_MGMT_2
 * @see MPU6050_PWR2_STBY_ZA_BIT
 */
bool MPU6050::getStandbyZAccelEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_2,
MPU6050_PWR2_STBY_ZA_BIT, buffer);
    return buffer[0];
}
/** Set Z-axis accelerometer standby enabled status.
 * @param New Z-axis standby enabled status
 * @see getStandbyZAccelEnabled()
 * @see MPU6050_RA_PWR_MGMT_2
 * @see MPU6050_PWR2_STBY_ZA_BIT
 */
void MPU6050::setStandbyZAccelEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_2,
MPU6050_PWR2_STBY_ZA_BIT, enabled);
}
/** Get X-axis gyroscope standby enabled status.
 * If enabled, the X-axis will not gather or report data (or use power).
 * @return Current X-axis standby enabled status
 * @see MPU6050_RA_PWR_MGMT_2
 * @see MPU6050_PWR2_STBY_XG_BIT
 */
bool MPU6050::getStandbyXGyroEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_2,
MPU6050_PWR2_STBY_XG_BIT, buffer);
    return buffer[0];
}
/** Set X-axis gyroscope standby enabled status.
 * @param New X-axis standby enabled status
 * @see getStandbyXGyroEnabled()
```

```
 * @see MPU6050_RA_PWR_MGMT_2
 * @see MPU6050_PWR2_STBY_XG_BIT
 */
void MPU6050::setStandbyXGyroEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_2,
MPU6050_PWR2_STBY_XG_BIT, enabled);
}
/** Get Y-axis gyroscope standby enabled status.
 * If enabled, the Y-axis will not gather or report data (or use power).
 * @return Current Y-axis standby enabled status
 * @see MPU6050_RA_PWR_MGMT_2
 * @see MPU6050_PWR2_STBY_YG_BIT
 */
bool MPU6050::getStandbyYGyroEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_2,
MPU6050_PWR2_STBY_YG_BIT, buffer);
    return buffer[0];
}
/** Set Y-axis gyroscope standby enabled status.
 * @param New Y-axis standby enabled status
 * @see getStandbyYGyroEnabled()
 * @see MPU6050_RA_PWR_MGMT_2
 * @see MPU6050_PWR2_STBY_YG_BIT
 */
void MPU6050::setStandbyYGyroEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_2,
MPU6050_PWR2_STBY_YG_BIT, enabled);
}
/** Get Z-axis gyroscope standby enabled status.
 * If enabled, the Z-axis will not gather or report data (or use power).
 * @return Current Z-axis standby enabled status
 * @see MPU6050_RA_PWR_MGMT_2
 * @see MPU6050_PWR2_STBY_ZG_BIT
 */
bool MPU6050::getStandbyZGyroEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_PWR_MGMT_2,
MPU6050_PWR2_STBY_ZG_BIT, buffer);
    return buffer[0];
}
/** Set Z-axis gyroscope standby enabled status.
 * @param New Z-axis standby enabled status
 * @see getStandbyZGyroEnabled()
 * @see MPU6050_RA_PWR_MGMT_2
 * @see MPU6050_PWR2_STBY_ZG_BIT
 */
void MPU6050::setStandbyZGyroEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_PWR_MGMT_2,
MPU6050_PWR2_STBY_ZG_BIT, enabled);
}

// FIFO_COUNT* registers
```

```
/** Get current FIFO buffer size.
 * This value indicates the number of bytes stored in the FIFO buffer. This
 * number is in turn the number of bytes that can be read from the FIFO buffer
 * and it is directly proportional to the number of samples available given the
 * set of sensor data bound to be stored in the FIFO (register 35 and 36).
 * @return Current FIFO buffer size
 */
uint16_t MPU6050::getFIFOCount() {
    I2Cdev::readBytes(devAddr, MPU6050_RA_FIFO_COUNTH, 2, buffer);
    return (((uint16_t)buffer[0]) << 8) | buffer[1];
}

// FIFO_R_W register

/** Get byte from FIFO buffer.
 * This register is used to read and write data from the FIFO buffer. Data is
 * written to the FIFO in order of register number (from lowest to highest). If
 * all the FIFO enable flags (see below) are enabled and all External Sensor
 * Data registers (Registers 73 to 96) are associated with a Slave device, the
 * contents of registers 59 through 96 will be written in order at the Sample
 * Rate.
 *
 * The contents of the sensor data registers (Registers 59 to 96) are written
 * into the FIFO buffer when their corresponding FIFO enable flags are set to 1
 * in FIFO_EN (Register 35). An additional flag for the sensor data registers
 * associated with I2C Slave 3 can be found in I2C_MST_CTRL (Register 36).
 *
 * If the FIFO buffer has overflowed, the status bit FIFO_OFLOW_INT is
 * automatically set to 1. This bit is located in INT_STATUS (Register 58).
 * When the FIFO buffer has overflowed, the oldest data will be lost and new
 * data will be written to the FIFO.
 *
 * If the FIFO buffer is empty, reading this register will return the last byte
 * that was previously read from the FIFO until new data is available. The user
 * should check FIFO_COUNT to ensure that the FIFO buffer is not read when
 * empty.
 *
 * @return Byte from FIFO buffer
 */
uint8_t MPU6050::getFIFOByte() {
    I2Cdev::readByte(devAddr, MPU6050_RA_FIFO_R_W, buffer);
    return buffer[0];
}
void MPU6050::getFIFOBytes(uint8_t *data, uint8_t length) {
    I2Cdev::readBytes(devAddr, MPU6050_RA_FIFO_R_W, length, data);
}
/** Write byte to FIFO buffer.
 * @see getFIFOByte()
 * @see MPU6050_RA_FIFO_R_W
 */
```

```
void MPU6050::setFIFOByte(uint8_t data) {
    I2Cdev::writeByte(devAddr, MPU6050_RA_FIFO_R_W, data);
}

// WHO_AM_I register

/** Get Device ID.
 * This register is used to verify the identity of the device (0b110100, 0x34).
 * @return Device ID (6 bits only! should be 0x34)
 * @see MPU6050_RA_WHO_AM_I
 * @see MPU6050_WHO_AM_I_BIT
 * @see MPU6050_WHO_AM_I_LENGTH
 */
uint8_t MPU6050::getDeviceID() {
    I2Cdev::readBits(devAddr, MPU6050_RA_WHO_AM_I, MPU6050_WHO_AM_I_BIT,
MPU6050_WHO_AM_I_LENGTH, buffer);
    return buffer[0];
}
/** Set Device ID.
 * Write a new ID into the WHO_AM_I register (no idea why this should ever be
 * necessary though).
 * @param id New device ID to set.
 * @see getDeviceID()
 * @see MPU6050_RA_WHO_AM_I
 * @see MPU6050_WHO_AM_I_BIT
 * @see MPU6050_WHO_AM_I_LENGTH
 */
void MPU6050::setDeviceID(uint8_t id) {
    I2Cdev::writeBits(devAddr, MPU6050_RA_WHO_AM_I, MPU6050_WHO_AM_I_BIT,
MPU6050_WHO_AM_I_LENGTH, id);
}

// ======== UNDOCUMENTED/DMP REGISTERS/METHODS ========

// XG_OFFS_TC register

uint8_t MPU6050::getOTPBankValid() {
    I2Cdev::readBit(devAddr, MPU6050_RA_XG_OFFS_TC,
MPU6050_TC_OTP_BNK_VLD_BIT, buffer);
    return buffer[0];
}
void MPU6050::setOTPBankValid(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_XG_OFFS_TC,
MPU6050_TC_OTP_BNK_VLD_BIT, enabled);
}
int8_t MPU6050::getXGyroOffset() {
    I2Cdev::readBits(devAddr, MPU6050_RA_XG_OFFS_TC,
MPU6050_TC_OFFSET_BIT, MPU6050_TC_OFFSET_LENGTH, buffer);
    return buffer[0];
}
void MPU6050::setXGyroOffset(int8_t offset) {
```

```
    I2Cdev::writeBits(devAddr, MPU6050_RA_XG_OFFS_TC,
MPU6050_TC_OFFSET_BIT, MPU6050_TC_OFFSET_LENGTH, offset);
}

// YG_OFFS_TC register

int8_t MPU6050::getYGyroOffset() {
    I2Cdev::readBits(devAddr, MPU6050_RA_YG_OFFS_TC,
MPU6050_TC_OFFSET_BIT, MPU6050_TC_OFFSET_LENGTH, buffer);
    return buffer[0];
}
void MPU6050::setYGyroOffset(int8_t offset) {
    I2Cdev::writeBits(devAddr, MPU6050_RA_YG_OFFS_TC,
MPU6050_TC_OFFSET_BIT, MPU6050_TC_OFFSET_LENGTH, offset);
}

// ZG_OFFS_TC register

int8_t MPU6050::getZGyroOffset() {
    I2Cdev::readBits(devAddr, MPU6050_RA_ZG_OFFS_TC,
MPU6050_TC_OFFSET_BIT, MPU6050_TC_OFFSET_LENGTH, buffer);
    return buffer[0];
}
void MPU6050::setZGyroOffset(int8_t offset) {
    I2Cdev::writeBits(devAddr, MPU6050_RA_ZG_OFFS_TC,
MPU6050_TC_OFFSET_BIT, MPU6050_TC_OFFSET_LENGTH, offset);
}

// X_FINE_GAIN register

int8_t MPU6050::getXFineGain() {
    I2Cdev::readByte(devAddr, MPU6050_RA_X_FINE_GAIN, buffer);
    return buffer[0];
}
void MPU6050::setXFineGain(int8_t gain) {
    I2Cdev::writeByte(devAddr, MPU6050_RA_X_FINE_GAIN, gain);
}

// Y_FINE_GAIN register

int8_t MPU6050::getYFineGain() {
    I2Cdev::readByte(devAddr, MPU6050_RA_Y_FINE_GAIN, buffer);
    return buffer[0];
}
void MPU6050::setYFineGain(int8_t gain) {
    I2Cdev::writeByte(devAddr, MPU6050_RA_Y_FINE_GAIN, gain);
}

// Z_FINE_GAIN register

int8_t MPU6050::getZFineGain() {
```

```
    I2Cdev::readByte(devAddr, MPU6050_RA_Z_FINE_GAIN, buffer);
    return buffer[0];
}
void MPU6050::setZFineGain(int8_t gain) {
    I2Cdev::writeByte(devAddr, MPU6050_RA_Z_FINE_GAIN, gain);
}

// XA_OFFS_* registers

int16_t MPU6050::getXAccelOffset() {
    I2Cdev::readBytes(devAddr, MPU6050_RA_XA_OFFS_H, 2, buffer);
    return (((int16_t)buffer[0]) << 8) | buffer[1];
}
void MPU6050::setXAccelOffset(int16_t offset) {
    I2Cdev::writeWord(devAddr, MPU6050_RA_XA_OFFS_H, offset);
}

// YA_OFFS_* register

int16_t MPU6050::getYAccelOffset() {
    I2Cdev::readBytes(devAddr, MPU6050_RA_YA_OFFS_H, 2, buffer);
    return (((int16_t)buffer[0]) << 8) | buffer[1];
}
void MPU6050::setYAccelOffset(int16_t offset) {
    I2Cdev::writeWord(devAddr, MPU6050_RA_YA_OFFS_H, offset);
}

// ZA_OFFS_* register

int16_t MPU6050::getZAccelOffset() {
    I2Cdev::readBytes(devAddr, MPU6050_RA_ZA_OFFS_H, 2, buffer);
    return (((int16_t)buffer[0]) << 8) | buffer[1];
}
void MPU6050::setZAccelOffset(int16_t offset) {
    I2Cdev::writeWord(devAddr, MPU6050_RA_ZA_OFFS_H, offset);
}

// XG_OFFS_USR* registers

int16_t MPU6050::getXGyroOffsetUser() {
    I2Cdev::readBytes(devAddr, MPU6050_RA_XG_OFFS_USRH, 2, buffer);
    return (((int16_t)buffer[0]) << 8) | buffer[1];
}
void MPU6050::setXGyroOffsetUser(int16_t offset) {
    I2Cdev::writeWord(devAddr, MPU6050_RA_XG_OFFS_USRH, offset);
}

// YG_OFFS_USR* register

int16_t MPU6050::getYGyroOffsetUser() {
    I2Cdev::readBytes(devAddr, MPU6050_RA_YG_OFFS_USRH, 2, buffer);
```

```
    return (((int16_t)buffer[0]) << 8) | buffer[1];
}
void MPU6050::setYGyroOffsetUser(int16_t offset) {
    I2Cdev::writeWord(devAddr, MPU6050_RA_YG_OFFS_USRH, offset);
}

// ZG_OFFS_USR* register

int16_t MPU6050::getZGyroOffsetUser() {
    I2Cdev::readBytes(devAddr, MPU6050_RA_ZG_OFFS_USRH, 2, buffer);
    return (((int16_t)buffer[0]) << 8) | buffer[1];
}
void MPU6050::setZGyroOffsetUser(int16_t offset) {
    I2Cdev::writeWord(devAddr, MPU6050_RA_ZG_OFFS_USRH, offset);
}

// INT_ENABLE register (DMP functions)

bool MPU6050::getIntPLLReadyEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_INT_ENABLE,
MPU6050_INTERRUPT_PLL_RDY_INT_BIT, buffer);
    return buffer[0];
}
void MPU6050::setIntPLLReadyEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_ENABLE,
MPU6050_INTERRUPT_PLL_RDY_INT_BIT, enabled);
}
bool MPU6050::getIntDMPEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_INT_ENABLE,
MPU6050_INTERRUPT_DMP_INT_BIT, buffer);
    return buffer[0];
}
void MPU6050::setIntDMPEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_INT_ENABLE,
MPU6050_INTERRUPT_DMP_INT_BIT, enabled);
}

// DMP_INT_STATUS

bool MPU6050::getDMPInt5Status() {
    I2Cdev::readBit(devAddr, MPU6050_RA_DMP_INT_STATUS,
MPU6050_DMPINT_5_BIT, buffer);
    return buffer[0];
}
bool MPU6050::getDMPInt4Status() {
    I2Cdev::readBit(devAddr, MPU6050_RA_DMP_INT_STATUS,
MPU6050_DMPINT_4_BIT, buffer);
    return buffer[0];
}
bool MPU6050::getDMPInt3Status() {
    I2Cdev::readBit(devAddr, MPU6050_RA_DMP_INT_STATUS,
```

```
MPU6050_DMPINT_3_BIT, buffer);
    return buffer[0];
}
bool MPU6050::getDMPInt2Status() {
    I2Cdev::readBit(devAddr, MPU6050_RA_DMP_INT_STATUS,
MPU6050_DMPINT_2_BIT, buffer);
    return buffer[0];
}
bool MPU6050::getDMPInt1Status() {
    I2Cdev::readBit(devAddr, MPU6050_RA_DMP_INT_STATUS,
MPU6050_DMPINT_1_BIT, buffer);
    return buffer[0];
}
bool MPU6050::getDMPInt0Status() {
    I2Cdev::readBit(devAddr, MPU6050_RA_DMP_INT_STATUS,
MPU6050_DMPINT_0_BIT, buffer);
    return buffer[0];
}

// INT_STATUS register (DMP functions)

bool MPU6050::getIntPLLReadyStatus() {
    I2Cdev::readBit(devAddr, MPU6050_RA_INT_STATUS,
MPU6050_INTERRUPT_PLL_RDY_INT_BIT, buffer);
    return buffer[0];
}
bool MPU6050::getIntDMPStatus() {
    I2Cdev::readBit(devAddr, MPU6050_RA_INT_STATUS,
MPU6050_INTERRUPT_DMP_INT_BIT, buffer);
    return buffer[0];
}

// USER_CTRL register (DMP functions)

bool MPU6050::getDMPEnabled() {
    I2Cdev::readBit(devAddr, MPU6050_RA_USER_CTRL,
MPU6050_USERCTRL_DMP_EN_BIT, buffer);
    return buffer[0];
}
void MPU6050::setDMPEnabled(bool enabled) {
    I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL,
MPU6050_USERCTRL_DMP_EN_BIT, enabled);
}
void MPU6050::resetDMP() {
    I2Cdev::writeBit(devAddr, MPU6050_RA_USER_CTRL,
MPU6050_USERCTRL_DMP_RESET_BIT, true);
}

// BANK_SEL register

void MPU6050::setMemoryBank(uint8_t bank, bool prefetchEnabled, bool userBank) {
```

```
    bank &= 0x1F;
    if (userBank) bank |= 0x20;
    if (prefetchEnabled) bank |= 0x40;
    I2Cdev::writeByte(devAddr, MPU6050_RA_BANK_SEL, bank);
}

// MEM_START_ADDR register

void MPU6050::setMemoryStartAddress(uint8_t address) {
    I2Cdev::writeByte(devAddr, MPU6050_RA_MEM_START_ADDR, address);
}

// MEM_R_W register

uint8_t MPU6050::readMemoryByte() {
    I2Cdev::readByte(devAddr, MPU6050_RA_MEM_R_W, buffer);
    return buffer[0];
}
void MPU6050::writeMemoryByte(uint8_t data) {
    I2Cdev::writeByte(devAddr, MPU6050_RA_MEM_R_W, data);
}
void MPU6050::readMemoryBlock(uint8_t *data, uint16_t dataSize, uint8_t bank, uint8_t
address) {
    setMemoryBank(bank);
    setMemoryStartAddress(address);
    uint8_t chunkSize;
    for (uint16_t i = 0; i < dataSize;) {
        // determine correct chunk size according to bank position and data size
        chunkSize = MPU6050_DMP_MEMORY_CHUNK_SIZE;

        // make sure we don't go past the data size
        if (i + chunkSize > dataSize) chunkSize = dataSize - i;

        // make sure this chunk doesn't go past the bank boundary (256 bytes)
        if (chunkSize > 256 - address) chunkSize = 256 - address;

        // read the chunk of data as specified
        I2Cdev::readBytes(devAddr, MPU6050_RA_MEM_R_W, chunkSize, data + i);

        // increase byte index by [chunkSize]
        i += chunkSize;

        // uint8_t automatically wraps to 0 at 256
        address += chunkSize;

        // if we aren't done, update bank (if necessary) and address
        if (i < dataSize) {
            if (address == 0) bank++;
            setMemoryBank(bank);
            setMemoryStartAddress(address);
        }
```

```
    }
}
bool MPU6050::writeMemoryBlock(const uint8_t *data, uint16_t dataSize, uint8_t bank,
uint8_t address, bool verify, bool useProgMem) {
    setMemoryBank(bank);
    setMemoryStartAddress(address);
    uint8_t chunkSize;
    uint8_t *verifyBuffer;
    uint8_t *progBuffer;
    uint16_t i;
    uint8_t j;
    if (verify) verifyBuffer = (uint8_t *)malloc(MPU6050_DMP_MEMORY_CHUNK_SIZE);
    if (useProgMem) progBuffer = (uint8_t
*)malloc(MPU6050_DMP_MEMORY_CHUNK_SIZE);
    for (i = 0; i < dataSize;) {
        // determine correct chunk size according to bank position and data size
        chunkSize = MPU6050_DMP_MEMORY_CHUNK_SIZE;

        // make sure we don't go past the data size
        if (i + chunkSize > dataSize) chunkSize = dataSize - i;

        // make sure this chunk doesn't go past the bank boundary (256 bytes)
        if (chunkSize > 256 - address) chunkSize = 256 - address;

        if (useProgMem) {
            // write the chunk of data as specified
            for (j = 0; j < chunkSize; j++) progBuffer[j] = pgm_read_byte(data + i + j);
        } else {
            // write the chunk of data as specified
            progBuffer = (uint8_t *)data + i;
        }

        I2Cdev::writeBytes(devAddr, MPU6050_RA_MEM_R_W, chunkSize, progBuffer);

        // verify data if needed
        if (verify && verifyBuffer) {
            setMemoryBank(bank);
            setMemoryStartAddress(address);
            I2Cdev::readBytes(devAddr, MPU6050_RA_MEM_R_W, chunkSize,
verifyBuffer);
            if (memcmp(progBuffer, verifyBuffer, chunkSize) != 0) {
                /*Serial.print("Block write verification error, bank ");
                Serial.print(bank, DEC);
                Serial.print(", address ");
                Serial.print(address, DEC);
                Serial.print("!\nExpected:");
                for (j = 0; j < chunkSize; j++) {
                    Serial.print(" 0x");
                    if (progBuffer[j] < 16) Serial.print("0");
                    Serial.print(progBuffer[j], HEX);
                }
```

```
            Serial.print("\nReceived:");
            for (uint8_t j = 0; j < chunkSize; j++) {
                Serial.print(" 0x");
                if (verifyBuffer[i + j] < 16) Serial.print("0");
                Serial.print(verifyBuffer[i + j], HEX);
            }
            Serial.print("\n");*/
            free(verifyBuffer);
            if (useProgMem) free(progBuffer);
            return false; // uh oh.
        }
    }

    // increase byte index by [chunkSize]
    i += chunkSize;

    // uint8_t automatically wraps to 0 at 256
    address += chunkSize;

    // if we aren't done, update bank (if necessary) and address
    if (i < dataSize) {
        if (address == 0) bank++;
        setMemoryBank(bank);
        setMemoryStartAddress(address);
    }
  }
  if (verify) free(verifyBuffer);
  if (useProgMem) free(progBuffer);
  return true;
}
bool MPU6050::writeProgMemoryBlock(const uint8_t *data, uint16_t dataSize, uint8_t
bank, uint8_t address, bool verify) {
  return writeMemoryBlock(data, dataSize, bank, address, verify, true);
}
bool MPU6050::writeDMPConfigurationSet(const uint8_t *data, uint16_t dataSize, bool
useProgMem) {
  uint8_t *progBuffer, success, special;
  uint16_t i, j;
  if (useProgMem) {
      progBuffer = (uint8_t *)malloc(8); // assume 8-byte blocks, realloc later if necessary
  }

  // config set data is a long string of blocks with the following structure:
  // [bank] [offset] [length] [byte[0], byte[1], ..., byte[length]]
  uint8_t bank, offset, length;
  for (i = 0; i < dataSize;) {
      if (useProgMem) {
          bank = pgm_read_byte(data + i++);
          offset = pgm_read_byte(data + i++);
          length = pgm_read_byte(data + i++);
      } else {
```

```
            bank = data[i++];
            offset = data[i++];
            length = data[i++];
        }

        // write data or perform special action
        if (length > 0) {
            // regular block of data to write
            /*Serial.print("Writing config block to bank ");
            Serial.print(bank);
            Serial.print(", offset ");
            Serial.print(offset);
            Serial.print(", length=");
            Serial.println(length);*/
            if (useProgMem) {
                if (sizeof(progBuffer) < length) progBuffer = (uint8_t *)realloc(progBuffer,
length);
                for (j = 0; j < length; j++) progBuffer[j] = pgm_read_byte(data + i + j);
            } else {
                progBuffer = (uint8_t *)data + i;
            }
            success = writeMemoryBlock(progBuffer, length, bank, offset, true);
            i += length;
        } else {
            // special instruction
            // NOTE: this kind of behavior (what and when to do certain things)
            // is totally undocumented. This code is in here based on observed
            // behavior only, and exactly why (or even whether) it has to be here
            // is anybody's guess for now.
            if (useProgMem) {
                special = pgm_read_byte(data + i++);
            } else {
                special = data[i++];
            }
            /*Serial.print("Special command code ");
            Serial.print(special, HEX);
            Serial.println(" found...");*/
            if (special == 0x01) {
                // enable DMP-related interrupts

                //setIntZeroMotionEnabled(true);
                //setIntFIFOBufferOverflowEnabled(true);
                //setIntDMPEnabled(true);
                I2Cdev::writeByte(devAddr, MPU6050_RA_INT_ENABLE, 0x32);  // single
operation

                success = true;
            } else {
                // unknown special command
                success = false;
            }
```

```
        }

        if (!success) {
            if (useProgMem) free(progBuffer);
            return false; // uh oh
        }
    }
    if (useProgMem) free(progBuffer);
    return true;
}
bool MPU6050::writeProgDMPConfigurationSet(const uint8_t *data, uint16_t dataSize) {
    return writeDMPConfigurationSet(data, dataSize, true);
}

// DMP_CFG_1 register

uint8_t MPU6050::getDMPConfig1() {
    I2Cdev::readByte(devAddr, MPU6050_RA_DMP_CFG_1, buffer);
    return buffer[0];
}
void MPU6050::setDMPConfig1(uint8_t config) {
    I2Cdev::writeByte(devAddr, MPU6050_RA_DMP_CFG_1, config);
}

// DMP_CFG_2 register

uint8_t MPU6050::getDMPConfig2() {
    I2Cdev::readByte(devAddr, MPU6050_RA_DMP_CFG_2, buffer);
    return buffer[0];
}
void MPU6050::setDMPConfig2(uint8_t config) {
    I2Cdev::writeByte(devAddr, MPU6050_RA_DMP_CFG_2, config);
}

//end MPU6050.cpp
////////////////////////////////////////////////////////////////////


////////////////////////////////////////////////////////////////////
//helper_3dmath.h

// I2C device class (I2Cdev) demonstration Arduino sketch for MPU6050 class, 3D math
helper
// 6/5/2012 by Jeff Rowberg <jeff@rowberg.net>
// Updates should (hopefully) always be available at https://github.com/jrowberg/i2cdevlib
//
// Changelog:
//     2012-06-05 - add 3D math helper file to DMP6 example sketch

/* ============================================
I2Cdev device library code is placed under the MIT license
```

```
#ifndef _HELPER_3DMATH_H_
#define _HELPER_3DMATH_H_

class Quaternion {
    public:
        float w;
        float x;
        float y;
        float z;

        Quaternion() {
            w = 1.0f;
            x = 0.0f;
            y = 0.0f;
            z = 0.0f;
        }

        Quaternion(float nw, float nx, float ny, float nz) {
            w = nw;
            x = nx;
            y = ny;
            z = nz;
        }
```

```
        Quaternion getProduct(Quaternion q) {
            // Quaternion multiplication is defined by:
            //     (Q1 * Q2).w = (w1w2 - x1x2 - y1y2 - z1z2)
            //     (Q1 * Q2).x = (w1x2 + x1w2 + y1z2 - z1y2)
            //     (Q1 * Q2).y = (w1y2 - x1z2 + y1w2 + z1x2)
            //     (Q1 * Q2).z = (w1z2 + x1y2 - y1x2 + z1w2
            return Quaternion(
                w*q.w - x*q.x - y*q.y - z*q.z,  // new w
                w*q.x + x*q.w + y*q.z - z*q.y,  // new x
                w*q.y - x*q.z + y*q.w + z*q.x,  // new y
                w*q.z + x*q.y - y*q.x + z*q.w); // new z
        }

        Quaternion getConjugate() {
            return Quaternion(w, -x, -y, -z);
        }

        float getMagnitude() {
            return sqrt(w*w + x*x + y*y + z*z);
        }

        void normalize() {
            float m = getMagnitude();
            w /= m;
            x /= m;
            y /= m;
            z /= m;
        }

        Quaternion getNormalized() {
            Quaternion r(w, x, y, z);
            r.normalize();
            return r;
        }
};

class VectorInt16 {
    public:
        int16_t x;
        int16_t y;
        int16_t z;

        VectorInt16() {
            x = 0;
            y = 0;
            z = 0;
        }

        VectorInt16(int16_t nx, int16_t ny, int16_t nz) {
            x = nx;
```

```
        y = ny;
        z = nz;
    }

    float getMagnitude() {
        return sqrt(x*x + y*y + z*z);
    }

    void normalize() {
        float m = getMagnitude();
        x /= m;
        y /= m;
        z /= m;
    }

    VectorInt16 getNormalized() {
        VectorInt16 r(x, y, z);
        r.normalize();
        return r;
    }

    void rotate(Quaternion *q) {
        // http://www.cprogramming.com/tutorial/3d/quaternions.html
        //
http://www.euclideanspace.com/maths/algebra/realNormedAlgebra/quaternions/transforms/index.htm
        //
http://content.gpwiki.org/index.php/OpenGL:Tutorials:Using_Quaternions_to_represent_rotation
        // ^ or:
http://webcache.googleusercontent.com/search?q=cache:xgJAp3bDNhQJ:content.gpwiki.org/index.php/OpenGL:Tutorials:Using_Quaternions_to_represent_rotation&hl=en&gl=us&strip=1

        // P_out = q * P_in * conj(q)
        // - P_out is the output vector
        // - q is the orientation quaternion
        // - P_in is the input vector (a*aReal)
        // - conj(q) is the conjugate of the orientation quaternion (q=[w,x,y,z], q*=[w,-x,-y,-z])
        Quaternion p(0, x, y, z);

        // quaternion multiplication: q * p, stored back in p
        p = q -> getProduct(p);

        // quaternion multiplication: p * conj(q), stored back in p
        p = p.getProduct(q -> getConjugate());

        // p quaternion is now [0, x', y', z']
        x = p.x;
        y = p.y;
```

```
        z = p.z;
    }

    VectorInt16 getRotated(Quaternion *q) {
        VectorInt16 r(x, y, z);
        r.rotate(q);
        return r;
    }
};

class VectorFloat {
  public:
    float x;
    float y;
    float z;

    VectorFloat() {
        x = 0;
        y = 0;
        z = 0;
    }

    VectorFloat(float nx, float ny, float nz) {
        x = nx;
        y = ny;
        z = nz;
    }

    float getMagnitude() {
        return sqrt(x*x + y*y + z*z);
    }

    void normalize() {
        float m = getMagnitude();
        x /= m;
        y /= m;
        z /= m;
    }

    VectorFloat getNormalized() {
        VectorFloat r(x, y, z);
        r.normalize();
        return r;
    }

    void rotate(Quaternion *q) {
        Quaternion p(0, x, y, z);

        // quaternion multiplication: q * p, stored back in p
        p = q -> getProduct(p);
```

```
        // quaternion multiplication: p * conj(q), stored back in p
        p = p.getProduct(q -> getConjugate());

        // p quaternion is now [0, x', y', z']
        x = p.x;
        y = p.y;
        z = p.z;
    }

    VectorFloat getRotated(Quaternion *q) {
        VectorFloat r(x, y, z);
        r.rotate(q);
        return r;
    }
};

#endif /* _HELPER_3DMATH_H_ */

//end helper_3dmath.h
////////////////////////////////////////////////////////////////////
```